

Can't Decide Whether to Use a DATA Step or PROC SQL? You Can Have It Both Ways with the SQL Function!

Jeremy W. Poling, B&W Y-12 L.L.C., Oak Ridge, TN

ABSTRACT

Have you ever thought that it would be nice if you could execute a PROC SQL SELECT statement from within a DATA step? Well, now you can! This paper describes how to create a DATA step SQL function using the FCMP procedure and the RUN_MACRO function. The SQL function accepts a character expression analogous to a SELECT statement as its only argument. By using the SQL function, you now have the ability to integrate the DATA step and the SQL procedure.

INTRODUCTION

A SAS® function accepts a set of arguments and returns a value that can be used in an expression or an assignment statement. As you probably know, SAS programmers have an abundance of useful built-in functions at their disposal. SAS 9.2 software enhancements to the FCMP procedure made it possible to create even more user-defined functions that can be used in the DATA step. One useful feature of the FCMP procedure is the RUN_MACRO function, which is used to execute a predefined SAS macro. A macro executed through the RUN_MACRO function can contain complete DATA steps and/or PROC steps. We will use the RUN_MACRO function to execute a macro that contains a PROC SQL step. The result is a flexible user-defined function that can execute a PROC SQL step from within the DATA step.

CREATING THE SQL FUNCTION

Before we create the SQL function with the FCMP procedure, we must create the macro that contains the PROC SQL step to be executed. We will execute the following macro using the RUN_MACRO function in the FCMP procedure:

```
%macro sqlfunc;  
  %let sqlselect=%sysfunc(dequote(&sqlselect)); 1  
  %let sqlresults = ; 2  
  proc sql noprint;  
    create view _TempView_ as &sqlselect; 3  
    select * into: sqlresults separated by " " from _TempView_; 4  
    drop view _TempView_; 5  
  quit;  
  %let sqlresults=&sqlresults; 6  
%mend;
```

The %SQLFUNC macro uses a macro variable, named **sqlselect**, which will be created by the FCMP procedure. The **sqlselect** macro variable will resolve to a PROC SQL SELECT statement, enclosed in quotes. This SELECT statement must return only a single column. Notes pertaining to selected SAS statements in the macro are:

#	Notes
1	Removes the quotes from the value of the sqlselect macro variable (i.e. the SELECT statement).
2	Initializes a macro variable, named sqlresults , by assigning it a null value.
3	Creates a temporary view, named _TempView_ , based on the query.
4	Updates the macro variable sqlresults based on the results of the query. If more than one value is returned by the query, then the values are delimited with spaces.
5	Deletes the _TempView_ view.
6	Removes leading and trailing blanks from the sqlresults macro variable. The resulting value of the sqlresults macro variable will be the value returned by the user-defined SQL function.

If you intend to use the SQL function in multiple applications, the %SQLFUNC macro should be stored as an autocall macro.

The following code creates the SQL function using the FCMP procedure:

```
proc fcmp outlib=sasuser.funcs.general; 1
  function sql(sqlselect $) $; 2
  length sqlresults $ 32767; 3
  rc=run_macro('sqlfunc', sqlselect, sqlresults); 4
  return(sqlresults); 5
endsub; 6
quit;
```

Notes pertaining to selected statements in the FCMP procedure are:

#	Notes
1	Invokes the FCMP procedure. The OUTLIB= option specifies the name of a data set to which the compiled SQL function will be written.
2	Declares a character function, named SQL, which accepts one character argument, named <i>sqlselect</i> .
3	Initializes a PROC FCMP character variable named sqlresults with a length of 32,767. This length is the maximum length of a character variable. A length of 32,767 is acceptable if you are using SAS 9.3 software. However, if you are using SAS 9.2 software, the maximum allowable length is only 260.
4	Executes the %SQLFUNC macro we defined previously. Before SAS executes the macro, macro variables named sqlselect and sqlresults are created. The sqlselect macro variable is given the same value as the <i>sqlselect</i> argument supplied. The %SQLFUNC macro executes the PROC SQL step and sets the value of the sqlresults macro variable. After SAS executes the macro, the value of the sqlresults macro variable is copied back to the corresponding PROC FCMP variable.
5	Returns the value of the sqlresults PROC FCMP variable.
6	Ends the definition of the SQL function.

