Paper 101

Dynamically Creating a Where Statement

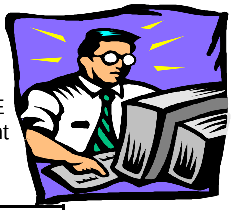
Curtis A. Smith, *Defense Contract Audit Agency, La Mirada, Ca.*

ABSTRACT

Frequently, within a user application, the developer uses a WHERE statement to subset a data set. The developer can pass constant values to the WHERE statement from the user utilizing a variety of methods. For example, the developer can use a macro variable in the WHERE statement whose value is passed when the user makes a selection from a SAS/AF® FRAME. However, what if the developer wants to allow the user to make any number of selections? For example, if the developer uses a WHERE statement to subset based upon the *account* variable, what if one time the user wants to select one account, but the next time the user wants to select ten accounts? How does the developer program a WHERE statement that can accommodate a varying number of constants in the selection criteria? Well, thanks to some fancy help from the SAS Institute's technical staff, the author will present a handy macro loop to dynamically build a WHERE statement.

INTRODUCTION

SAS System® application developers often use a WHERE statement to subset an input file within a DATA step or PROCEDURE. Passing the user's specified value or values to the WHERE statement can be easy if the developer uses a fixed number of expressions in the WHERE statement. For example, if the developer allows the user to select only one value, then passing a value to the WHERE statement is a simple matter. However, what if the developer wants the users to select any number of values? I know my users do not like having to run an application repeatedly as the only way to make all the needed selections. Of course, we have a way around this problem. We are, after all, using the SAS System to develop our applications. (I have found I can do anything I need with the SAS System - I only need to define the problem and then ask the right people how to solve it.) After defining my problem, the SAS Institute's technical support people wrote macro code that solved the problem for me. Following, I will discuss the macro code and how it works. However, this paper relies heavily upon the SAS System macro language and I assume the reader is familiar with at least some macro language basics. If the reader is not familiar with the macro language, the author recommends Art Carpenter's book, "Carpenter's Complete Guide to the SAS(R) Macro Language."



THE PROBLEM

A simple example using a WHERE statement to subset a SAS data set might look like this:

```
data work.fileout;
   set work.filein;
   where account like "61%";
run;
```

If we do not want the value in the where expression hard coded, we can replace it with a macro variable. The value resolved by the macro variable can be assigned many ways, one being from a SAS/AF FRAME and an SCL® statement. Using a macro variable in place of hard coding will change our example to look like this (where *mvar* is the name of our macro variable):

```
data work.fileout;
   set work.filein;
   where account like "&mvar.";
run;
```

If we want our users to select two values to pass to the WHERE statement, we can modify the WHERE statement to look something like this:

```
data work.fileout;
   set work.filein;
   where  account like "&mvar1." or
          account like "&mvar2.";
run;
```

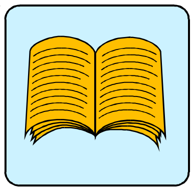
This construction works as long as we limit our users to a fixed number of values that must be passed each time they run the application. However, if we do not what to limit our users (mine do not like to be limited) we quickly see that because our users will select differing numbers of values each time, we cannot construct a WHERE statement with hard coding.

THE SOLUTION

One solution could be to use a SAS/AF FRAME to store the user selections in a SAS data set. We could then merge that SAS data set with the input data set to create the desired subset. Within the DATA step merge, we would select only the observations that match the observations from the SAS Data Set we created from the user's selections. This method would work when we want to subset within a DATA step, but not within a SAS PROCEDURE. Also, this method may not always be the most efficient method when using a DATA step.

There is however, a simple solution using a macro to dynamically create a WHERE statement. (My thanks to the SAS Institute's technical support wizards for devising the macro code.) First, I will present the code (I know that is what you have been waiting for), then I will describe how it works. I will show two code examples: the first with a WHERE statement using the IN operator and the second using the LIKE operator. From the two variations we will see how the form can be modified to meet other possible requirements. I have included line numbers in the code for easy reference in the explanations.

