Paper 197-25

Using the Magical Keyword "INTO:" in PROC SQL

Thiru Satchi
Blue Cross and Blue Shield of Massachusetts, Boston, Massachusetts

Abstract

"INTO:" host-variable in PROC SQL is a powerful tool. It simplifies programming code while minimizing the risk of typographical errors. SQL INTO: creates one or more macro variables, based on the results of a SELECT statement. This macro variable(s) reflects a list of values that can then be used to manipulate data in both DATA and PROC steps. The usefulness of SQL INTO: will be demonstrated by analyzing a large medical claims database.

Keywords: INTO:, host-variable, macro, SQL

Introduction

The "INTO:" host-variable in PROC SQL is a valuable resource for creating a macro variable made up of values. It overcomes several limitations in hard coding values, including the possibility of typographical errors, resource constraints, and does not account for dynamic data. Previous presentations have explored the application and utility of this host-variable (1-2).

The purpose of this presentation is to review previously covered, as well as to introduce new forms and applications of the "INTO:" host-variable to address common business needs.

Variations of the "INTO:" Host-Variable

Prior to the Release 6.11, the "INTO:" host-variable simply stored the first row of values (3). For example, the host-variable in Listing 1 that refers to the sample data in Listing 2 would store the following: P01 53071.

Listing 1. Release 6.06 form of the "INTO:" host-variable.

```
1.  PROC SQL NOPRINT;
2.    SELECT EMPID, DIAG
3.      INTO :EMP_LIST, : DIAGLIST
4.        FROM MASTER;
5.  QUIT;
6.
7.  %PUT &EMP_LIST &DIAGLIST;
```

Listing 2. Sample data.

```
1.  DATA MASTER;
2.    INPUT EMPID $3. DIAG $5. MEMID 9.;
3.    CARDS;
4.  P01    53071   258766
5.  P02    99215   92139678
6.  P03    99201   921396
7.  P04    45355   566511
8.  P05    45383   464467896
9.  P06    43260   87932
10. P07    99213   73771
11. P08    45380   846420987
12. P09    88714   346987
13. P10    55431   3469871
14. ;
```

However with this release, multiple rows of values can now be stored. In Listing 3a, each row of values is stored in separate macro variables (Listing 3b). In addition, a dash (-) or the keywords 'THROUGH' or 'THRU' can be used to denote a range of macro variables. And the keyword 'DISTINCT' is used to generate a unique list of values.

Listing 3a.
Basic Form of the "INTO:" Host-Variable (Release 6.11).

```
1.  PROC SQL NOPRINT;
2.    SELECT DISTINCT EMPID, DIAG
3.      INTO :E1 - :E4, :D1 - :D3
4.        FROM MASTER;
5.  QUIT;
6.
7.  %PUT &E1 &D1;
8.  %PUT &E2 &D2;
9.  %PUT &E3 &D3;
```

Listing 3b. Values Generated in Listing 3a.

```
%PUT &E1 &D1:
      P01     53071

%PUT &E2 &D2:
      P02     99215

%PUT &E3 &D3:
      P03     99201
```

The "INTO:" host-variable can also be used to generate lists of values, the value of which has been previously demonstrated (2). These lists can be modified with modifiers (Listing 4a). For example,

the 'SEPERATED BY' qualifier indicates how this list of values should be concatenated; in Listing 4a, macro variable 'E1', is separated by a comma (results are presented in Listing 4b). Another modifier is 'QUOTE', which flanks each value with double quotes (")(Listing 4a, macro variable 'E2'; results are presented in Listing 4b). It should be noted that leading and trailing blanks are deleted from the values by default when using the QUOTE modifier, but 'NOTRIM' can be added to retain these blanks. Values can also be manually concatenating the quotes (Listing 4a, macro variable 'E3'; results are presented in Listing 4b). This feature is useful when adapting lists to other systems. For example, the SQL in the DB2 environment accepts single quotes, not double quotes. Therefore, we must manually create a list of values separated by a single quote, because of the QUOTE modifier (see reference 2).

