PAPER 031-30

WHAT WOULD I DO WITHOUT PROC SQL AND THE MACRO LANGUAGE

Jeff Abolafia, Rho, Inc., Chapel Hill, NC

Introduction

The SAS® macro language used in conjunction with PROC SQL provides the programmer a powerful set of tools for building effective and efficient applications. These tools can be utilized for a wide range of applications only limited by one's imagination. In this paper, we present a series of examples taken from several real world applications. These examples demonstrate a more efficient approach to common tasks using the macro language in combination with PROC SQL.

Using PROC SQL with the Macro Language

Most SAS users would agree that PROC SQL is an effective tool for building applications. Combining PROC SQL with the macro language interface increases the functionality of PROC SQL and allows us to build more robust and efficient applications. PROC SQL provides the INTO and SEPARATED BY statements for interfacing with the Macro language. The INTO option for the SELECT statement can store the value of a variable, constant, calculation, or expression in one or more macro variables. The SEPARATED BY option allows the user to assign all of the values of an entire column to a macro variable. A delimiter chosen by the user separates values. For a more complete description of these two statements see SAS Macro Language: Reference, First Edition (1997).

Building applications using PROC SQL and the macro language offers several efficiencies compared with alternative approaches: 1) in a single PROC you can obtain summary statistics and store them in one or more macro variables; 2) the code to store the contents of an entire column in a macro variable is less complex than in a data step; 3) Dictionary tables can only be accessed by PROC SQL (dictionary views can be accessed without PROC SQL); 4) in many cases, SAS code can be generated more efficiently and dynamically than in a data step.

Example 1: Storing Ns in Macro Variables

When producing data displays, we typically report the number of observations in various subgroups or the total number of observations. These statistics are usually displayed in titles, footnotes, or columns headers. The traditional approach is to execute PROC MEANS to get observations counts and then to use a data step with CALL SYMPUT to store the observation counts in macro variables. Example one shows a more efficient approach using PROC SQL and the macro language.

Example 1

```
proc sql noprint;
  select n(drug) into :n1 - :n2
  from anal
  group by drug    ;
quit;

proc format ;
   value $cft
      1 = "Drug    (N = &n1) "
      2 = "Placebo (N = &n2) "
      ;
 Run;
```

The SQL code in Example 1 uses the SELECT and INTO statements to store the number of observations in each of the two drug groups in two macro variables. These macro variables are then used in a subsequent step to display the number of observations in column headers.

Example 2: Storing Counts in Macro Variables

Example 2 extends the concepts presented in Example 1. While Example 1 shows how to store frequency counts of a given variable into macro variables, Example 2 demonstrates how to store counts of observations meeting any criteria in one or more macro variables. In this example we use the SUM function to count the number of males where Drug=1, Drug=2, and Drug = 1 or 2. These counts are then stored in three macro variables using the INTO statement.

Example 2

```
proc sql noprint;
  select sum(drug=1),sum(drug=2),sum(1<=drug<=2) into :n1,:n2,:n3
  from anal
  where gender='Male'  ;
quit;
```

Example 3: Print from Every Dataset in a Library

There is often a need to print selected records from every dataset in a library. For example we may want to produce a data dump of all subjects with cardiovascular disease. One approach would be to first run a program to output subjects with cardiovascular disease (CVD). Then merge this file with each dataset in our library, keeping only subjects with cardiovascular disease and then running a PROC PRINT on each of these datasets. While this works, it is extremely cumbersome and requires *a priori* knowledge of all data set names in our library. Example 3 demonstrates a more efficient and dynamic approach.

PROC SQL is used to create two lists that are stored in macro variables. Macro variable IDS is created from the dataset containing CVD information and contains a comma-separated list of subjects with a history of CVD. This list is used to filter observations to PROC PRINT. The second macro variable, DSNS, is created from DICTIONARY.TABLES and contains a list of datasets in our library. Once these two macro variables are created, macro PRINTIT can be used to list subjects in each dataset with CVD.

Example 3 demonstrates one of the strengths of using SQL with the macro language – the ability to easily store lists or the contents of an entire column in a macro variable. In this example we store both the list of IDS and the list of datasets in macro variables. In each case this is done with a single statement. The code using this method is much less complex than the code using a Data step. To accomplish the same task with a Data step we would have to: 1) create a new variable; 2) use the RETAIN statement to declare the variable a retained variable; 3) use an assignment statement to load the contents of a column into the retained variable; 4) use the END= dataset option to identify the last record; and 5) at the last record use CALL SYMPUT to load the contents of the retained variable into a macro variable.

Example 3

```
libname VACC 'S:\RHO\VASOCOR\VVS_Acc\data\crf' ;

proc sql noprint;
     select quote(trim(id))
       into :ids separated by ','
       from vacc.riskmstr

 where CVDYN='Y' ;
```

```
        select memname into :dsns separated by ' '
        from dictionary.tables
       where libname="VACC"
      ;
quit;

%put IDs with CVD are: &ids ;
%put datasets in VACC are: &dsns ;

/* SAS LOG
IDs with CVD are:
"1581","1587","1588","1600","1606","1619","1621","2522","2533","2558", "2571","2577"

dataets in VACC are: CHEMMSTR CMEDMSTR DEMOMSTR ECG1MSTR ECHOMSTR EXERMSTR

END of SAS LOG
*/


%Macro printit ;
       %let i =1 ;
       %do %until (%scan(&dsns,&i)= ) ;
           %let dsn = %scan(&dsns,&i);
           proc print data= Vacc.&dsn ;
             where id in(&ids) ;
             title2 "Records with CVD from &DSN.";
           run;
           %let i = %eval(&i + 1);
       %end;
%mend;
```

Example 4: Creating SAS Code Part 1

In large and complex projects variable and dataset attributes are often stored as metadata. When creating SAS datasets, labels are harvested from the metadata. The traditional approach was to read the metadata containing the labels for each dataset, write one external file for each dataset containing the label statements, and then use a %INCLUDE statement in a DATA step or PROC DATASETS to create labels for dataset variables. Using this approach, if a project contains 50 datasets, one must write and then read 50 external files to capture the labels (or any other attribute) for each dataset. Example 4 presents a more efficient approach and also demonstrates how PROC SQL combined with the macro language can be used to dynamically create SAS code.