EXAMPLE OF USING THE SQL FUNCTION

For example purposes, suppose we have three SAS data sets, named **Employee**, **Sales**, and **SaleProducts**, defined by the following SAS code:

```
data Employee;
input EmployeeID $ Name $ StartDate:mmddyy10.;
datalines;
E40624 John 08/12/2009
E37421 Jane 11/14/2010
E00525 Joe 01/05/2011
E73904 Alice 07/17/2011
E62963 Peter 03/7/2011
;
run;
```

```

data Sales;
input PurchaseOrderID EmployeeID $ Date:mmddyy10.;
datalines;
4629736 E40624 09/15/2009
2947377 E40624 10/23/2009
3536363 E40624 06/19/2010
0542063 E40624 03/30/2011
6297432 E37421 05/17/2011
6420765 E00525 03/12/2011
3265329 E62963 04/13/2011
5639763 E62963 05/03/2011
;
run;

data SaleProducts;
input PurchaseOrderID ProductID $ Qty Price;
datalines;
4629736 P7537 1 3275.70
2947377 P3452 1 1056.24
2947377 P3525 1 4670.00
3536363 P3525 3 4670.00
0542063 P3763 1 722.20
6297432 P6446 5 10544.00
6420765 P6867 4 589.00
3265329 P5156 1 1037.20
3265329 P7537 1 3275.70
3265329 P6867 2 589.00
5639763 P1648 1 1897.00
;
run;

```

Suppose that we want to create a new data set, named **NewEmployeeSales**. A “new employee” is defined as an employee whose start date is after January 1, 2011. The data set is to contain the identification numbers and names of new employees, the purchase order identification numbers and dates for any sales made by the new employees, and the total dollar amount of each sale. Additionally, the data set is to contain a variable for the cumulative dollar amount of all sales made by each new employee.

The following code creates the **NewEmployeeSales** data set. The code also illustrates a few ways that the SQL function can be used.

```

options cmplib=sasuser.funcs; 1

proc sort data=Sales;
  by EmployeeID Date;
run;

data NewEmployeeSales(drop=NewEmployees);
  length NewEmployees $ 200 Name $ 8; 2
  retain NewEmployees Name;

```

```

if _n_ = 1 then NewEmployees = sql("select EmployeeID
                                   from Employee
                                   where StartDate >= '01Jan2011'd"); 3

set Sales;
by EmployeeID;

if find(NewEmployees, strip(EmployeeID)); 4
if first.EmployeeID then do;

    call missing(Name, CumulativeSalePrice); 5
    name = sql("select Name
               from Employee
               where EmployeeID=" || quote(EmployeeID)); 6

end;
TotalSalePrice = input(sql("select sum(Qty*Price)
                           from SaleProducts
                           where PurchaseOrderID=" || PurchaseOrderID), best10.); 7

CumulativeSalePrice + TotalSalePrice; 8
format Date mmddyy10. TotalSalePrice CumulativeSalePrice dollar10.2;

run;

```

Notes pertaining to selected statements in the program are:

#	Notes
1	The CMPLIB= option tells SAS where to look for previously compiled functions. You can also set this option in the SAS configuration file or a SAS autoexec file.
2	Defines the length of character variables which are assigned a value based on the results returned by the SQL function. If a length is not previously defined, these variables are assigned a length of 32,767.
3	Uses the SQL function to select the employee identification numbers for all employees whose start date is after January 1, 2011. The results are stored in the NewEmployees variable. This statement is executed only during the first iteration of the DATA step. Because the NewEmployees variable is specified in a RETAIN statement, its value will be available during subsequent iterations of the DATA step.
4	The subsetting IF statement eliminates any observation that does not correspond to a new employee. For new employee observations, the value of EmployeeID , which is read into the program data vector from the Sales data set, can be found in the value of the NewEmployees variable, which was established with the SQL function.
5	Resets the values of the variables Name and CumulativeSalePrice to missing on the first observation for each new employee.
6	Uses the SQL function to obtain the name of the employee on the first observation for each new employee. The result is stored in the Name variable. Because the Name variable is specified in a RETAIN statement, its value will be retained for subsequent observations processed for the same employee.
7	Uses the SQL function to compute the total dollar amount of each sale for the new employees. Because the SQL function always returns character values, the INPUT function is used to convert the results to a number. The result is stored in the TotalSalePrice variable.
8	Computes the cumulative sale price of all sales for the new employee.

The code used to print the resulting data set follows. The output generated by the PRINT procedure is shown in Figure 1.

```

proc print data=NewEmployeeSales noobs;
  by EmployeeID;
  var Name PurchaseOrderID Date TotalSalePrice CumulativeSalePrice;
  Title "Sales by New Employees";
run;

```

Sales by New Employees				
EmployeeID=E00525				
Name	PurchaseOrderID	Date	TotalSalePrice	CumulativeSalePrice
Joe	6420765	03/12/2011	\$2,356.00	\$2,356.00
EmployeeID=E62963				
Name	PurchaseOrderID	Date	TotalSalePrice	CumulativeSalePrice
Peter	3265329	04/13/2011	\$5,490.90	\$5,490.90
Peter	5639763	05/03/2011	\$1,897.00	\$7,387.90

Figure 1: PROC PRINT output of the **NewEmployeeSales** data set

CONSIDERATIONS WHEN USING THE SQL FUNCTION

The SQL function presented in this paper is very flexible. The few examples that we have discussed hardly scratch the surface of what is possible with the SQL function. For example, the SELECT statement specified in the SQL function argument could join multiple data sets together or it could contain subqueries. However, there are some limitations and other considerations to the SQL function which should be discussed.

