SESUG Paper 192-2019

Three Ways to Transform Your Code into PROC SQL®

Charity Wilson, Cobb EMC

ABSTRACT

Does learning PROC SQL® seem overwhelming? Or maybe you feel that it is not necessary? But did you know that PROC SQL is a powerful tool used to manipulate data in a dataset all in one procedure? This paper will introduce PROC SQL and demonstrate how to easily transform common DATA and PROC Steps you already use into PROC SQL code.

Why would you want to use PROC SQL instead of DATA and PROC Steps you may ask? You can accomplish multiple tasks in one SELECT statement, easily join and append datasets, automatically print results without the use of a PROC Print and utilize a time saving secret weapon – no presorting your dataset. Topics covered are great for novice SAS users to legacy DATA Step programmers.

INTRODUCTION

SQL, Structured Query Language, has emerged as one of the most popular programming languages used by analyst to find answers in data. PROC SQL must begin with a SELECT statement that includes the column name(s) and the FROM clause that references the source table(s). Several sub-clauses can be used, but they are optional. Sub-clauses are used to subset, group and sort the final output.

This paper will present three business scenarios using the CARS dataset from the SASHELP library. This allows you to utilize the same code and practice when you go back to your company. Solutions to the scenarios will be shown using DATA/PROC Steps. Then, the same results will be accomplished with PROC SQL to demonstrate efficiencies gained with fewer Steps.

BASIC PROC SQL SYNTAX

The code below is basic syntax for PROC SQL:

```
PROC SQL;
   SELECT column-1, …, column-n
   FROM table-name
      WHERE expression
      GROUP BY column(s)
      ORDER BY column(s);
   QUIT;
```

PROC SQL Statement/ Clause	Description
SELECT Statement	Indicates the column(s) and rows of data retrieved from the source table
FROM Clause	Source table
WHERE Clause	Specifies conditions that each row must met to be included in the result table
GROUP BY Clause	Groups the data for summarizing or aggregating
ORDER BY Clause	Orders the rows of the result table by specified column(s)
QUIT Statement	Ends the SQL procedure

Table 1 Description of PROC SQL® statements and clauses.

ADDITIONAL PROC SQL SYNTAX

Additional options of PROC SQL that will be used in this paper.

PROC SQL Statement/ Clause	Description
CREATE TABLE Statement	Creates a table with desired results
Asterisk (*) on SELECT Statement	Selects all columns from the Source table
CASE Clause	Allows conditional processing
DESC in ORDER BY Clause	Specifies descending order for sorting
UNION Operator	Appends the rows of two or more tables

Table 2. Additional description of PROC SQL statements and clauses.

DIFFERENCES IN PROC SQL AND DATA/PROC STEPS

Understanding key difference between DATA Steps and PROC SQL will help ensure your success as you venture into the world of PROC SQL. Below are a few important differences in terminology and syntax structure, but they both have the same meaning.

DATA/PROC Step	PROC SQL
Dataset	Table
Variable	Column
Observation	Row
RUN Statement	QUIT Statement
Semicolon at the end of each statement	Semicolon listed at the end of a SELECT statement

Table 3 Differences between DATA Step and PROC SQL.

SCENARIO 1: IN OPERATOR WITH SORT OPTION

The IN operator allows for filtering a dataset that can have many possible values in a single column. In PROC SQL, sorting the data is performed using the ORDER BY clause. Ascending order is designated with ASC, but it is the default sort order and does not need to be specified. While descending order is designated with DESC and placed after the column name.

Suppose you need to create a report that list all the SUVs with a Make of Ford, GMC and Dodge. The goal is to find which Model has the lowest MSRP and gets the most miles per gallon in the City. The data needs to be sorted in ascending order by MSRP and descending order by MPG_City.