Listing 4a.
Variations of the "INTO:" Host-Variable (Release 6.11).

```
1.   PROC SQL NOPRINT;
2.    SELECT DISTINCT EMPID,
3.                 QUOTE(EMPID),
4.                 " ' " || (EMPID) || " ' ",
5.                 MEMID ,
6.                 MEMID FORMAT 9.
7.
8.       INTO    :E1 SEPERATED BY " , " ,
9.               :E2 SEPERATED BY " , " ,
10.              :E3 SEPERATED BY " , " ,
11.              :M1 SEPERATED BY " " ,
12.              :M2 SEPERATED BY " , "
13.     FROM MASTER;
14.  QUIT;
15.
16.  %PUT &E1; %PUT &E2; %PUT &E3;
17.  %PUT &M1; %PUT &M2;
```

Listing 4b. Lists of Values Generated in Listing 4a.

'E1' List:

P01,P02,P03,P04,P05,P05,P06,P07,P08,P09,P10

'E2' List:

"P01", "P02", "P03", "P04", "P05","P06","P07","P08", "P09", "P10"

'E3' List:

'P01', 'P02', 'P03', 'P04', 'P05','P05','P06','P07','P08', 'P09', 'P10'

'M1' List:

258766 92139678 921396 566511 4.6447E8, 87932 73771 8.4642E8, 346987 3469871

'M2' List:

258766, 92139678, 921396, 566511, 464467896, 87932, 73771, 846420987, 346987, 3469871

It is important to define numeric values in the SELECT statement (Listing 4, macro variable 'M1'). If not, variable length will be a maximum of 8 bytes by default. This demonstrated in Listing 4 (macro variable 'M2') as the 9-digit numbers, 846420987 and 464467896 are converted to 8.4642E8 and 4.6447E8, respectively (Listing 4b). It should be noted that SAS will accept a list of numeric variables separated by either a comma or a blank.

This variation of the INTO: host-variable does not differentiate between actual values and blanks (Listing 5). However, it is customary to encounter variables of varying rows from which I need to generate lists. Listing 6a presents a modified version of the INTO: host-variable to account for the data in Listing 5 (results are presented in Listing 6b). This version accomplishes two things: ignores blank values and creates multiple lists in one step.

It should be noted that the three ampersands (&) in Line 12 of the macro are necessary in generating the appropriate lists. That is, (1) the first two ampersands convert to a single one; (2) the third is used to establish the macro variable 'VNAME'; (3) this macro variable is replaced by the variables to be listed (Lines 16-18), and (4) these variables are then passed to the INTO: host-variable in Line 9.

Listing 5.
Sample Data Containing Variables of Varying Lengths

```
1.   DATA MASTER;
2.    INPUT     @01 DIAG     $5.
3.              @07 PROC     $5.
4.              @14 ACCT     $4.
5.   ;
6.              V270    67692   0600
7.              V271    59400   0700
8.              V272    59409   0710
9.              V273    59410   0750
10.             V274    59510   0800
11.             V275    59514   0850
12.             V276            0900
13.             V277            0950
14.             V300
15.             V3001
16.  ;
```

Listing 6a.
Variations of the "INTO:" Host-Variable (Release 6.11).

```
1.   %MACRO CREATE(VNAME);
2.   %GLOBAL &VNAME.LIST;
3.
4.   DATA TEST;  SET MASTER (KEEP=&VNAME);
5.    WHERE &VNAME = ' ';
6.
```

```
7.   PROC SQL FEEDBACK NOPRINT;
8.     SELECT QUOTE(&VNAME)
9.       INTO: &VNAME.LIST SEPERATED BY ','
10.        FROM TEST;
11.
12.  %PUT &&&VNAME.LIST;
13.
14.  %MEND CREATE;
15.
16.  %CREATE(DIAG);
17.  %CREATE(PROC);
18.  %CREATE(ACCT);
```

Listing 6b. Lists of Values Generated in Listing 6a.

'DIAG' List:

"V270","V271","V272","V273","V274","V275",
"V276","V277","V3000","V3001","V301"

'PROC' List:

"67696","59400","59409","59410","59510","59514",

'ACCT' List:

"0600","0700","0710","0750","0800","0850","0900",
"0950"

Application of the "INTO:" Host-Variable

I have presented an overview of the "INTO:" host-variable. I have previously illustrated the utility in overcoming limitations with the SQL Pass-Through facility (2). I will now demonstrate another application using the host-variable to generate a list of dummy variables. This is program is similar to that of a previous presentation (1), but it more applicable to health care claims data.