The SQL code in example 4 uses the SELECT statement to create the label statement for each variable in a dataset. The SELECT statement concatenates literals containing SAS statements and variables containing variable names and their associated label. Next the INTO and SEPARATED BY statements are used to store label statements for all variables in a single macro variable. Finally PROC DATASETS is used to attach the labels to their associated variables. This example shows how to provide labels for a single dataset. This code can be placed in a loop as in the previous example to provide labels for every dataset in a project library. Also, this example focuses on labels, but this approach can also be used to provide formats or to rename variables.

Example 4

```
proc sql noprint;
 select catx('',name,'="',label,'"')  into :labs separated by ' '
 from md.variables
 where dataset="&dsn"  ;
quit ;

%put labels for &dsn are: &labs ;

/* SAS LOG
```

```
   labels for DEMO are:
  AGE="Age" BIRTHDT="Date of Birth" DIAB_ST="Diabetes Mellitus"
  INFDT="Informed Consent Date" LOCKFLAG="Data Lock Flag"  RACE="Race" SEX="Sex"
  STUDY="Clinical Study"  VISIT="Visit"

   END of SAS LOG
   */
  proc datasets library=dm nolist ;
     modify &dsn ;
     label &labs ;
   quit ;
```

Example 5: Creating SAS Code Part 2

When creating SAS datasets, variables often must be rearranged in a specific order. Typically a set of key or identifier variables is placed before other dataset variables. You can reorder variables in an existing dataset by placing a LENGTH statement before a SET (or MERGE) statement in a data step. However writing out a LENGTH statement can be cumbersome and time consuming, especially for large datasets. Example 5 uses Dictionary Tables, SQL, and the macro language to generate the LENGTH statement for an existing SAS dataset.

As in the previous example, the SELECT statement is used to concatenate the variable name with its associated type and length. Then the INTO and SEPARATED BY statements are used to store the LENGTH statements for all variables in a single macro variable. The contents of the macro variable are then written to the log. Next you can cut/paste the LENGTH statement from the log into the program editor and edit the LENGTH to achieve the desired variable order.

Example 5

```
%macro getlen(libname=,dsn=) ;

  %let libname=%upcase(&libname) ;
  %let dsn = %upcase(&dsn) ;

    proc format ;
    value $tft
          'char'='$'
          other =' ' ;
    run;

  proc sql noprint ;
   select  catx(' ',name,put(type,$tft1.),put(length,5.))
         into :lvar separated by ' '
   from dictionary.columns
   where libname="&libname" and memname="&dsn"
    ;

  quit ;

  %put Length statment for &libname..&dsn is: &lvar ;

%mend getlen ;

/* SAS LOG

Length statment for IN.VARIABLES is: Protocol $ 20 Studynum $ 5 Dataset $ 32
Name612 $ 8 Name8 $ 32 Lbl612 $ 40 Crf $ 3 Derived $ 3
Definition $ 750 Crfpg $ 150 CRFPageNum $ 30 Lgth $ 8 Fmt $ 25 Submitdb $ 3

END of SAS LOG
*/

Data new;
```

```
   length
      /* cut/paste LENGTH statement here
         then edit LENGTH statement
      */
    set old ;

     other SAS statements
Run;
```

Example 5 also demonstrates the use of SAS dictionary tables and a new Version 9 function – CATX. These two tools can be used in combination with PROC SQL and the macro language to develop more efficient and dynamic applications. Dictionary tables provide metadata (or data about data) about various aspects of the SAS system and are described in detail in DiIorio and Abolafia (2004). In Example 5 the dictionary table COLUMNS in used. COLUMNS provides information about variables in currently allocated datasets and views. In this example COLUMNS is utilized to obtain the list of variables in a user specified dataset and their associated types and lengths. For a more complete discussion of dictionary tables and examples showing how dictionary tables can be used most efficiently with PROC SQL and the macro language see DiIorio and Abolafia (2004).

CATX along with related functions CAT, CATS, and CATT are new functions introduced in Version 9 of the SAS System. These functions facilitate concatenating variables, constants, and expressions by reducing the need to use the TRIM, LEFT, and PUT functions.

CONCLUSION

PROC SQL used in combination with the macro language provides the programmer with a powerful set of tools. In many cases these tools provide a more efficient approach than alternative coding methods. Finally, these tools are an integral part of a programmer's bag of tricks for developing robust and dynamic applications.

REFERENCES

Frank DiIorio and Jeffrey Abolafia (2004), "Dictionary Tables and Views: Essential Tools for Serious Applications," Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference, Paper 237-29.

SAS Institute Inc. (1997), SAS Macro Language Reference, First Edition, Cary, NC: SAS Institute Inc.

CONTACT

The author may be contacted at:

Jeffrey Abolafia
RHO, inc.
6330 Quadrangle Drive, Suite 500
Chapel Hill, NC 27517
Phone: (919)-408-8000 Ext. 331
Fax: (919)-408-0999
Email: jabolafi@rhoworld.com