SAS® GLOBAL FORUM 2019

USERS PROGRAM

APRIL 28 – MAY 1, 2019 | DALLAS, TX

INTO:

Creating Lists! Using the Powerful INTO Clause with PROC SQL to Store Information in Macro Lists

Julie Melzer

Educational Testing Service

Abstract & Overview
Methods
Methods 2
Methods 3
Conclusion

Abstract

Lists can be invaluable for numerous operations in SAS® including dropping or altering many columns in a data set or removing all rows containing particular values of a variable. However, typing a long list of values can be tedious and error prone. One potential solution is to use the INTO clause with the SQL procedure to assign a single value or multiple values to a macro variable and thereby reduce this manual work along with potential typographical errors or omissions. The INTO clause even has additional benefits such as the ability to retrieve and store data set characteristics into a macro variable and the ability to easily customize formats, which makes this a great tool for a wide variety of uses.

Topic overview

- Storing a single value
- Storing multiple values
- Storing column values
- Storing row values

Creating Lists! Using the Powerful INTO Clause with PROC SQL to Store Information in Macro Lists

Julie Melzer

Educational Testing Service

INTO:

Abstract & Overview
Methods
Methods 2
Methods 3
Conclusion

Options

TRIMMED – Used to trim any leading and trailing blanks from the stored value.

DISTINCT – Retrieves only unique values.

Storing a Single Value

- Information about the dataset can be gathered and stored into a macro variable with the use of a summary function.
- The INTO clause in combination with the Count function is used to find the number of records in the file. The macro variable 'Number_Rows' is created and contains the number of rows in the Sashelp.Cars dataset.
- The value of the macro variable is printed to the log using %put.
- This is not exclusive to 'Count' and any other similar summary function will also work in its place.

Sample Code

```
proc sql;
  select Count(*)
  into :Number_Rows trimmed
  from Sashelp.Cars;
quit;

%put &Number_Rows;
```

LOG

428

Storing Multiple Values

- Multiple features of the data can be summarized through the use of several summary functions.
- Characteristics of the Sashelp.Cars dataset are extracted by using the Count, Mean, and Max functions.
- Options are used to put the results into a useful format. 'Trimmed' will remove any leading and trailing blanks from the stored values. The 'distinct' keyword will pull only unique values.
- The stored values are global macro variables that can be called upon at any time throughout the program.

Sample Code

```
proc sql;
  select Count(distinct Make), Mean(MSRP) format DOLLAR9.2,
         Max(Horsepower)
  into :Count_Make trimmed, :Mean_MSRP trimmed,
       :Max_Horsepower trimmed
  from Sashelp.Cars;
quit;

%put "The dataset represents &Count_Make brands of cars with average
MSRP of &Mean_MSRP and max horsepower of &Max_Horsepower..";
```

LOG

"The dataset represents 38 brands of cars with average MSRP of $32774.86 and max horsepower of 500."

INTO:

Creating Lists! Using the Powerful INTO Clause with PROC SQL to Store Information in Macro Lists

Julie Melzer

Educational Testing Service

Abstract & Overview
Methods
Methods 2
Methods 3
Conclusion

Options

SQLOBS – Automatic macro variable that contains the number of rows produced by the SQL procedure.

SEPARATED BY – Separates stored values by specified character (automatically trims leading and trailing blanks unless specify NOTRIM).

Storing Multiple Values (Continued)

- A range of macro variables each with a distinct value can be created by separating the names of the variables with a hyphen.
- When unsure of the upper bound of the range, a number larger than the actual value can be specified or the upper bound left blank.
- The unique values of Origin from the Sashelp.Cars dataset are assigned to macro variables. The number of origins represented in the data is unknown, so an upper bound of 99 is used (a value larger than the actual value). The 3 origin values are stored in Origin1, Origin2, and Origin3.

Storing Column Values

- Dictionary tables can be used to gather a list of variables based on fields such as variable name or variable format.
- A list of numeric variables that are written with a leading $ sign can be obtained. Column names where the format is DOLLAR are extracted and inserted into macro variable ‘Column_List.’
- The list is formatted into values separated by a comma. The ‘separated by’ option separates stored values by a specified character and trims leading/trailing blanks.
- The macro variable can be easily called to perform further manipulation of the DOLLAR formatted columns.