Before I present the code (patience), let us create a typical scenario to use as an example. We have created an application that will generate some reports after allowing the user to specify desired accounts (account is a four numeral character variable) to subset an input SAS data set that resides in the user's data warehouse. The user will specify the accounts using a SAS/AF FRAME that utilizes a Data Table object to create a SAS Data Set. We will then pass the specified accounts stored in the SAS Data Set to a WHERE statement in a DATA step and to a SORT PROCEDURE.



THE CODE

Now the code. We can use the following SAS System code, placed somewhere in the beginning of our SAS program, to subset the SAS Data Set.

```
001  data _null_;
002      set work.userspec;
003      call symput('mvar'||left(_n_),trim(account);
004      call symput('max',left(_n_));
005  run;
006
007  %macro genin;
008      %do i=1 %to &max;
009          "&&mvar&i"
010          %if &i ne &max %then %do;
011              ,
012          %end;
013      %put      user input = "&&mvar&i";
014      %end;
015  %mend genin;
016
017  data work.fileout;
018      set work.filein;
019      where account in (%genin);
```

Now for an explanation of the code.

We use the _null_ DATA step beginning on line 001 to read into macro variables the data set variables from the SAS data set holding the user's specified criteria. In this example, the SAS data set called 'work.userspec' has only one variable, called *account*. The SAS data set can contain any number of observations. For this example, let us assume our user has specified three accounts, so the data set contains three observations. The CALL SYMPUT statement on line 003 reads the value of *account* from each observation and stores each value into a separate macro variable. Each macro variable will have a unique name, using the literal *mvar* plus a numeral equal to the iteration number (_n_) associated with each observation read. The iteration number is concatenated to the literal *mvar* using the concatenation function (||). To ensure there are no spaces within the macro variable name or value, we read the iteration number from the left using the "left" function, and we trim the data set variable value using the "trim" function. The three observations in our example are '6101', '6521', and '6999', so we will have three macro variables called *mvar1*, *mvar2*, and *mvar3* holding these three values.

We use the CALL SYMPUT statement on line 004 to read the iteration number associated with each observation in the DATA step and places it into a single macro variable named *max*. After the last observation is read, the macro variable *max* will hold the number of the last iteration. Because there will be one *mvar* macro variable for each DATA step iteration, the last value placed in *max* will be the maximum number of *mvar* macro variables created.

We now have macro variables as follows: *mvar1*=6101; *mvar2*=6521; *mvar3*=6999; *max*=3. For our example, we want to build a WHERE statement, when all macro variables are resolved, that looks like the following:

```
where account in ('6101','6521','6999');
```

Or, when the macro variables are unresolved, looks like the following:

```
where account in ("&mvar1","&mvar2","&mvar3");
```

We need a macro that will put quoted literal values, separated by a comma, within our hard coded, parenthetical WHERE statement. We will need a macro loop that will dynamically create part of a WHERE statement using the values the user specified, which are the values stored in our *mvar* macro variables. The macro loop in lines 007-015 will create a WHERE statement using the IN operator. We declare and close our macro with the %macro and %mend statements followed by a name of our choice (lines 007 and 015). Here, we use the name 'genin' (if you must know, meaning generate WHERE IN).

The macro loop consists of three things: a %do loop to resolve each *mvar* macro variable; a %if statement to evaluate each iteration of the %do loop; and a %put statement. The %do statement (line 008) instructs the %do loop to continue execution if the iteration (i) is between 1 and the value stored in our macro variable *max*, which in our example is 3. When the number of iterations exceeds the value of *max*, the %do loop will stop executing.

Next, the %do loop will write the value stored in our *mvar* macro variables (line 009). It does this by using the number of the %do loop iteration (i) concatenated to the literal macro variable name *mvar*, resolving to *mvar1*, *mvar2*, etc. In our example, the first iteration of the %do loop will cause &&*mvar*&i to resolve to &*mvar1*, which in turn, will resolve to '6101'. Because in our example, the *account* variable is a character variable, we want our WHERE statement values to be enclosed in quotation marks. So, we enclose &&*mvar*&i in double quotation marks (remembering that macro variables will not resolve when placed in single quotation marks). Notice this line of code is not closed with a semicolon (;).