Health care claims data contains multiple rows of transactions per patient that varies by the number of services received. It is often necessary to summarize this data which may comprise of millions of rows. For this example, I will focus on summarizing the following variables for the claims data: unique patient ID ('PAT_ID'), treatment group ('TG_GRP'), service date ('SVC_DT'), and paid amount for that service ('PAID-AMT'). Here is an abbreviated sample of medical claims data (taken from Appendix A, Step 1). The treatment group here represents a classification of the treatment the patient receives and range from risk factors (e.g., obesity) to conditions (e.g., coronary artery disease).

```
DATA MASTER;
INPUT    @01 PAT_ID      $2.
         @04 TG_GRP      $4.
         @10 SVC_DT      MMDDYY10.
         @22 PAID_AMT    2.
;
DATALINES;
         P1 TG01  01/21/1999  66
         P1 TG12  02/10/1999  11
         P1 TG03  03/16/1999  46
         P1 TG15  03/16/1999  46
         P1 TG04  05/09/1999  99
                       ⋮
         P1 TG18  12/31/1999  45
         P1 TG12  01/07/1999  32
         P1 TG99  05/18/1999  12
```

I would like to summarize this information to the patient level by summing the total number of medical visits and the corresponding paid amounts and identifying the treatment group(s) that afflicted each patient. Such a summary would look like the following:

ID	Visits	Paid	TG1	TG3	…	TG74	TG88	TG99
P1	14	713	1	1	…	0	1	1

There are several potential predicaments in conducting such an analysis. First, there is no assurance that all possible treatment groups will affect the patient population. And second, the data source could be very large. That is, an analysis of health care claims data would likely comprise of millions of rows of data for tens of thousands of patients. Thus it would be very resource consuming to modify SAS programming code to reflect varying number of treatment groups for a potentially very large population. Thus, a program that accounted for a dynamic data source, yet require minimal maintenance, would be useful for this analysis. Such a program incorporating the INTO: host-variable is presented in Appendix A and will now be discussed in detail.

Step 1

The first step of this program (Appendix A) initially reads in the data (Lines 1-30), which is then summarized by patient (PAT_ID) and treatment group (TG_GRP) using PROC MEANS (Lines 34-39). This summary is outputted to a SAS dataset called 'TG_SUM'. In the process, we establish a variable called 'VISIT', which is the number of service visits, based on the frequency, or the number of times each unique combination of patient and treatment group occurs. A printout of the TG_SUM data is presented in Appendix B ("First Step").

Step 2
The next step utilizes the INTO: host-variable in PROC SQL to generate a unique list of treatment groups that are separated by a space. This list is made of only those treatment groups present in the patient population and is stored as a macro variable, 'TGLIST'.

Step 3
The third step takes the list of all available treatment groups stored in TGLIST and converts them into variables using the array feature. In addition, these newly formed variables are assigned with a '0' using a DO LOOP. This DO LOOP works relies on the DIM function, which tracks the number of newly formed variables. A printout of the resulting dataset is presented in Appendix B ("Third Step"). It should be noted that this list of dummy variables is generated and applied to all patients, regardless if they are affected by a treatment group or not.

Step 4
The next step of this program tags the newly formed variables if the corresponding treatment group is present. This is accomplished with the SAS CEIL function, which scans the TGLIST macro variable and populates the variables with a '1'. Appendix B ("Fourth Step") illustrates this tagging.

Step 5
The fifth and final step of this program performs a patient-level summary of the data using PROC MEANS. This summary is then outputted to a separate dataset, 'PAT_SUM', which is presented in Appendix B ("Fifth Step").

Conclusion

"INTO:" host-variable in PROC SQL is a powerful resource that can overcome numerous coding issues. I have illustrated different variations of this host-variable and an example of its application to overcome a typical issue in summarizing a large medical claims dataset.

