SESUG Paper 267-2018

PROC SQL - GET "INTO:" IT!

Kelly Schlessman, The Emmes Corporation

ABSTRACT

The "INTO:" clause within PROC SQL is an extremely useful tool, but may be a mystery to those unfamiliar with SQL. A SELECT statement paired with the INTO clause in PROC SQL provides a simple method of storing data in a macro variable - or many macro variables! What can go into these macro variables? Values of selected variables from a dataset, summary statistics, and delimited lists of values, to name a few. A task that may require multiple steps in traditional DATA STEP programming can be achieved in a single line of PROC SQL code. All that is required to harness the power of the INTO clause is basic knowledge of SELECT statements in PROC SQL. This paper will provide an introduction to the INTO clause for novice PROC SQL users, and demonstrate some useful ways in which it can be put to work in your code.

INTRODUCTION

PROC SQL can help accomplish a wide variety of programming tasks, often in a more efficient manner than the traditional DATA step or other SAS® procedures. Within a single procedure you can identify columns to keep or modify, sort data, calculate summary statistics, and more. This paper assumes basic knowledge of PROC SQL syntax, including the SELECT statement:

```
PROC SQL options;
  SELECT columns(s)
    FROM table-name|view-name
    WHERE expression
    GROUP BY column(s)
    HAVING expression
    ORDER BY column(s);
QUIT;
```

One great example of the power of PROC SQL is its ability to create macro variables from the results of a PROC SQL SELECT statement. Macro variables can then be called elsewhere in the program. This process can create more efficient and dynamic programs by combining multiple steps into a single procedure, and by eliminating the need to hard-code values or modify them each time a program is run.

The basic syntax of the INTO keyword:

```
PROC SQL;
  SELECT macro-specification
    INTO :macro-variable-name
    FROM source-data
QUIT;
```

The following code produces a dataset called *enrollment*, which will be used in all subsequent examples:

```
DATA enrollment;
INPUT obs id $ enrolldt MMDDYY10. sex $ age state $ drug $;
DATALINES;
1 S01 10/15/2017 F 52 MD D2
2 S05 12/01/2017 M 70 MD D2
3 S04 01/30/2018 F 39 MD D1
4 S06 01/30/2018 M 55 MD D3
5 S02 02/02/2018 F 48 VA D2
6 S03 02/03/2018 M 41 MD D3
7 S07 02/28/2018 F 61 PA D1
;
```

CREATING A SINGLE MACRO VARIABLE

In its simplest form, INTO can store a single value in a macro variable. This example selects a single value of *age* from the enrollment dataset and stores it in a macro variable called *subjage*:

```
proc sql;
  select age
    into :subjage
    from enrollment;
quit;
%put Subject age is &subjage.;
```

View the result of the %PUT statement in the log:

```
Subject age is       52
```

Note that 52 is the age from the first row of the table. If a particular row is not specified, the value from the first row will be selected by default. A WHERE statement can be included to select a different row.

Also you can see when printed to the log that there are leading blanks before the number. When storing a value in a single macro variable, PROC SQL preserves leading or trailing blanks by default. To remove leading and trailing blanks, simply use the TRIMMED option:

```
proc sql;
  select age
    into :s05age TRIMMED
    from enrollment
    where id='S05';
quit;
%put Subject S05 age = &s05age.;
```

```
Subject S05 age = 70
```

CREATING A MACRO VARIABLE CONTAINING SUMMARY STATISTICS

Use the power of the SELECT statement to create macro variables from more than just individual data values. Counts and other summary statistics can also be calculated in a SELECT statement and stored into macro variables. The simple example below shows how a total count can be saved into a macro variable *called totsubj*:

```
proc sql;
    select count(id)
    into :totsubj trimmed
    from enrollment;
quit;
```

```
%put Total Number of Subjects = &totsubj.;
```

```
Total Number of Subjects = 7
```

Another example stores the first chronological date from the enrollment dataset in *firstdt*. Note that a date format is defined in the SELECT statement, otherwise the SAS numeric date value will be displayed when *firstdt* is called:

```
proc sql;
  select min(enrolldt) format=date9.
    into :firstdt
    from enrollment;
quit;
```

```
%put &firstdt.;
```

```
15OCT2017
```

Figure 1 demonstrates how the *totsubj* and *firstdt* macro variables above can be inserted into titles and footnotes.

```
title2 "&totsubj. Total Subjects Enrolled";
footnote j=left h=12pt "*The first subject was enrolled on &firstdt.";
```

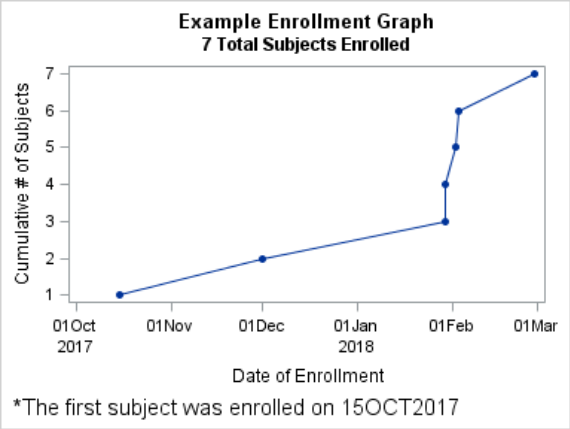

Figure 1. Macro Variables in Titles and Footnotes

CONCATENATING MULTIPLE VALUES WITH "SEPARATED BY"

