SESUG 293-2019

Data Step versus Everybody: Approaching Problems as a Beginning Coder

Brian Varney, Experis Solutions

ABSTRACT

SAS has had the Data Step and Base SAS procedures since its inception. PROC SQL was added in the late 1980s and introduced an additional powerful tool for problem solving. If you are relatively new to SAS, it can be confusing to choose an approach. This paper intends to guide a beginning SAS programmer on what to use depending on the ETL and/or macro variable creation needed for your programming problem.

INTRODUCTION

This paper intends to focus on the challenges and issues that beginning coders may face when faced with writing a SAS program to solve a problem. There are a lot of situations in which we learn our coding styles and habits based on the programs we inherit to modify that were already written by other programmers. There are probably a mixture of good and not so good habits in those programs. What should your mindset be when you get to begin from scratch.

A decision often facing a programmer when solving problems is: “Which tool in SAS should I use to solve the problem?” As in any language, there are many ways to do the same thing. But if you pigeon-hole yourself to using the one tool you know best, you will face a lot of awkward coding to make the round peg fit into the square hole.

WHY THE “DATA STEP” VS EVERYBODY?

The data step is unique compared to the procedures in SAS that do similar types of processing. The data step allows a programmer to have total control over the processing that happens. Other procedures read in, process, and output the designated data based on options and statements but the programmer does not have complete control as the records pass through.

When the SQL procedure came out in the late 1980s, many programmers flocked to it as the best tool for everything. PROC SQL is a great tool to know how to use but it is not the best for every task. In most instances, PROC SQL is easier to read than the SAS Data Step and other procedures.

This paper is not implying that that any one tool is the best tool for every task. But it is good to understand all of the tools available so you can choose the one that is best for the task at hand.

WHY IS THIS IMPORTANT?

There are many attributes of a program that makes it well written. We will briefly discuss a few of them below.

1. Maintainable
    a. contains a complete and up to date header block
    b. has comments in the code to explain the intent of your code and anything that should be explained
    c. is well organized and uses descriptive data set and variable names
2. Accurate

    a. the actual results match the expected results

3. Robust
    a. data driven techniques are used
    b. there is not hard coding to get the program to work
4. Efficient
    a. reads in only the records and variables that are needed for the program
    b. does not read through data sets more times that necessary
    c. allow databases to do filtering and sorting before the data is pulled into SAS
    d. uses the right tool for the necessary processing

The last bullet is what this paper intends to address. It is important to use the right tool for the task at hand. If you use the wrong tool for the job, it will increase the time to complete the program, make it more difficult to maintain, and possibly less efficient.

We will now look at some common tasks that need to be addressed in programs and then we will look at some problem patterns that lend themselves to some tools over others.

COMMON TASKS

This section will focus on tasks that are very common in programs. In fact, it would be difficult to write a program without using one or all of the processing tasks that are covered here.

CHANGING THE ATTRIBUTES OF A SAS DATA SET VARIABLE

There are many programmers out there that will use a SAS Data Step to change the attributes of a SAS Data Set variable. This is very inefficient if that is the only task that is carried out in that SAS Data Step. The SAS Data Step reads through the entire SAS Data Set record by record and that is not necessary if one is only changing the attributes of a SAS Data Step Variable. While you cannot change the length or type of a SAS Data Set variable, there are other attributes that can be easily handled by PROC DATASETS.

Please see the example below.

```
data class;
   set class;
   rename name=student_name;
   format age words.;
   label height="Student's Height";
run;


proc datasets lib=work nolist;
   modify class;
      rename name=student_name;
        format age words.;
        label height="Student's Height";
quit;
```

Please note that the SAS Data Step must entirely re-create the SAS Data Set while PROC DATASETS only updates the SAS Data Set header. Even if the SAS Data Set above has millions of records, the PROC DATA SETS will run instantaneously while the SAS Data Step would take significantly longer because it has to re-write the entire SAS Data Set.