SOLUTION 1: DATA/PROC STEP

```
DATA SUVLowMSRP(KEEP=Make Model Type MSRP Invoice MPG_City);
SET SASHELP.CARS;
      IF Make IN('Dodge','Ford','GMC')
                  and Type = 'SUV';
RUN;

PROC SORT DATA= SUVLowMSRP;
      BY MSRP DESCENDING MPG_City;
RUN;
```

SOLUTION 1: PROC SQL

```
PROC SQL;
   SELECT Make, Model, Type, MSRP, Invoice, MPG_City
   FROM SASHELP.CARS
      WHERE Make IN('Dodge','Ford','GMC')
            AND Type = 'SUV'
      ORDER BY MSRP, MPG_City DESC;
QUIT;
```

Output 1:

Make	Model	Type	MSRP	Invoice	MPG (City)
Ford	Escape XLS	SUV	$22,515	$20,907	18
Ford	Explorer XLT V6	SUV	$29,670	$26,983	15
GMC	Envoy XUV SLE	SUV	$31,890	$28,922	15
Dodge	Durango SLT	SUV	$32,235	$29,472	15
Ford	Expedition 4.6 XLT	SUV	$34,560	$30,468	15
GMC	Yukon 1500 SLE	SUV	$35,725	$31,361	16
Ford	Excursion 6.8 XLT	SUV	$41,475	$36,494	10
GMC	Yukon XL 2500 SLT	SUV	$46,265	$40,534	13

Output 1 For Scenario 1 from using a DATA Step or PROC SQL statement.

SCENARIO 2: APPEND DATASETS AND SORT

Suppose two stores that both sell Mercedes-Benz are closing and consolidating into one store; both stores sell SUVs and Sedans, but Store A sells Wagons while Store B sells Sports. The goal is to create a new table called StoreC that combine the two tables into one and removes all duplicates. You need a report that shows a list of all the cars that will be sold at the new location sorted by MSRP in ascending order.

Note: You will see two solutions (A and B) for DATA/PROC step in this scenario.

CREATING TABLES FOR STORE A AND STORE B

You will use the code below to create two tables from the CARS: StoreA and StoreB. The two tables are a subset of the CARS and only include vehicles with the Make Mercedes-Benz. StoreA should contain 21 rows and 15 columns. StoreB should contain 23 rows and 15 columns. An example of the CREATE TABLE statement is used in the code below.

```
PROC SQL;
CREATE TABLE StoreA as
   SELECT Make, Model, Type, MSRP, MPG_City
   FROM sashelp.cars
      WHERE Type IN('SUV','Sedan','Wagon') AND Make = 'Mercedes-Benz';
QUIT;

PROC SQL;
CREATE TABLE StoreB as
   SELECT Make, Model, Type, MSRP, MPG_City
   FROM sashelp.cars
     WHERE Type IN('SUV','Sedan','Sports') AND Make = 'Mercedes-Benz';
QUIT;
```

SOLUTION 2A: DATA/PROC STEP

```
PROC SORT DATA=StoreA OUT=StoreA_sort;
     BY Model;
run;

PROC SORT DATA=StoreB OUT=StoreB_sort;
     BY Model;
run;

DATA StoreC;
     MERGE StoreA_sort StoreB_sort;
     BY Model;
RUN;

PROC SORT DATA=StoreC;
     BY MSRP;
RUN;
```

SOLUTION 2B: DATA/PROC STEP

```
PROC APPEND BASE=StoreA DATA=StoreB;
RUN;

PROC SORT DATA=StoreA OUT=StoreC NODUP;
      BY MSRP;
RUN;
```

SOLUTION 2: PROC SQL

To display all the columns of a table, you can use the asterisk (*) wildcard on the SELECT list instead of typing the name of every column. Asterisk used in the example below:

```
PROC SQL;
CREATE TABLE StoreC AS
      SELECT *
      FROM StoreA
            UNION
      SELECT *
      FROM StoreB
            ORDER BY MSRP;
QUIT;
```

Output 2:

Make	Model	Type	MSRP	MPG (City)
Mercedes-Benz	C230 Sport 2dr	Sedan	$26,060	22
Mercedes-Benz	C320 Sport 2dr	Sedan	$28,370	19
Mercedes-Benz	C240 4dr	Sedan	$32,280	20
Mercedes-Benz	C240 4dr	Sedan	$33,480	19
Mercedes-Benz	C240	Wagon	$33,780	19
Mercedes-Benz	C320 Sport 4dr	Sedan	$35,920	19
Mercedes-Benz	C320 4dr	Sedan	$37,630	20
Mercedes-Benz	C320 4dr	Sedan	$38,830	19

Output 2 For Scenario 2 from using a DATA Step or PROC SQL statement (partial output).

