More About "INTO:Host-Variable" in PROC SQL: Examples

John Q. Zhang, AT&T at Basking Ridge, NJ

ABSTRACT

The examples presented in this paper show that the INTO: clause of PROC SQL (both before and after enhancement) can considerably facilitate SAS® data manipulation in both DATA and PROC steps.

INTRODUCTION

Even SAS is one of a few languages that require minor typing, the programming time is still a major factor concerned with the efficiency of SAS data processing. Tremendous efforts have been made to free users from as much keystroke as possible. SAS and SAS users are making progress. INTO: clause of PROC SQL is one of the most indicative.

EXAMPLES OF PROC STEPS

INTO: clause can make much easier some of PROC steps.

Example 1:
If PROC FREQ is required on all the variables of a SAS data set. A PROC CONTENTS should be run and output a data set with only NAME and/or TYPE in it; then use SELECT…INTO: of PROC SQL to define those variables as macro variables; then run PROC FREQ on the variables. 2 or more SELECT statements might be used to define the macro variables for concerned PROC FORMAT statements if necessary. Here is the code.

```
PROC CONTENTS DATA=LIB.SASDATA NOPRINT
OUT=OUT(KEEP=NAME TYPE);
RUN;

PROC SQL NOPRINT;
SELECT NAME INTO :ALLVAR SEPARATED BY ' '
FROM OUT;
SELECT NAME INTO :NUMVAR SEPARATED BY ' '
FROM OUT
WHERE TYPE=1;
SELECT NAME INTO :CHRVAR SEPARATED BY ' '
FROM OUT
WHERE TYPE=2;
QUIT;

PROC FREQ DATA=LIB.SASDATA;
TABLE &ALLVAR/LIST MISSING NOCUM;
FORMAT &NUMVAR NUMFMT.
       &CHRVAR $CHRFMT.;
RUN;
QUIT;
```

The same way could be applied to PROC MEANS & PROC SUMMARY.

Example 2:
Sometimes it is difficult to code PROC REG statements if amount of dependent variables or independent variables is large. Thanks to the INTO clause it would be quite easy to do by the following code.

```
PROC CONTENTS DATA=LIB.SASDATA NOPRINT
OUT=OUT(KEEP=NAME);
RUN;
```

The following statements managed to separate dependent variables and independent ones. Of course in reality it would not be so simple to classify them as 'IND%' or 'DEP%' in the WHERE clause. This is just an example.

```
PROC SQL NOPRINT;
SELECT * INTO :IND SEPARATED BY ' ' FROM
OUT WHERE NAME LIKE 'IND%';
SELECT * INTO :DEP SEPARATED BY ' ' FROM
OUT WHERE NAME LIKE 'DEP%';
QUIT;

PROC REG DATA=LIB.SASDATA SIMPLE;
MODEL &DEP=&IND;
RUN;
QUIT;
```

EXAMPLES OF DATA STEPS

INTO: clause can also make easier the DATA step processing.

Example 1:
Suppose a external file contains 100 names of flat files that would be concatenated and read into a SAS file. Usually it could be done by an INFILE option called 'FILEVAR'. INTO: clause is another alternative.
First create a SAS data set based on the external file with the names of flat files on it; then do:

```
PROC SQL NOPRINT;
SELECT QUOTE(FILENAME) INTO :FILNAME
SEPARATED BY ' ' FROM SASFILE;

FILENAME IN (&FILNAME) DISP=SHR;
DATA STEP;
INFILE IN OPTIONS;
INPUT VARNAME FORMAT.;
……;
RUN;
……;
```

The only thing left to be desired is the limit on the files to be concatenated is 200. For FILEVAR there is no such limit.

Example 2:
This example could be treated as an alternative to ARRAY, beside PROC TRANSPOSE.
Heavy involvement of IF/THEN statements in DATA steps might make program writers and users feel tedious. With the help of INTO: part (only part) of the hard work could be done without much hard coding.
Suppose the conditions go like this: IF A=1 OR B=1 OR C=1 OR … THEN OUTPUT; if the conditions involve hundreds of variables the following alternative should be taken:

```
PROC CONTENTS DATA=OLDDATA NOPRINT
OUT=OUT(KEEP=NAME);
RUN;
PROC SQL NOPIRNT;
SELECT NAME INTO :VAR SEPARATED BY '=1 OR '
FROM OUT WHERE CONDITION IS NOT NULL;
DATA NEWDATA;
SET OLDDATA;
IF &VAR=1;
RUN;
```

If some values of an old variable need to be checked out in order to assign a particular value to a new variable, for instance, IF ABC=0 OR ABC=2 OR ABC=5 OR… THEN NEWVAR=1, here is the code:

```
PROC CONTENTS DATA=OLDDATA…;
PROC SQL NOPIRNT;
SELECT 'ELSE IF ABC=' AS CONDITN, ABC INTO
:INA1-:INA99, :INB1-:INB99 FROM OUT WHERE
CONDITION IS NOT NULL;
QUIT;
%MACRO ASSIGN;
 %DO I=1 %TO &SQLOBS;
    &&INA&I"&&INB&I" THEN NEWVAR=1;
 %END;
%MEND;
DATA NEWDATA;
SET OLDDATA;
IF ABC='0' THEN NEWVAR=1;
%ASSIGN;
RUN;
```

I would like to take the opportunity to show my appreciation for the paper presented at NESUG'97, which is called 'Getting More Out of "INTO" in PROC SQL: An Example for Creating Macro Variables' by Mary-Elizabeth Eddlestone, SAS Institute Inc. I did benefit from the paper. However, I have some different opinion about the statements made in the section called 'The Problem: an Illustration'. Part of SAS statements in the section was:
……;
if sic='D12345' then D12345=1;
else if sic='D23456' then D23456=1; etc.
The author argued that hard coding of those conditions would be error prone, arduous and static instead of being dynamic. This is true and I agree. I think that pre-enhanced INTO: could handle the situation together with IF/THEN statements and would be another alternative beside post-enhanced INTO: plus ARRAY.
The code I suggested is the following:

```
PROC SQL NOPRINT;
SELECT SIC INTO :SIC1-:SIC999 FROM SASDATA;
QUIT;
%MACRO TEST;
 %DO I=1 %TO &SQLOBS;
     IF SIC="&&SIC&I" THEN &&SIC&I=1;
     ELSE &&SIC&I=0;
 %END;
%MEND;
DATA SASDATA;
SET SASDATA;
%TEST;
RUN;
QUIT;
```

INTO: AND &SYSPBUFF

Enhanced INTO: could work together with automatic macro variable SYSPBUFF to perform the function that pre-enhanced INTO: does.

If it is necessary to transform the values of some variables to blank or other characters of a SAS data set, here is one way to do it.

```
PROC CONTENTS DATA=ASUSUAL…
PROC SQL NOPRINT;
SEELCT NAME INTO :VAR SEPARATED BY ' ' FROM
OUT WHERE TYPE=2;
%MACRO CHNG(X)/PBUFF;
 %DO I=1 %TO &SQLOBS;
 %LET VARS=%SCAN(&SYSPBUFF,&I);
   &VARS=TRANSLATE(&VARS,' ','U');
 %END;
%MEND;
%CHNG(&VAR);
RUN;
QUIT;
```

The job could not be done by enhanced INTO: alone in view of the function TRANSLATE but could be done by pre-enhanced one.

CONCLUSION

INTO: clause (pre & post enhanced) and SYSPBUFF combined would be one of the most powerful tools ever produced by SAS. The more exploited more users would be released from tedious hard coding.

The author can be reached at 'jzhang79@yahoo.com'.