CALCULATING NEW VARIABLES

Here we will only focus on the SAS Data Step and PROC SQL. We are intending to compare these two methods from an ETL sense.

The PROC SQL and the SAS Data Step code below, show how to calculate new variables for the following scenarios:

- setting a new numeric constant (YEAR)
- setting a new character constant (CLASSYEAR)
- calculating a new character variable (NAMESEX)
- calculating a new numeric variable (DOGAGE)
- calculating using conditional logic (AGE_CAT and GENDER_CAT)

```
proc sql number;
   create table classof2019 as
   select 2019 as year,
          "Class of 2019" as classyear,
          name, age,
          trim(name)||","||sex as namesex,
          age*7 as dogage,
          case
             when age=. then "Miss"
             when age < 14 then "Young"
             else "Old"
             end as age_cat label="Age Category" length=20,
          sex,
          case sex
             when " " then "Unsure"
             when "M" then "Mars"
             else "Venus"
             end as gender_cat
   from sashelp.class;
quit;
```

```
data classof2019;
   set sashelp.class(keep=name sex age);
   length age_cat $5 gender_cat $6;
   year=2019;
   classyear="Class of 2019";
   namesex=trim(name)||","||sex;
   dogage=age*7;
   if age=. then age_cat="Miss";
   else if age < 14 then age_cat="Young";
   else age_cat="Old";
   select (sex);
```

```
      when(" ") gender_cat="Unsure";
        when("M") gender_cat="Mars";
        otherwise gender_cat="Venus";
   end;
run;
```

For normal variable calculations like above, it really does not matter whether you use the SAS Data Step or PROC SQL. It is more of a matter of personal preference and the possibly the personal preference of the person that will have to maintain the code moving forward.

CREATING SIMPLE SUMMARY STATISTICS

In this section we are comparing SAS Data Step vs PROC SQL vs PROC SUMMARY/MEANS/UNIVARIATE. We compute the summary statistics and then sort by the AVEAGE.

Simple Summary Statistics Using the SAS Data Step

```
proc sort data=sashelp.class out=class;
   by sex;
run;

data sss(drop=age);
   retain minage maxage;
   set class(keep=sex age);
   by sex;
   if first.sex then
   do;
      n=0;
        nmiss=0;
        totage=0;
        aveage=0;
        minage=age;
        maxage=age;
   end;
   if age ne . then
      n+1;
   if age eq . then
      nmiss+1;
   totage+age;
   minage=min(age,minage);
   maxage=max(age,maxage);
   if last.sex then
   do;
      aveage=totage/n;
      output;
   end;;
run;

proc sort data=sss;
   by aveage;
run;

proc print data=sss;
run;
```

Simple Summary Statistics Using PROC SQL

```
proc sql;
   create table sss as
   select sex,
          n(age) as n,
          nmiss(age) as nmiss,
          sum(age) as totage ,
          mean(age) as aveage ,
          min(age) as minage ,
          max(age) as maxage
   from sashelp.class
   group by sex
   order by aveage;

   select * from sss;
quit;
```

Simple Summary Statistics Using PROC SUMMARY/MEANS/UNIVARIATE

```
proc summary nway data=sashelp.class(keep=sex age) n nmiss mean sum min max;
   class sex;
   var age;
   output out=sss n=n nmiss=nmiss mean=mean sum=sum min=minage max=maxage;
run;

proc sort data=sss;
   by aveage;
run;

proc print data=sss;
run;
```

As you can see above, all three are feasible. But unless you need to do some type of custom rule that requires conditional, record level, etc. type processing, you would not use the data step. It is too much manual calculation coding plus you have to sort before-hand and afterwards using PROC SORT. For summary statistics that require multiple passes through the data such as standard error, the SAS Data Step should be avoided as the PROCs offer much easier and less error prone coding.

COMBINING DATA SETS

In this section, we will address two scenarios for combining SAS Data Sets.