Because each value in our WHERE statement must be separated by a comma, but without a trailing comma, we need to place a comma in our WHERE statement for each iteration except the last. We accomplish this with the %if statement on line 010. If the number of the current %do loop iteration is not equal to the value of our *max* variable (meaning the iteration is not the last iteration) then the %if statement will execute the resulting %do loop. That %do loop will place a comma after the just written literal value from our *mvar* macro variable (line 011). Notice this line of code is not closed with a semicolon (;). We use the %end statement on line 012 to close our second %do loop. If the first %do loop iteration is equal to the value of our *max* variable, then the %if statement will not be true, the second %do loop will not execute, and the comma will not be placed after the value of our last *mvar* macro variable.

Line 013 contains a %put statement used to place the value stored in the *mvar* macro variables in the SAS log. (This was my big contribution to the code I received from the SAS Institute.) I use this statement just to have a record in the SAS log of the user's selections (in my line of work, we call this the 'audit trail'). The %end statement on line 014 closes the first %do loop.

To use this macro, we can refer to it in a DATA step or a SAS PROCEDURE. In our example (lines 017-020), we create a new SAS data set 'work.fileout' by reading 'work.filein' and subsetting with a WHERE statement. We use the WHERE statement on line 019 that refers to our macro loop %genin. When that macro is executed, the values specified by the user will be placed within the parentheses, with each separated by a comma. That's it. It is that simple.

We may not always want to use a WHERE statement with the IN operator. Often, my users want to use wildcards in their selections. So, I like to use the WHERE statement with the LIKE operator. Let us assume this time our user enters the following values for the *account* variable: '61%', '658_', and 69%'. We want to build a WHERE statement that looks like the following:

```
where   account like '61%' or
        account like '658_' or
        account like '69%';
```

To build this dynamic WHERE statement, we need only make slight modifications to our previous macro loop. Our _null_ DATA step will remain the same. In this example, we name our macro loop ‘genlike’ (why, you guessed it). This WHERE statement is different from our first WHERE statement using the IN operator. Everything except the literal ‘where’ must be dynamically created by the macro loop. However, only two lines of the macro loop need to be modified: the lines that write the literal words into the WHERE statement. Our new macro loop code follows.

```
021  %macro genlike;
022      %do i=1 %to &max;
023          account like "&&mvar&i"
024          %if &i ne &max %then %do;
025              or
026          %end;
027      %put      user input = "&&mvar&i";
028      %end;
029  %mend genlike;
030
031  proc sort data=work.filein;
032      where %genlike and pool = ‘92';
033      by account;
034  run;
```

This time, our line to dynamically write the WHERE statement (line 023) writes the words ‘account like’ and the quoted value from our *mvar* macro variables. As long as the %do loop iteration is not equal to the maximum number of user inputs (the *max* macro variable) the %if statement will cause the literal word ‘or’ (line 025) to be placed after the ‘account like’ plus quoted value. In our example, we then use this macro in a PROC SORT (lines 031-034). We simply use a WHERE statement composed of ‘where’ followed by the reference to our %genlike macro. For variation, in this example, we also have another condition in our WHERE statement, which happens to be a hard coded criterion.

CONCLUSION

Using relatively simple macro code, we can dynamically create WHERE statements that will fit any number of user inputs. We can also use any form of the WHERE statement. We have seen two forms of the WHERE statement in our examples: one using the IN operator, another using the LIKE operator. Likewise, we could use any operator and connectors with only minor modifications to the macro code. This technique has added much flexibility to my applications while simplifying the SAS code.

REFERENCES

SAS MACRO Variables:
SAS Guide to Macro Processing, Version 6, Second Edition, Chapter 2
SAS Language Reference, Version 6, First Edition, Chapter 20
Carpenter's Complete Guide to the SAS® Macro Language

WHERE Statement:
SAS Language Reference, Version 6, First Edition, Chapter 9

DATA Step:
SAS Language Reference, Version 6, First Edition, Chapter 2

Questions: e-mail the author at casmith@mindspring.con.