Paper 200-2009

PROC SQL and SAS® Macro: Beyond the Basics

Louie Huang, Baxter Healthcare Corporation, Westlake Village, CA
Norma Guzman-Becerra, Baxter Healthcare Corporation, Westlake Village, CA

ABSTRACT

SAS macro facility has been a very important tool in SAS programming for many years. The CALL SYMPUT routine and DATA _NULL_ are the traditional methods to create macro variables from SAS data. However, PROC SQL is much more powerful and efficient in creating macro variables thanks to the in-line view capability and the SELECT, INTO, GROUP BY, HAVING, and ORDER BY clauses. In clinical programming practice, PROC SQL can solve a lot of programming problems, sometimes it is even impossible to solve a problem without using PROC SQL. The purpose of this paper is to demonstrate creation of macro variables from SAS data using PROC SQL. Concrete examples are provided to illustrate the advantages of PROC SQL in creating macro variables over the CALL SYMPUT routine.

INTRODUCTION

You often want to manipulate your SAS data and convert a list of unique values of one variable, or the unique combinations of several variables by concatenation, or a list of unique variables of one data file, or even a list of unique data file names of the entire library into macro variables in order to efficiently run and maintain your SAS programs. PROC SQL offers a lot of useful features, which includes, but is not limited to: 1) combine the functionality of DATA and PROC steps into one single step, 2) sort, summarize, join (merge) and concatenate datasets, 3) construct in-line views using the FROM and SELECT clauses, 4) line up multiple macro variables using the INTO clause. There are many efficient ways to create macro variables using PROC SQL. This presentation will demonstrate the useful tricks and skills by a few practical examples.

EXAMPLE ONE

For example, you are asked to generate multiple graphs for the surgeons who have 100 or more subjects. Your graphs will be generated from multiple SAS data files. The common variable in these data files is the surgeon ID. However, the subject information is only available from the ***demog*** data file. In order to avoid tedious coding, dynamically control your output, and minimize your workload on each revision, it is helpful to create the macro variables found in Table 1 by using the SAS code listed in Figure 1.

Table 1. List of the macro variable examples

Macro	Value
&SURGLIST	SB67#SB17#SB22#SB35#SD01#SD03#SD41#SD15#SB19#SD38#SB20#SA32#SD33#SB77#SB63#SB93
&QUOTED	"SB67" "SB17" "SB22" "SB35" "SD01" "SD03" "SD41" "SD15" "SB19" "SD38" "SB20" "SA32" "SD33" "SB77" "SB63" "SB93"
&PTNUMLST	731#375#373#327#325#283#232#229#158#151#149#144#138#129#121#103
&CT	16

Figure 1. SAS Coding Used to Creating the Macro Variables displayed in Table 1: Comparison of PROC SQL and CALL SYMPUT in DATA _NULL_

PROC SQL

```
proc sql noprint;
  select surgeon,obs,
  quote(trim(surgeon)),n(surgeon)
  into :surglist separated by '#',
       :ptnumlst separated by '#',
       :quoted separated by ' ',:ct
  from (select surgeon,count(pt) as obs
  from sugi.demg
  group by 1
  having calculated obs >= 100)
  order by obs desc;
quit;
```

CALL SYMPUT and DATA _NULL_

```
proc freq data=sugi.demg noprint;
  tables surgeon / out=surge_ID(drop=percent) nocum;
run;

proc sort data=surge_ID(where=(count>=100));
  by descending count;
run;

data _null_;
  length list1-list3 $120;
  retain list1-list3 ' ';
  set surge_ID end=last;
   if _n_=1 then sp=' ';
   else sp='#'; /* sp separates macro variables */
 list1=trim(left(list1))||sp||left(surgeon);
 list2=trim(left(list2))||sp||left(put(count,3.));
 list3=trim(left(list3))||sp||
       left(quote(trim(surgeon)));
  if last then do;
  call symput('surglist',list1);
  call symput('ptnumlst',list2);
  call symput('quoted',list3);
  call symput('ct',_n_);
end;
run;
```

Notice that in Table 1 and Figure 1, the obvious advantages of PROC SQL are : 1) multiple macro variables are created with one step; 2) your coding is significantly shorter; 3) the data values are summarized by an in-line view (highlighted with yellow color) in PROC SQL. You have to rely on one PROC FREQ and one PROC SORT to summarize the data if you use the CALL SYMPUT routine. In addition, the DATA _NULL_ STEP requires more coding to create the same types of macro variables.