1. Stacking – same variables additional rows
2. Merging/Joining – common key variables to match on to add variables

SAS Data Step SET vs PROC SQL OUTER UNION CORRESPONDING vs PROC APPEND

First we will be stacking SAS Data Sets. The assumption is that the SAS Data Sets have the same variables but we are just adding records from additional SAS Data Sets.

Example Data – four SAS Data Sets

```
data class_k class_1st class_2nd class_3rd;
   set sashelp.class;
run;
```

SAS Data Step SET Statement

```
data all_classes;
   set class_k class_1st class_2nd class_3rd;
run;
```

PROC SQL OUTER UNION CORRESPONDING

```
proc sql;
   create table all_classes as
   select *
   from class_k
outer union corresponding
   select *
   from class_1st
outer union corresponding
   select *
   from class_2nd
outer union corresponding
   select *
   from class_3rd;

QUIT;
```

PROC APPEND

```
proc append base=all_classes data=class_k;
run;
proc append base=all_classes data=class_1st;
run;
proc append base=all_classes data=class_2nd;
run;
proc append base=all_classes data=class_3rd;
run;
```

SAS Data Step MERGE vs PROC SQL JOIN

The second will be match merging/joining. The assumption is that two or more SAS Data Sets have common key variables that are appropriate for match merging/joining the tables.

Example Data – Two SAS Data Sets with the variable PATIENT as the match key.

```
data demo;
   input patient sex $ race $ age;
   cards;
1 M 5K 29
```

```
2 F Marathon 34
3 M 5K 39
5 F 10K 25
run;

data vitals;
   input patient visit pulse @@;
   cards;
1 1 100 1 2 120 1 3 99
2 1 101 2 2 110 2 3 91
3 1 97 3 2 119 3 3 95
6 1 91 6 2 118
run;
```

INNER JOIN - Patient’s Must Appear in Both DEMO and VITALS

SAS Data Step MERGE to do INNER JOIN

```
data both;
   merge demo(in=d) vitals(in=v);
   by patient;
   if d and v;
run;
```

PROC SQL – Two different methods for specifying (different syntax)

```
proc sql;
   create table innerdv1 as
   select demo.*,vitals.visit,vitals.pulse
   from demo,
        vitals
   where demo.patient=vitals.patient;
quit;
```

OR

```
proc sql;
   create table innerdv2 as
   select demo.*,vitals.visit,vitals.pulse
   from demo   inner join
        vitals
   on (demo.patient=vitals.patient);
quit;
```

LEFT JOIN - Patient’s Must Appear in DEMO

SAS Data Step MERGE to do LEFT JOIN

```
data both;
   merge demo(in=d) vitals;
   by patient;
   if d;
run;
```

PROC SQL

```
proc sql;
   create table leftdv as
   select demo.*,vitals.visit,vitals.pulse
   from demo   left join    vitals
   on demo.patient=vitals.patient;
quit;
```

RIGHT JOIN - Patient's Must Appear in VITALS

SAS Data Step MERGE to do RIGHT JOIN

```
data both;
   merge demo vitals(in=v);
   by patient;
   if v;
run;
```

PROC SQL

```
proc sql;
   create table rightdv as
   select vitals.*,demo.sex,demo.race,demo.age
   from demo   right join    vitals
   on demo.patient=vitals.patient;
quit;
```

FULL JOIN - Patient's Can Appear in either data set

SAS Data Step MERGE to do FULL Join

```
data both;
   merge demo vitals;
   by patient;
run;
```

PROC SQL

```
proc sql;
   create table fulldv as
   select coalesce(demo.patient,vitals.patient) as patient,
          demo.sex,
          demo.race,
          demo.age,
          vitals.visit,
          vitals.pulse
   from demo   full join
         vitals
   on demo.patient=vitals.patient;

title1 "Full Join Example";
   select * from fulldv;
quit;
```

CREATING SAS MACRO VARIABLES

In this section, we will address methods for creating SAS Macro variables.

