ABSTRACT

Is there an easy way to create a series of dummy variables whose names are the unique values of a character variable? Yes! This paper will show how PROC SQL can be used to generate a macro variable containing a list of unique values of a variable and how this macro can be used in conjunction with a do-loop to create and code all the variables required.

INTRODUCTION

We often hear that necessity is the mother of invention. When working in a software system as broad as the SAS® System, the axiom should probably be changed to “necessity is the mother of discovery”. Some of my best discoveries in the SAS System have been driven by people coming into my office and asking, “Do you know how to….?”, the seemingly simple question, with a seemingly simple answer just waiting to be discovered. This paper arose form one of those innocent questions: “Is it possible to create a series of dummy variables whose names are the unique values of a character variable in my data set?” This question led me to a powerful feature of PROC SQL, a procedure that, because of its slightly different syntax, is probably less familiar to many SAS programmers.

THE PROBLEM: AN ILLUSTRATION

In this example the character variable, SIC, represents a Standardized Industry Code (SIC). In order to comply with SAS variable naming conventions, the variable should be character and represent 8 or fewer bytes. In this case, a 7-digit SIC code was converted to character simply by concatenating the numeric value to the letter “D”.

\[
\begin{array}{c|c|c|c|c|c|c|c}
\hline
\text{SIC} & \text{D12345} & \text{D1234} & \text{D23456} & \text{D34567} & \text{D45678} \\
\hline
\text{D12345} & 1 & 0 & 0 & 0 & 0 \\
\text{D1234} & 1 & 0 & 0 & 0 & 0 \\
\text{D23456} & 0 & 1 & 0 & 0 & 0 \\
\text{D34567} & 0 & 1 & 0 & 0 & 0 \\
\text{D45678} & 0 & 0 & 1 & 0 & 0 \\
\hline
\end{array}
\]

This is what we needed to accomplish:

On the surface, this seemed to be a simple coding problem. But the simplicity of the question was deceptive. If there were only a handful of known unique values, the job could be done, theoretically, in a multi-step process that involves hard-coding the values into if-then statements:

```
data dummies;
set sic;
*initialize dummy variables to 0;
retain d12345 d23456 d34567 d45678 0;
*change value of dummy variables depending on value of SIC;
if sic="D12345" then D12345=1;
else if sic = "D23456" then D23456=1;
e tc.
```

Yes, the task could be accomplished. But:

- The need for so much typing leaves one open for typographical errors which could lead to further problems.
- This method becomes arduous if there are many unique values.
- If the data set is dynamic and the set of unique values changes, the SAS code would need to be rewritten for each modification to the data set.

For these and many other reasons, an automated method was sought.

CREATING A MACRO VARIABLE IN PROC SQL

Did you know that you can create a macro variable in PROC SQL? Look at the syntax for the SELECT statement and notice a magical keyword: INTO.

```
SELECT <DISTINCT> object-item <,object-item>...
<INTO macro-variable specification <,macro-variable specification> ...>
FROM from-list ...;
```

This magical word allows a programmer to place the result of a select statement into one or more macro variables. For example, the macro variables &style and &sqfeet are created from the following PROC SQL:

```
proc sql noprint;
select style, sqfeet
into :style, :sqfeet
from sasuser.houses;
```
LIMITATION

In Release 6.06 there was a limitation to this potentially powerful feature: only the first row of the SQL was put into the macro variable. If the SELECT statement resulted in more than one value, the values were not concatenated in any way: all but the first value were ignored. In some contexts, it might make sense to create a new macro variable for each row in the result of the select statement. A dash (-) is used in the INTO clause to imply a range or macro variables (or the keywords THROUGH or THRU):

```
proc sql noprint;
select distinct style, sqfeet
into :style1 - :style4,
      :sqfeet1 - :sqfeet3
from sasuser.houses;
```

Unfortunately, in the given context, the ability to create multiple macro variables did not help resolve the coding problems. A single macro variable containing a string of values was sought.

ENHANCEMENT

The solution to my problem was found in Changes and Enhancements to Version 6.11. A seemingly simple addition was made to the SELECT statement in PROC SQL: SEPARATED BY. It turned out to be a very powerful modification and just the tool I needed.

If a SELECT statement returns more than one value, the SEPARATED BY qualifier allows the user to tell SAS how to concatenate the values. The solution had revealed itself. Here was the tool that would produce an accurate list of unique values for me, even if the data set were dynamic. It also led me to a much better way of coding the dummy variables.

EXAMPLE AND USE

```
proc sql;
select distinct sic into :unique separated by ' ' 
          from mydata;
quit;
%put &unique;
d12345 d23456 d34567 d45678
```

This produced the desired macro variable, &unique, which contained a list of the distinct values of SIC separated by a space!

The next steps required were: (1) creating a variable for each of the distinct values; (2) assigning a default value of 0 to each variable; and (3) depending of the value of SIC recode the dummy variables to 1. It is important to remember that one of the desired outcomes was a solution that would be dynamic: the code shouldn't have to be changed if the composition of SIC changes.

Steps One and Two were straightforward. The new variables are created and assigned the value of 0 through the use of an array statement and a do-loop. The dynamic requirement is met in each: the array uses (*) to count the number of elements and the do-loop uses the DIM function to terminate the do-loop based on the dimensions of the array.

```
data newdata (drop=I);
  set mydata;
  /* step 1 */
  array temp(*) &unique;
  /* step 2 */
  do i=1 to dim(temp);
    temp(i)=0;
  end;
```

Step Three required a bit more ingenuity. A method was needed whereby I could evaluate the value of SIC and address the corresponding element of the array. Although the resulting statement looks complex, it is just a combination of SAS functions that return the position in the array for each observation of SIC and code that element to be 1.

```
/* step 3 */
temp(int(index("&unique",trim(sic)) /
   (length(sic)+1)))=1;
```

trim(sic): trims any trailing spaces in SIC.
index("&unique",trim(sic)): searches &unique macro and returns the column location of the trimmed value of SIC.
length(sic)+1: returns the number of characters in SIC plus one for the space separating each SIC (total length).
int(…): returns the integer value from the division of the index value and the total length.

CONCLUSION

The ability to place the results of a PROC SQL select statement into a macro variable is a powerful feature that provides opportunities for solving data coding problems, limited only by the programmer’s imagination.

The author can be reached for comment at sasmee@unx.sas.com.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ©indicates USA registration.

A special thanks to Michele Austin in Technical Support at SAS Institute for her assistance and guidance.

i You can check the PROC SQL macro variable SQLOBS to see the number of rows produced by a query-expression. See “Macro Variables Set by Statements in the SQL Procedure” on page 118 of the SAS Guide to the SQL Procedure for more information.