EXAMPLE TWO

It is very common that your clients ask you to add some summarized information into a well-refined table or graph in order to make the presentation more informative. Of course, your favorite tools are macro variables, because you can conveniently display the information in a footnote or title by revoking macro variables. As presented in Figure 2, two macro variables are created for the surgeons who have the most and least patients. You have to count the number of patients by surgeon and identify the maximum and minimum counts before you make the macro variables. PROC SQL can perform a very nice job with a few lines of coding only. However, CALL SYMPUT routine requires two SORT procedures and two DATA STEPS (Figure 2).

Figure 2. Convert the Numbers of most and least patients into macro variables

PROC SQL

```
proc sql noprint;
  select max(ptnum),min(ptnum)
     into :maxnum,:minnum
    from
  (select surgeon,count(pt) as ptnum
    from sugi.demg
  group by 1);
quit;
```

CALL SYMPUT and DATA _NULL_

```
proc sort data=sugi.demg out=demg;
  by surgeon;
run;

data counted(keep=surgeon ptnum);
  set demg;
  by surgeon;
  retain ptnum;
  if first.surgeon then ptnum=0;
  ptnum+1;
  if last.surgeon;
run;

proc sort data=counted;
  by descending ptnum;
run;

data _null_;
set counted end=last;
 if _n_=1 then
 call symput('maxnum',left(ptnum));
 if last then
 call symput('minnum',left(ptnum));
run;
```

EXAMPLE THREE

The SAS code displayed in Figure 3 can create five sets of macro variables, which are used to create patient profiles for 53 subjects. The treatment start and end dates vary from subject to subject. Therefore, you have to convert the date values into macro variables. Set one (&pt1 to &pt53) contains Subject Identifications. Set two (&bday1 to &bday53) contains the value of Treatment Start Date. Set three (&eday1 to ebday53) contains the value of Treatment End Date. The information for each subject displays within each profile by creating an annotation dataset. The other two sets of macro variables specify the tick marks of the horizontal axis by 30 days. Set four (&bdy1 to &bdy53) contains the value of Treatment Start Date minus 3 days. The reason of doing this is to allow the first date value a little away from the vertical axis. Set five (&edy1 to &edy53) contains the value of Treatment End Date plus 54 days. The reason of doing this is to ensure the Treatment End Date included in the horizontal axis.

As you review the code in Figure 3, you will realize that the value calculation and grouping in PROC SQL is completed by a short nested query (marked with yellow color). However, you need a PROC MEAN and a DATA STEP to perform the same task when you use CALL SYMPUT routine to create the macro variables. Furthermore, the code used to create the macro variables in the DATA _NULL step is more complicated.

Figure 3. Comparison of SAS Code in Creating Multiple Sets of Macro Variables (PROC SQL and CALL SYMPUT)

PROC SQL

```
proc sql noprint;
  select pt,firstday format=date9.,
 lastday format=date9.,firstday2,lastday2
  into :pt1-:pt53,:bday1-:bday53,
  :eday1-:eday53, :bdy1-:bdy53,:edy1-:edy53
     from (select pt,min(txdt) as
    firstday,max(txdt) as lastday,
    min(txdt)-3 as  firstday2,
    max(txdt)+54 as lastday2
  from data180
  group by pt);
quit;
```