Sample Code

```
%macro Origin;
  proc sql;
    select distinct(Origin) into :Origin1 - :Origin99
    from Sashelp.Cars;
  quit;
  %do i=1 %to &sqlobs;
    %put &i &&Origin&i;
  %end;
%mend Origin;
%Origin;
```

LOG

```
1 Asia
2 Europe
3 USA
```

Sample Code

```
proc sql noprint;
  select name
  into :Column_List separated by ' , '
  from dictionary.columns
  where libname="SASHELP" and memname="CARS" and
        format contains("DOLLAR");
quit;

%put &Column_List;
```

LOG

```
MSRP , Invoice
```

INTO:

Creating Lists! Using the Powerful INTO Clause with PROC SQL to Store Information in Macro Lists

Julie Melzer

Educational Testing Service

Abstract & Overview
Methods
Methods 2
Methods 3
Conclusion

Options

SEPARATED BY – Separates stored values by specified character (automatically trims leading and trailing blanks unless specify NOTRIM).

QUOTE – Separates stored values by quotations.

Storing Row Values

- Rows containing particular values of a variable can be dropped or kept.
- In the example, there is a list of 4 car models in the MyCars dataset. The INTO clause is used to create macro variable 'Model_List' with a list of the 4 car models. Afterwards a data step is used to limit the larger file to only rows pertaining to those specific models.
- If the INTO statement results in multiple values then the 'separated by' option can be used to separate those values into a desired format.
- As with the other examples shown, this method allows the data to speak for itself and does not require input from the user. This saves time and is far less risky than manual entry.
- This code is flexible and does not need to be revised if the list of car models in the MyCars dataset changes.

Sample Code

```
data MyCars;
  input @1 Make $13. @14 Model $40.;
  datalines;
Honda         Accord LX V6 4dr
Chevrolet     Malibu LT 4dr
Subaru        Legacy GT 4dr
Toyota        Camry LE V6 4dr
  ;
run;

proc sql noprint;
  select quote(trim(Model))
  into :Model_List separated by ','
  from MyCars;
quit;
%put &Model_List;

data MyCars_Stats; /*Limit dataset to only Models in MyCars list*/
  set Sashelp.Cars;
  where strip(Model) in (&Model_List);
run;
```

LOG

"Accord LX V6 4dr" , "Malibu LT 4dr" , "Legacy GT 4dr" , "Camry LE V6 4dr"

Make	Model	Type	Origin	DriveTrain	MSRP	Invoice	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
Chevrolet	Malibu LT 4dr	Sedan	USA	Front	$23,495	$21,551	3.5	6	200	23	32	3315	106	188
Honda	Accord LX V6 4dr	Sedan	Asia	Front	$23,760	$21,428	3	6	240	21	30	3349	108	190
Subaru	Legacy GT 4dr	Sedan	Asia	All	$25,645	$23,336	2.5	4	165	21	28	3395	104	184
Toyota	Camry LE V6 4dr	Sedan	Asia	Front	$22,775	$20,325	3	6	210	21	29	3296	107	189

Resulting MyCars_Stats dataset

INTO:

Creating Lists! Using the Powerful INTO Clause with PROC SQL to Store Information in Macro Lists

Julie Melzer

Educational Testing Service

Abstract & Overview
Methods
Methods 2
Methods 3
Conclusion

Benefits

- The INTO clause easily assigns values to one or more macro variables for use later in the program by a DATA or PROC step.
- This process is flexible enough that it can store a wide variety of information including data set characteristics, column values, and row values.
- Many values can be gathered into a list without needing to type out all of the values. This saves time and reduces tediousness, as well as eliminates the risk of typographical error or omission of a value.
- The formats of the lists created can be easily customized using options such as TRIMMED, QUOTE, and SEPARATED BY.
- This approach produces global macro variables which can be called upon at any time throughout the program.
- Less hard coding of values is needed with this method, leading to greater flexibility and simpler code.
- The INTO clause used with the SQL procedure allows for quick and efficient manipulation to large datasets.