Contact Information

Thiru Satchi
150 Rumford Avenue
Mansfield, MA 02048
TSatchi@hotmail.com
(617) 832-3432 (work)

Acknowledgement

I am indebted to Edgar Mounib for his assistance on this presentation.

References

1. Eddlestone M-E. (1997), "Getting More Out of "INTO" in PROC SQL: An Example for Creating Macro Variables," *NESUG '97 Proceedings*, 307-308.
2. Satchi T. (1999), "An SQL List Macro to Retrieve Data from Large SAS/DB2 Databases," *Proceedings of the Twenty-Fourth Annual SAS® Users Group International Conference*, 24, 1297-1302.
3. SAS Institute Inc. (1995), *SAS® Software: Changes and Enhancements, Release 6.11*. Cary, NC: SAS Institute Inc.

Appendix A

Application of the "INTO:" Host-Variable – Sample Code

Step	Line	Code
1	1.	DATA MASTER;
	2.	INPUT @01 PAT_ID $2.
	3.	@04 TG_GRP $4.
	4.	@10 SVC_DT MMDDYY10.
	5.	@22 PAID_AMT 2.
	6.	;
	7.	
	8.	DATALINES;
	9.	P1 TG01 01/21/1999 66
	10.	P1 TG12 02/10/1999 11
	11.	P1 TG03 03/16/1999 46
	12.	P1 TG15 03/16/1999 46
	13.	P1 TG04 05/09/1999 99
	14.	P1 TG04 05/10/1999 92
	15.	P1 TG04 05/15/1999 96
	16.	P1 TG03 01/25/1999 56
	17.	P1 TG12 11/29/1999 35
	18.	P1 TG04 01/12/1999 27
	19.	P1 TG18 12/26/1999 50
	20.	P1 TG18 12/31/1999 45
	21.	P1 TG12 01/07/1999 32
	22.	P1 TG99 05/18/1999 12
	23.	P2 TG12 07/26/1999 61
	24.	P2 TG04 10/23/1999 61
	25.	P2 TG46 11/24/1999 61
	26.	P2 TG74 12/25/1999 61
	27.	P2 TG88 03/08/1999 78
	28.	P2 TG88 02/28/1999 29
	29.	P2 TG11 12/01/1999 58
	30.	P2 TG11 12/31/1999 21
	31.	;
	32.	PROC SORT; BY TG_GRP;
	33.	
	34.	PROC MEANS SUM NOPRINT NWAY DATA=MASTER;
	35.	CLASS PAT_ID TG_GRP;
	36.	VAR PAID_AMT;
	37.	OUTPUT OUT=TG_SUM(RENAME=(_FREQ_=VISIT) DROP=_TYPE_)
	38.	SUM=TOT_AMT;
	39.	RUN;
	40.	
	41.	PROC PRINT DATA=TG_SUM;
	42.	TITLE "TREATMENT GROUP SUMMARY BY PATIENT";
	43.	RUN;
2	1.	PROC SQL NOPRINT;
	2.	SELECT DISTINCT TG_GRP
	3.	INTO: TGLIST SEPERATED BY ' '
	4.	FROM MASTER;
	5.	%PUT &TGLIST;

Appendix A *(continued)*

Step	Line	Code
3	1.	DATA NEWDATA (DROP=I);
	2.	SET TG_SUM;
	3.	
	4.	ARRAY TEMP{*} &TGLIST;
	5.	
	6.	DO I=1 TO DIM(TEMP);
	7.	TEMP{I}=0;
	8.	END;
	9.	
	10.	PROC PRINT DATA=NEWDATA;
	11.	TITLE "ESTABLISH TREATMENT GROUPS WITH VALUES SET TO ZERO”;
	12.	RUN;
4	1.	DATA FIXDATA; SET NEWDATA;
	2.	
	3.	ARRAY TEMP{*} &TGLIST;
	4.	
	5.	TEMP(CEIL(INDEX("&TGLIST",TRIM(TG_GRP))/(LENGTH(TG_GRP)+1))) = 1;
	6.	
	7.	PROC PRINT DATA=FIXDATA;
	8.	TITLE "TREATMENT GROUPS ASSIGNED 1 IF CONDITION IS PRESENT”;
	9.	RUN;
5	1.	PROC MEANS SUM NOPRINT NWAY DATA=FIXDATA;
	2.	CLASS PAT_ID ;
	3.	VAR VISIT TOT_AMT &TGLIST;
	4.	OUTPUT OUT=PAT_SUM(DROP= _FREQ_ _TYPE_) SUM=;
	5.	RUN;
	6.	
	7.	PROC PRINT DATA=PAT_SUM;
	8.	TITLE "PATIENT SUMMARY REPORT";
	9.	RUN;