CALL SYMPUT and DATA _NULL_

```
proc means data=data180 noprint
           maxdec=2 nway;
  class pt;
  var txdt;
  output out=data1(drop=_type_ _freq_)
        min=firstday max=lastday;
run;

data data2;
  set data1;
  firstday2=firstday-3;
  lastday2=lastday+54;
run;

data _null_;
  set data2 end=last;
  by pt;
  if last then call symput('cnt',_n_);
  call symput(compress('pt'||_n_),pt);
  call symput
(compress('bday'||_n_),put(firstday,date9.));
  call symput
(compress('eday'||_n_),put(lastday,date9.));
call symput (compress('bdy'||_n_),firstday2);
call symput(compress('edy'||_n_),lastday2);
run;
```

EXAMPLE FOUR

This is an example on efficiency of PROC SQL in creating macro variables other than the advantage of shortening the code. These two macro variables are used for two different reasons. Macro ***trtdt2*** is used to specify the tick marks of the horizontal axis by day. However, macro ***trtdt*** is used to label the tick marks on the horizontal axis. The SAS code used to create these two macro variables with PROC SQL and CALL SYMPUT routine is shown in Figure 4.

Table 2. List of macro *trtdt* and macro *trtdt2*

Macro ***trtdt***:

```
"30MAR03" "31MAR03" "01APR03" "02APR03" "03APR03" "04APR03" "05APR03" "06APR03"
"07APR03" "08APR03" "09APR03" "10APR03" "11APR03" "12APR03" "13APR03" "14APR03"
"15APR03" "16APR03" "17APR03" "18APR03"
```

Macro ***trtdt2***:

```
"30MAR03"d "31MAR03"d "01APR03"d "02APR03"d "03APR03"d "04APR03"d "05APR03"d
"06APR03"d "07APR03"d "08APR03"d "09APR03"d "10APR03"d "11APR03"d "12APR03"d
"13APR03"d "14APR03"d "15APR03"d "16APR03"d "17APR03"d "18APR03"d
```

Figure 4. Example of Programming Efficiency Other than Shortening SAS Code

PROC SQL

```
proc sql noprint;
  select quote(put(trt_dt,date7.)),
         quote(put(trt_dt,date7.))||'d'
         into :trtdt separated by ' ',
         :trtdt2 separated by ' '
  from
  (select distinct trt_dt from sugi.try3);
quit;
```

CALL SYMPUT and DATA _NULL_

```
proc sort data=sugi.try3 out=sorted
          nodupkey;
  by trt_dt;
run;

data _null_;
  length listed1-listed2 $200;
  retain listed1-listed2 ' ';
  set sorted end=eof;
  by trt_dt;
  listed1=trim(left(listed1))||' '||
          left(quote(put(trt_dt,date7.)));
  listed2=trim(left(listed2))||' '||
     left(quote(put(trt_dt,date7.))||'d');
  if eof then do;
  call symput('trtdt',listed1);
  call symput('trtdt2',listed2);
    end;
run;
```

