CREATING MACRO VARIABLES VIA PROC SQL

Mike S. Zdeb, New York State Department of Health

INTRODUCTION

There are a number of ways to create macro variables. You can use %LET to create a macro variable containing a character string. You can use CALL SYMPUT if you want to store the value of a given variable rather than a character string. A third way is to use a SELECT statement together with an INTO option within PROC SQL. Both %LET and CALL SYMPUT are quite limited as to how creative you can get in using them to accomplish various tasks. However, since SELECT INTO offers you the power of PROC SQL, your creativity is bounded not by the limitations on the method of creating a macro variable, but more by how much you know about SAS and PROC SQL. Several examples are used to show you how SELECT INTO can be used to accomplish some tasks via the 'automated' creation of macro variables.

SELECTING RECORDS

Given that you had a list of social security numbers (SSNs), how could you select records from a large, raw data file that had a social security matching a number in your list. There are a number of different ways to do this. You could do a data set merge, but this would involve creating SAS data sets from both files, sorting both, then merging. PROC SQL avoids the sort but still requires two SAS data sets. If you create a SAS data set from the list of SSNs, you could add an INDEX and filter records from the large file as they are read by searching the list via the index. A format could be created via a CNTLIN data set and used as a 'look table' as you read the large file. Yes, there are a lot of ways to do this task. I'd like to add another - a macro variable created via SELECT INTO.

EXAMPLE 1...use a macro variable to select records from a raw data file

```
data ssn_list;
input ssn  @@;
datalines;
123456789 234567890 345678901 456789012
;
run;

proc sql noprint;
select ssn into :ssnok
separated by ' '
from ssn_list;
quit;

*** not necessary - display the value of
the macro variable in the LOG;
%put &ssnok;

data selected;
input @1 ssn  @;
*** use the macro variable (a list of SSNs)
to filter records;
if ssn not in (&ssnok) then return;
input <list of variables>;
output;
run;
```

The syntax of the SELECT statement for macro variable creation is...

```
SELECT <variable name in a SAS data set>
INTO :<macro variable name>
SEPARATED BY '<a blank, a character, or a
character string>'
```

In this example, we want to store the values of the variable SSN from data set SSN_LIST as the text of the macro variable &SSNOK. The text stored in the macro variable is displayed in the LOG via %PUT...

```
106  proc sql noprint;
107  select ssn into :ssnok
108  separated by ' '
109  from ssn_list;
110  quit;
NOTE: The PROCEDURE SQL used 0.0 seconds.
111  *** not necessary - display the value
     of the macro variable in the LOG;
112  %put &ssnok;
1.2346E8 2.3457E8 3.4568E8 4.5679E8
```

The chance of finding anyone who uses their SSN in scientific notation is pretty slim. What happened? If the values of a numeric variable are stored in a macro variable via SELECT INTO, any number with more than eight digits will be stored in scientific notation. You can get around that rule by adding a format statement to the data step in which the data set SSN_LIST was created...

```
format ssn 9.;
```

or by adding a format to the SELECT statement in PROC SQL...

```
select ssn format=9. into :ssnok ...
```

If you add the format in either way, the LOG will show...

```
156  %put &ssnok;
123456789 234567890 345678901 456789012
```

Once the macro variable has been created, it can be used to filter records from the large, raw data file by using it with the IN operator. The IN operator is an alternative to writing...

```
if ssn ne 123456789 and
   ssn ne 234567890 and
   ssn ne 345678901 and
   ssn ne 456789012 then return;
```

Though the IN operator is a convenient way to save keystrokes, it is a less efficient way of filtering data than the longer statement just shown. So, since SAS is writing the code, why not have it write the more efficient code.

```
proc sql noprint;
select ssn format=9. into :ssnok
separated by ' and ssn ne '
from ssn_list;
quit;
```

A portion of the LOG...

```
201  %put &ssnok;
123456789 and ssn ne 234567890 and ssn ne
345678901 and ssn ne 456789012
```

The SAS code in the data step that reads the large, raw data file must be changed...

```
data selected;
input @1 ssn  @;
if ssn not eq &ssnok then return;
input <list of variables>;
output;
run;
```

If you prefer to use SSN as a character variable (or you are reading and selecting records from a SAS data set in which SSN is stored as a character variable), you know that the values of SSN in either the IN operator example or the 'long hand' example must be surrounded by quotes. SELECT INTO can add the quotes for you.

```
select quote(ssn)
into :ssnok separated by ' and ssn ne '
```

If SSN is a character variable and you use the QUOTE option in the SELECT statement, the LOG will show...

```
215  %put &ssnok;
"123456789" and ssn ne "234567890" and ssn
ne "345678901" and ssn ne "456789012"
```

ELIMINATING OBSERVATIONS FROM ANALYSIS

If you have a SAS data set comprising observations from a number of different facilities, how can you use PROC UNIVARIATE to analyze your data, eliminating those facilities that contribute fewer than 100 observations to the data set? One way to do this would be to run PROC FREQ and count the number of observations in each facility. If the number of 'low count' facilities is small, you could write a WHERE statement within PROC UNIVARIATE to eliminate those facilities from the analysis. However, if the number of such facilities is large, or if you want to automate the process, you can one again use SELECT INTO and create a macro variable that contains values representing the 'low count' facilities. Assume that FACILITY is a character variable and your data set is named MYDATA.

EXAMPLE 2...use a macro variable to eliminate observations from analysis (PROC FREQ+PROC SQL)

```
proc freq data=mydata;
table facility /
noprint out=low (where=(count lt 100));
run;

proc sql noprint;
select quote(facility) into :lowcount
separated by ' and facilty ne '
from low;
quit;

proc univariate data=mydata;
where facility ne &lowcount;
var <variables>;
<more univariate option>
run;
```

PROC FREQ is used to create a SAS data set that contains one observation for each facility with a frequency of less than 100. That data set (LOW) is used in PROC SQL to create a macro variable (&LOWCOUNT) that is then used in a WHERE statement within PROC UNIVARIATE to eliminate all observations from 'low count' facilities.

Since we are using PROC SQL to create the macro variable, why not use also use it to do the identification of 'low count' facilities.

EXAMPLE 3...use a macro variable to eliminate observations from analysis (PROC SQL only)

```
proc sql noprint;
select quote(facility) into :lowcount
separated by ' and facilty ne '
from mydata
group by facility
having count(*) lt 100;
quit;

proc univariate data=mydata;
where facility ne &lowcount;
var <variables>;
<more univarite option>
run;
```

Two PROC SQL statements are added to eliminate the need for PROC FREQ...

```
group by facility
having count(*) lt 100
```

The observations are put into groups based on values of the variable facility. You can count the observations within each group with COUNT(*).

CORRESPONDENCE

Any questions regarding this paper can be sent to the author at...

msz03@health.state.ny.us