The SELECT statement passed as an argument to the SQL function can return only a single column. If it is necessary for the SELECT statement to return multiple variables from a data set, then all the variables can be concatenated together using the CATX function so that only a single column is returned. The value returned by the SQL function can then be parsed into individual DATA step variables using the SCAN function.

The SQL function always returns a character value. If a numeric value is needed instead, the INPUT function must be used to convert the result to a number. Be aware that, due to the usage of macro variables, loss of numeric precision may occur if the result contains many significant digits.

The maximum length of the value returned by the SQL function is 32,767 when using SAS 9.3 software. This length is the maximum length of a character variable. Therefore, the SQL function cannot be used when the SELECT statement returns an excessively large number of values. When using SAS 9.2 software, the maximum length of the value returned by the SQL function is only 260.

If the length of the variable has not been previously defined, a default length of 32,767 will be assigned to any variable created using the SQL function in an assignment statement. A LENGTH statement can be used in the DATA step to define the variable's length before the SQL function is used in the assignment statement.

If you encounter problems when attempting to use the SQL function and find it necessary to debug your applications, the use of the MPRINT option is recommended. This option will write the text generated by the %SQLFUNC macro to the SAS log, which will enable you to view the code for every PROC SQL step generated by the macro.

If the SQL function is used unconditionally within a DATA step, the PROC SQL step generated by the %SQLFUNC macro will be executed once for each iteration of the DATA step. As a result, the execution time of the DATA step could increase dramatically. Care should be taken to ensure that the same argument to the SQL function is not used during multiple iterations of the DATA step. For example, you may need to use the SQL function only during the first iteration of the DATA step or you may need to use the SQL function only for the first observation of each BY group.

The SQL function is a convenient tool that can often simplify the appearance of your code. However, from a program execution time perspective, there are typically more efficient techniques to programming than using the SQL function. If efficiency is a concern, one way to modify the SQL function to significantly improve performance is to avoid creating the temporary view in the %SQLFUNC macro. The alternative code for the %SQLFUNC macro follows:

```
%macro sqlfunc;
  %let sqlselect=%sysfunc(dequote(&sqlselect));
  %let sqlresults = ;
  proc sql noprint;
    &sqlselect;
  quit;
  %let sqlresults=&sqlresults;
%mend;
```

However, the gain in efficiency comes at some expense to user-friendliness. If the temporary view is not created in the %SQLFUNC macro, then you must include an INTO: SQLRESULTS clause in the *sqlselect* argument whenever the function is used. The following statement is an example of using the SQL function in conjunction with the more efficient alternative %SQLFUNC macro:

```
name = sql("select Name into:sqlresults
           from Employee
           where EmployeeID=" || quote(EmployeeID));
```

CONCLUSIONS

The FCMP procedure and the RUN_MACRO function are powerful programming tools. Using PROC FCMP and the RUN_MACRO function, a flexible user-defined SQL function can be created that integrates the PROC SQL SELECT statement with the DATA step.

REFERENCES

Eberhardt, Peter, "A Cup of Coffee and PROC FCMP: I Cannot Function Without Them." Proceedings of the SAS Global Forum 2009 Conference. Available at <http://support.sas.com/resources/papers/proceedings09/147-2009.pdf>

SAS Institute Inc. 2011. "Base SAS® 9.3 Procedures Guide." Available at <http://support.sas.com/documentation/cdl/en/proc/63079/HTML/default/viewer.htm#titlepage.htm>

ACKNOWLEDGEMENTS

The author would like to thank his coworkers for taking the time to review the content of this paper as well as his management for their support, especially Leah Cox, Steve McGuire, Rachel Hayes, and Amy Wilson.

CONTACT INFORMATION

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

Jeremy W. Poling
B&W Y-12 L.L.C.
Phone: 865-574-4610
E-mail: jpoling@vt.edu

DISCLAIMER AND COPYRIGHT NOTICE

This work of authorship and those incorporated herein were prepared by B&W Y-12 L.L.C. as accounts of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor B&W Y-12 L.L.C, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, use made, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency or B&W Y-12 L.L.C thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency or B&W Y-12 L.L.C. thereof.

This document has been authored by a contractor/subcontractor of the U.S. Government under contract DE-AC05-00OR-22800. Accordingly, the U.S. Government retains a paid-up, nonexclusive, irrevocable, worldwide license to publish or reproduce the published form of this contribution, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, or allow others to do so, for U. S. Government purposes.

SAS and all other SAS Institute Inc. product or service names are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.