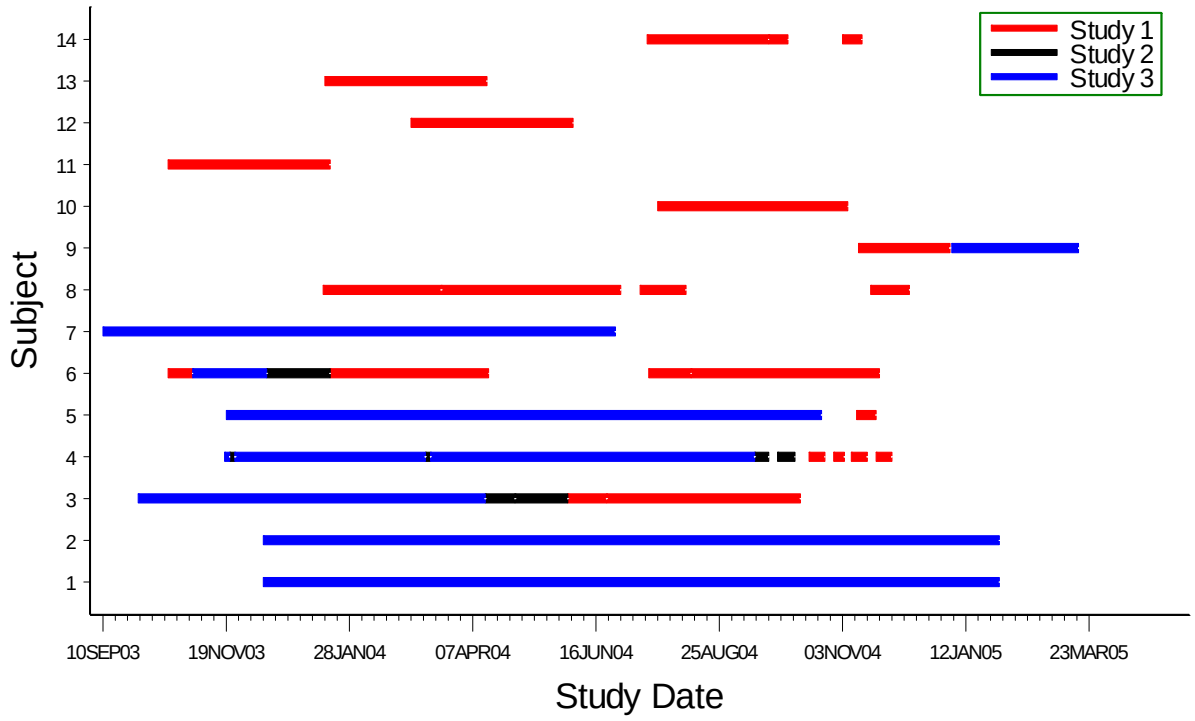
Let us start with the DATA _NULL_ step and CALL SYMPUT routine. You need a PROC SORT procedure to sort the data. But, this is not the only shortcoming. First of all, the length of the dummy variables ***listed1*** and ***listed2*** is specified at $200 by guessing. When you check the log window, you are surprised because of missing the macro variable ***trt_dt2***. There are no error and warning messages available in the log window. The warning and error messages display in the log window only if you execute the code one more time. The messages say "WARNING: The quoted string currently being processed has become more than 262 characters long. You may have unbalanced quotation marks." and "ERROR: Open code statement recursion detected.". This tells you that the actual length of the character string for the dummy variable ***listed2*** is greater than 200 characters long, and the unbalanced quotation marks are generated from the truncation of the character string between the two quotation marks. You can figure out the actual length by checking the length of one quoted treatment date. Alternatively, you can avoid the problem by generously setting the length, i.e., setting the length of listed1and listed2 at $1000. Unfortunaltely, this lesson is expensive for you will not realize it until a number of trials. Please understand that the length statement in the DATA _NULL_ step can't be omitted. Otherwise the dummy variables ***listed1*** and ***listed2*** would become blanks.

However, you can easily create these two long macro variables with PROC SQL without having to worry about the actual length of the clustered text. You do not have to experience the problems discussed above.

EXAMPLE FIVE

One day, your boss came to you with the output of Figure 5. He said " This is a nice graph. However, if you could display the real Subject ID along the y-axis and the total study days at the end of last study for each subject it would be nicer."

Figure 5 Subject study path without subject IDS and total study days



You can accomplish the task very easily. First of all, you convert the values stored in the data file to macro variables by using the following code:

```
proc sql noprint;
 select
quote(put(subjid,z6.))
     ,totdays
     ,date_out
   into :sublist2 separated by ' '
         ,:totday separated by ' '
             ,:date_out separated by ' '
   from
  (select subjid,sum(days) as totdays,max(date_out) as date_out
  from
  time
  group by 1)
  order by 2 desc;
quit;
```

You do not have to type the assigned subject numbers and subject identifications in a user-format. You can conveniently turn the assigned number into subject ID with the following SAS code:

```
proc format;
  value subj
 %macro fmt;
  %do i=1 %to &sqlobs;
  &i=%scan(&sublist2,&i)
  %end;
;
%mend;
%fmt
run;
```