List of Values in a Single SAS Macro Variable

SAS Data Step – CALL SYMPUT

```
proc sort data=sashelp.class out=class;
   by sex;
run;

data _null_;
   length list $10;
   retain list ' ';
   set class end=eof;
   by sex;
   if last.sex;
   count+1;
   if count eq 1 then
   do;
      list=sex;
   end;
   else do;
      list=trim(left(list))||'", "'||sex;
   end;
   if eof then
   do;
      call symput('sexex',list);
      call symput('numsex',trim(left(put(count,8.))));
   end;
run;
```

PROC SQL

```
proc sql noprint;
   select distinct sex into :sexes separated by '", "'
   from sashelp.class;
quit;
%let numsex=&sqlobs.;
```

List of Values in an Array of SAS Macro Variables

SAS Data Step – CALL SYMPUT

```
data _null_;
   set class;
   by sex;
   if last.sex;
   count+1;
   call symput('sex'||trim(left(put(count,8.))),sex);
   call symput('numsex',trim(left(put(count,8.))));
run;
```

PROC SQL

```
proc sql noprint;
   select sex into :sex1 - :sex9999
   from sashelp.class;
quit;
%let numsex=&sqlobs.;
```

APPROACHING PROBLEMS WITH CERTAIN PROCESSING PATTERNS

In this paper, we will be discussing different types of problems and then suggesting the best tool for the job. The tool may be a data step or a procedure such as SQL, MEANS, TRANSPOSE, FREQ, …, etc.

PROBLEM PATTERN #1:
PERFORMING THE SAME OPERATION ON MANY VARIABLES

This problem pattern commonly comes up in programming. I have a large number of variables of the same type and I want to perform the same function or calculation on each one.

Scenario – I want to convert all of the character variables in SASHELP.CLASS to lower case.

```
proc sql;
   create table class_lower as
   select lowcase(name) as name,
          lowcase(sex) as sex,
              age, weight, height
   from sashelp.class;
quit;


data class_lower(drop=i);
   set sashelp.class;
   array allchars(*) _character_;
   do i=1 to dim(allchars);
      allchar{i}=lowcase(allchar{i});
   end;
run;
```

Now imagine if there were hundreds of character variables in the SAS Data Set. We would not have to change the SAS Data Step code at all!

Scenario – I want to convert missing values to zero for all numeric variables in SASHELP.CLASS.

```
proc sql;
   create table class_missing_with_zero as
   select name as name,
          sex as sex,
              case
                 when age=. then 0
                   else age
                   end as age,
              case
                 when height=. then 0
                   else height
                   end as height,
              case
```

```
                    when weight=. then 0
                      else weight
                      end as weight
   from sashelp.class;
quit;


data class_missing_with_zero(drop=i);
   set sashelp.class;
   array allnums(*) _numeric_;
   do i=1 to dim(allnums);
      if allnums{i}=. then
         allnums{i}=0;
   end;
run;
```

Now imagine if there were hundreds of numeric variables in the SAS Data Set. We would not have to change the SAS Data Step code at all!

PROBLEM PATTERN #2:
USING VALUES FROM PREVIOUS RECORDS

Since the SAS Data Step is a looping mechanism and steps through the data record by record in sequential order, there is a lag function that is available to use to retrieve a value from the previous record.

```
data lags;
   input id date date9. value;
cards;
1 09SEP2019 23
2 10SEP2019 45
3 11SEP2019 .
4 12SEP2019 9
run;

data getpast;
   set lags;
   lagvalue=lag(value);
   if value=. then
      value=lagvalue;
run;
```

The lag function is not available in PROC SQL. To try and build queries to mimic the lag function is not worth attempting.

CONCLUSIONS

There is not a clear and consistent go to tool for every data processing task. However, knowing multiple tools such as the SAS Data Step, PROC SQL, and other SAS procedures will definitely make you a much more effective and efficient problem solver.

CONTACT INFORMATION

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

Brian Varney
Experis Solutions
269-553-5185
brian.varney@experis.com