Appendix B

Output Generated by the Application of the "INTO:" Host-Variable – Sample Code

First Step

TREATMENT GROUP SUMMARY BY PATIENT

PAT_ID	TG_GRP	VISIT	TOT_AMT
P1	TG01	1	66
P1	TG03	2	102
P1	TG04	4	314
P1	TG12	3	78
P1	TG15	1	46
P1	TG18	2	95
P1	TG99	1	12
P2	TG04	1	61
P2	TG11	2	79
P2	TG12	1	61
P2	TG46	1	61
P2	TG74	1	61
P2	TG88	2	107

Third Step

ESTABLISH TREATMENT GROUPS WITH VALUES SET TO ZERO

PAT_ID	TG_GRP	VISIT	TOT_AMT	TG01	TG03	TG04	TG11	TG12	TG15	TG18	TG46	TG74	TG88	TG99
P1	TG01	1	66	0	0	0	0	0	0	0	0	0	0	0
P1	TG03	2	102	0	0	0	0	0	0	0	0	0	0	0
P1	TG04	4	314	0	0	0	0	0	0	0	0	0	0	0
P1	TG12	3	78	0	0	0	0	0	0	0	0	0	0	0
P1	TG15	1	46	0	0	0	0	0	0	0	0	0	0	0
P1	TG18	2	95	0	0	0	0	0	0	0	0	0	0	0
P1	TG99	1	12	0	0	0	0	0	0	0	0	0	0	0
P2	TG04	1	61	0	0	0	0	0	0	0	0	0	0	0
P2	TG11	2	79	0	0	0	0	0	0	0	0	0	0	0
P2	TG12	1	61	0	0	0	0	0	0	0	0	0	0	0
P2	TG46	1	61	0	0	0	0	0	0	0	0	0	0	0
P2	TG74	1	61	0	0	0	0	0	0	0	0	0	0	0
P2	TG88	2	107	0	0	0	0	0	0	0	0	0	0	0

Appendix B *(continued)*

Fourth Step

TREATMENT GROUPS ASSIGNED 1 IF CONDITION IS PRESENT

PAT_ID	TG_GRP	VISIT	TOT_AMT	TG01	TG03	TG04	TG11	TG12	TG15	TG18	TG46	TG74	TG88	TG99
P1	TG01	1	66	1	0	0	0	0	0	0	0	0	0	0
P1	TG03	2	102	0	1	0	0	0	0	0	0	0	0	0
P1	TG04	4	314	0	0	1	0	0	0	0	0	0	0	0
P1	TG12	3	78	0	0	0	0	1	0	0	0	0	0	0
P1	TG15	1	46	0	0	0	0	0	1	0	0	0	0	0
P1	TG18	2	95	0	0	0	0	0	0	1	0	0	0	0
P1	TG99	1	12	0	0	0	0	0	0	0	0	0	0	1
P2	TG04	1	61	0	0	1	0	0	0	0	0	0	0	0
P2	TG11	2	79	0	0	0	1	0	0	0	0	0	0	0
P2	TG12	1	61	0	0	0	0	1	0	0	0	0	0	0
P2	TG46	1	61	0	0	0	0	0	0	0	1	0	0	0
P2	TG74	1	61	0	0	0	0	0	0	0	0	1	0	0
P2	TG88	2	107	0	0	0	0	0	0	0	0	0	1	0

Fifth Step

PATIENT SUMMARY REPORT

PAT_ID	VISIT	TOT_AMT	TG01	TG03	TG04	TG11	TG12	TG15	TG18	TG46	TG74	TG88	TG99
P1	14	713	1	1	1	0	1	1	1	0	0	1	1
P2	8	430	0	0	1	1	1	0	0	1	1	0	0