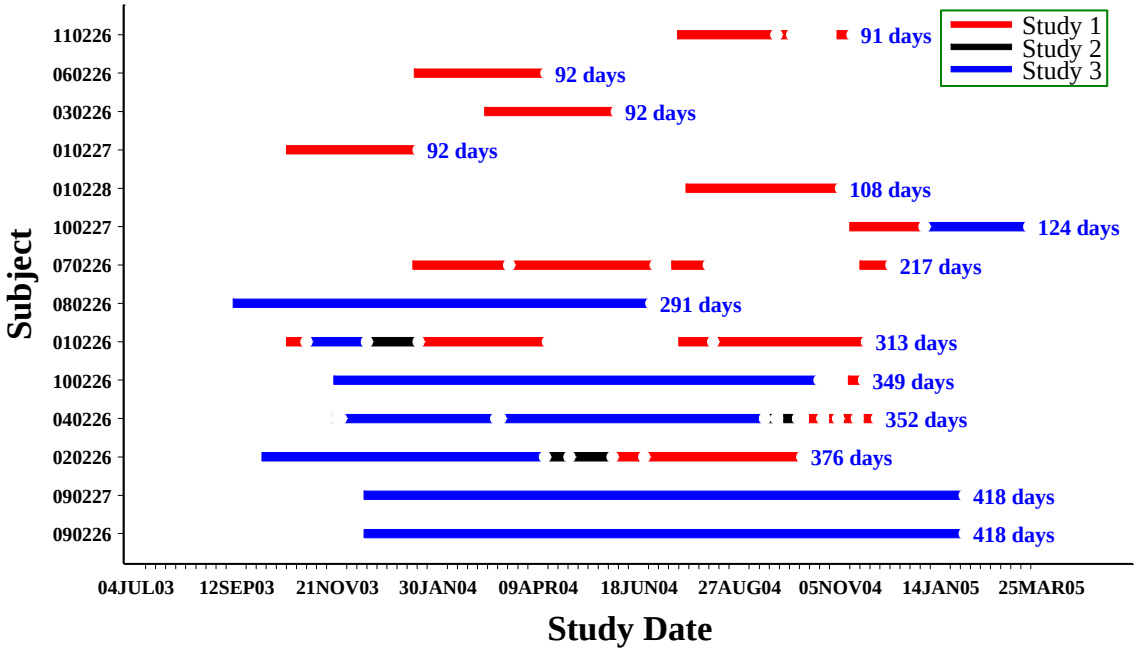
In order to label the total study days at the end of study, You can simply build the macro values into an annotate dataset by using the SAS code as follows:

```
data anot;
    length color $6.;
    retain xsys ysys '2' hsys '1' when 'a' position '6' size 3 function 'label' color 'blue';
  %macro lbt;
   %do i=1 %to &cnt;
   %let mdt=%scan(&mdtlist,&i);
   %let days=%scan(&totday,&i);
   x=&mdt+6; y=&i;
   text=catx(' ',"&days",'days');
output;
%end;
%mend;
%lbt
run;
```

Finally, you place the user-format subj and annotate dataset anot into the GPLOT procedure, the graph output will be what your boss expects.

```
proc gplot data=sorted anno=anot2;
    format date_out date7. subnum subj. ;
   plot subnum*date_out / anno=_anot noframe
                     haxis=axis1 hminor=9
                     vaxis=axis2 vminor=0
                     caxis=black nolegend;
run;
quit;
```

Figure 6. Subject study path with subject IDS and total study days



CONCLUSION

PROC SQL is a great joy of SAS programming. This paper only demonstrates a few practical examples of creating macro variables with PROC SQL. These examples present great flexibility and useful skills which offer some hints for you to create macro variables in your real work.

CONTACT INFORMATION

You can send your comments, questions, and/or inquiries to:
Louie Huang, Senior Technical Specialist
Norma Guzman-Becerra, Bio-Statistician
Baxter BioScience, Baxter Healthcare Corporation
One Baxter Way
Westlake Village, CA 91362
Tel: 805-372-3487 (Louie), 805-372-3009 (Norma)
Fax: 805-372-3462
Email: louie_huang@baxter.com, norma_guzman_becerra@baxter.com

TRADEMARK INFORMATION