SCENARIO 3: IF/THEN/ELSE LOGIC VS CASE OPTION

Use the CASE expression when you want to perform conditional processing within PROC SQL. It is like the IF/THEN/ELSE logic in a DATA Step. The syntax is below:

CASE

WHEN (condition) **THEN** (result)

WHEN (condition) **THEN** (result)

ELSE (result)

END AS new column-name

If the condition in the **WHEN** clause is true, then the result of the **THEN** clause is executed for each row. The following **WHEN** clause is skipped, and PROC SQL moves to the next row. If the condition in the **WHEN** clause is false, Proc SQL evaluates the next **WHEN** clause and so on. If all the **WHEN** clauses are false, PROC SQL will execute the result of the **ELSE** clause. The CASE option will return a missing value if all **WHEN** clauses are false, and no **ELSE** clause is present. Lastly, use the optional **AS** keyword to specify the new column name.

Suppose your company wants to direct marketing dollars to all the Mercedes-Benz cars that are less than $50,000 and get at least 15 MPG in the city. You need to send a report to Marketing that labels which vehicle Model gets marketing dollars. Marketing needs the report sorted by Type, MSRP in ascending order and MPG_City in descending order.

SOLUTION 3: DATA/PROC STEP

```
DATA Marketing(KEEP=Make Model Type MSRP MPG_City Marketing);
SET storec;

      IF MSRP < 50000 AND MPG_City > 15 THEN
            Marketing = 'Yes';
      ELSE
            Marketing = 'No';
RUN;

PROC SORT DATA=Marketing;
      BY Type MSRP DESCENDING MPG_City;
RUN;
```

SOLUTION 3: PROC SQL

```
PROC SQL;
   CREATE TABLE Marketing AS
   SELECT Make, Model, Type, MSRP, MPG_City,
            (CASE
               WHEN MSRP < 50000 and MPG_City > 15
               THEN 'Yes'
               ELSE 'No'
            END) AS Marketing
      FROM STOREC
      ORDER BY Type, MSRP, MPG_City DESC;
QUIT;
```

Output 3:

Make	Model	Type	MSRP	MPG (City)	Marketing
Mercedes-Benz	ML500	SUV	$46,470	14	No
Mercedes-Benz	G500	SUV	$76,870	13	No
Mercedes-Benz	C230 Sport 2dr	Sedan	$26,060	22	Yes
Mercedes-Benz	C320 Sport 2dr	Sedan	$28,370	19	Yes
Mercedes-Benz	C240 4dr	Sedan	$32,280	20	Yes
Mercedes-Benz	C240 4dr	Sedan	$33,480	19	Yes
Mercedes-Benz	C320 Sport 4dr	Sedan	$35,920	19	Yes
Mercedes-Benz	C320 4dr	Sedan	$37,630	20	Yes

Output 3 For Scenario 3 from using a DATA Step or PROC SQL statement (partial output).

CONCLUSION

The three scenarios, above, have shown how a DATA/PROC Step can easily be transformed into PROC SQL. Plus, it can be way more efficient! I hope you consider adding this tool to your programming tool belt.

PROC SQL advantages:

- used fewer lines of codes
- required no presorting
- completed multiple task in one Step

REFERENCES

SAS® Procedures Guide, Version 9.2; SAS Institute Inc., Cary, NC.

SAS® Language Reference: Concepts, Second Edition, Version 9.2; SAS Institute Inc., Cary, NC.

SAS® DATA Step Statements: Reference, Version 9.4; SAS Institute Inc., Cary, NC.

Kirk Paul Lafler. 2011. "Conditional Processing Using the Case Expression in PROC SQL" WUSS 2011 Conference, Software Intelligence Corporation, Spring Valley, CA.

Katie Minten, Ronk, Steve First, David Bea. 2002. "An Introduction to PROC SQL" SUGI 27 Conference, Systems Seminar Consultants, Inc., Madison, WI.

ACKNOWLEDGMENTS

I would like to thank Kirk Lafler for his help and review of this paper.

CONTACT INFORMATION

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

Charity Wilson
Cobb EMC
1000 EMC Pkwy
Marietta, GA 30060
Charity.wilson@cobbemc.com