The SEPARATED BY option allows you to concatenate a series of values into one macro variable, with the delimiter specified in quotation marks.

```
proc sql;
  select distinct state
    into :statelist SEPARATED BY ", "
    from enrollment;
quit;
%put &statelist.;
```

The above code produces a comma-delimited list of state codes:

```
MD, PA, VA
```

Note the use of the DISTINCT keyword in the SELECT statement. This instructs PROC SQL to select only unique values of the state variable. Omitting the DISTINCT keyword would result in repeated values:

```
MD, MD, MD, MD, VA, MD, PA
```

SEPARATED BY must be accompanied by a character string in quotation marks. If you do not wish to use a character to separate values, a blank space in quotation marks can be specified (" "). If all values should be concatenated together with no space in between, an empty set of quotation marks can be specified (""). Leading and trailing blanks are automatically removed when using SEPARATED BY. If you wish to preserve leading and trailing blanks, use the NOTRIM option.

COMPARISON WITH CALL SYMPUT

The CALL SYMPUT routine, used within a DATA step, is another method for creating macro variables. It can be used to produce many of the same results as PROC SQL INTO, but often requires several other steps to be completed before the macro variable can be created.

In the previous example we created a concatenated list of unique state codes. Doing this via CALL SYMPUT requires several steps:

1. Remove duplicate values
2. Transpose data so that all values are within a single observation

3. Use the CATX function to concatenate all values into a string, then put it in a macro variable using CALL SYMPUT

Here is how you can create the concatenated state list using a DATA step:

```
proc sort data=enrollment out=data1 NODUPKEY;
    by state;
run;
proc transpose data=data1 out=data2 prefix=state;
    var state;
run;
data _null_;
    set data2;
    call symput("statelistalt",catx(", ",of state:));
run;
%put &statelistalt.;
MD, PA, VA
```

The CATX function (available in SAS version 9 and later) is very useful here. It allows you to concatenate several values separated by a comma, and does not require you to know the total number of unique states. However, PROC SQL is still a simpler solution in this particular example, as it handles the elimination of duplicate results and reading values from a single column within a single SELECT statement.

Depending on the situation, either PROC SQL INTO or CALL SYMPUT may be the more efficient choice. It will certainly enhance your programming to know both options!

CREATING MULTIPLE MACRO VARIABLES

Multiple macro variables can be created in one step from the values of a single column. A range of macro variable names is specified after INTO using a dash (-) or the keyword THROUGH or THRU. The following example creates three macro variables containing the values of the *drug* column:

```
proc sql;
    select distinct(drug)
    into :drug1 - :drug3
    from enrollment;
quit;
%put &drug1. &drug2. &drug3.;
```

```
D1 D2 D3
```

When using this technique, you are required to know the number of macro variables that are being created (in the above example, this is the number of different drug codes). Depending on the nature of your data and the manner in which INTO is being used, the number of macro variables may not always be known. One solution to this problem is to automate the process by counting the number and storing it into a separate macro variable. This can be done within the same PROC SQL statement:

```
proc sql;
  select count(distinct drug)
    into :ncount trimmed
    from enrollment;

  select distinct(drug)
    into :drug1 - :drug&ncount.
    from enrollment;
quit;
```

CREATING MULTIPLE MACRO VARIABLES WITH "GROUP BY"

Another common way to create multiple macro variables from a single column is to use a GROUP BY clause:

```
proc sql;
  select count(id)
    into :ndrug1 - :ndrug3
    from enrollment
    group by drug;
quit;
%put &ndrug1. &ndrug2. &ndrug3.;
```

```
2 3 2
```

Note that the macro variables are assigned in the order of the GROUP BY variable. Even though the first observation in the raw data has drug=D2, the first macro variable created (ndrug1) contains a count of the number of observations with drug=D1.

CREATING MACRO VARIABLES FROM MULTIPLE COLUMNS

Macro variables can be created from multiple columns in a single SELECT statement, by listing the source columns separated by a comma, and then listing the corresponding INTO :macro-variable statements separated by a comma. The macro variables are assigned in the order of the columns in the SELECT statement.

In the second select statement of the following example, the first set of macro variables created (nd1-nd3) contains the total counts in each group, and the second set of macro variables created (md1-md3) contains the mean age of each group:

```
proc sql;
  select count(distinct drug)
      into :ndrug trimmed
      from enrollment;

  select count(id), mean(age)
      into :nd1 - :nd&ndrug., :md1 - :md&ndrug.
      from enrollment
      group by drug;
quit;
%put &nd3. &md3.;
3 56.66667
```

CONCLUSION

The INTO keyword can create macro variables in a variety of useful ways. Once the basic syntax is learned, the possibilities are endless. You can take advantage of SELECT INTO even if you aren't comfortable with other aspects of PROC SQL programming, since macro variables can be called later anywhere in the SAS code, from DATA steps to ODS output. Learning PROC SQL SELECT INTO can help make your code more efficient and dynamic.

CONTACT INFORMATION

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

Kelly Schlessman
Emmes Corporation
kschlessman@emmes.com