Paper B2B 9

A Quick and Gentle Introduction to PROC SQL

Shane Rosanbalm, Rho, Inc.

Sam Gillett, Rho, Inc.

ABSTRACT

If you are afraid of SQL, it is most likely because you haven't been properly introduced. Maybe you've heard of SQL but never quite had the time to explore. Or, maybe you were introduced to SQL, but it was at the hands of a database administrator turned SAS programmer who liked to write un-indented queries that were 60-lines long and combined 15 different datasets; egads! The goal of this paper is to provide an introduction to PROC SQL that is both quick and gentle. We will not discuss joins, subqueries, cases, calculated, feedback, return codes, or anything else that might dampen your initial momentum. Don't get me wrong, these are all wonderful topics. But they are better suited to a paper in the Hands-on Workshop or Beyond the Basics sections. Our focus will be limited to introductory topics such as select, from, into, where, group by, and order by. Once these basic topics have been introduced, we will use them to demonstrate some simple yet powerful applications of SQL. The applications will focus on situations in which the use of SQL will result in code that is both more efficient and more concise than what can be achieved when limiting oneself to the DATA step and other non-SQL procedures.

ORIENTATION

In this paper we will first do a short sprint through the fundamental concepts of SQL. After this short sprint we will regroup, pulling this initial bolus of information together in a more consolidated form, as well as elaborating on some of the finer points associated with these fundamental concepts. With these fundamentals in hand, we will then move on to some slightly more interesting features of SQL.

THE FUNDAMENTALS

Here begins a short sprint through the fundamental concepts of SQL. We begin with the simplest possible application of SQL and then layer on additional fundamentals one at a time.

SELECT FROM

We begin by comparing PROC SQL to PROC PRINT. Here is a straightforward application of PROC PRINT.

```
proc print data=sashelp.cars;
  var model;
run;
```

The corresponding SQL code looks like:

```
proc sql;
  select model
    from sashelp.cars
    ;
quit;
```

The output from both the PROC PRINT and PROC SQL would look like:

```
Model
----------------------------------------
 MDX
 RSX Type S 2dr
 TSX 4dr
 TL 4dr
 3.5 RL 4dr
 3.5 RL w/Navigation 4dr
```

Takeaway Message: The keywords are different in SQL: DATA= is replaced by FROM and VAR is replaced by SELECT.

COMMAS

Here is another straightforward PROC PRINT example, in which several variables are printed at once.

```
proc print data=sashelp.cars;
  var model msrp mpg_highway;
run;
```

The corresponding SQL code utilizes commas and looks like:

```
proc sql;
  select model, msrp, mpg_highway
    from sashelp.cars
    ;
quit;
```

Model	MSRP	MPG (Highway)
Matrix XR	$16,695	36
Touareg V6	$35,515	20
Golf GLS 4dr	$18,715	31
GTI 1.8T 2dr hatch	$19,825	31
Jetta GLS TDI 4dr	$21,055	46
New Beetle GLS 1.8T 2dr	$21,055	31

Takeaway Message: Use commas to delimit a list of variables in SQL.

WHERE

Subsetting in SQL is actually very similar to typical SAS syntax. The keyword is WHERE.

```
proc sql;
  select model, msrp, mpg_highway
    from sashelp.cars
    where make = 'Cadillac'
    ;
quit;
```

ORDER BY

In SQL you are able to sort while you print. The keyword is ORDER BY.

```
proc sql;
  select model, msrp, mpg_highway
    from sashelp.cars
    where make = 'Cadillac'
    order by msrp
    ;
quit;
```

Model	MSRP	MPG (Highway)
CTS VVT 4dr	$30,835	25
Deville 4dr	$45,445	26
SRX V8	$46,995	21
Seville SLS 4dr	$47,955	26
Deville DTS 4dr	$50,595	26

Vs. Traditional Methods: Similar output might be generated by a PROC SORT followed by a PROC PRINT. The main benefit of the SQL is less typing.

STATISTICS

In the DATA step, functions return one result per observation (e.g., `x=max(var1,var2,var3);`). In SQL, certain functions are enhanced to have the added ability to perform calculations across records. Functions capable of producing across-record calculations include MAX, MEAN, and COUNT.

The following application of the MAX function returns the MSRP of the most expensive automobile in the dataset.

```
proc sql;
  select max(msrp)
    from sashelp.cars
    ;
quit;
```

```
--------
  192465
```

The following application of the MEAN function returns the average MSRP of all automobiles in the dataset.

```
proc sql;
  select mean(msrp)
    from sashelp.cars
    ;
quit;
```

```
--------
32774.86
```

The following application of the COUNT function returns the number of automobiles in the dataset.

```
proc sql;
  select count(*)
    from sashelp.cars
    ;
quit;
```

```
--------
     428
```

Potential Gotcha: With COUNT, you must use an asterisk within the parentheses rather than a variable name. Explanation of this particular syntactic quirk is beyond the scope of a quick and gentle paper.

Takeaway Message: SQL is much more than just an alternative to PROC PRINT. It also contains some of the functionality of PROC FREQ and PROC MEANS.

GROUP BY

The above statistics can be calculated separately within subgroups. The keyword is GROUP BY.

The following application of the MEAN function returns the average MSRP of the models produced by each manufacturer.

```
proc sql;
  select make, mean(msrp)
    from sashelp.cars
    group by make
    ;
quit;
```

```
Make
-----------------------
Acura          42938.57
Audi           43307.89
BMW            43285.25
Buick          30537.78
Cadillac       50474.38
```

Potential Gotcha: You have to specify the subgroup variable twice: once in the SELECT clause and a second time in the GROUP BY clause.

The following application of the COUNT function returns the number of models produced by each manufacturer.

```
proc sql;
  select make, count(*)
    from sashelp.cars
    group by make
    ;
quit;
```

```
Make
-----------------------
Acura                   7
Audi                   19
BMW                    20
Buick                   9
Cadillac                8
```

Takeaway Message: SQL is more than just a different way to print.

FUNDAMENTALS CONSOLIDATED AND ELABORATED UPON

Thus far we have introduced the following SQL keywords:

- SELECT (variables)
- FROM (input datasets)
- WHERE (subsetting)
- GROUP BY (subgroups)
- ORDER BY (sorting)

Not all keywords are required in a given SELECT statement, but keywords that are used must be specified in the order shown above. A useful pneumonic for remembering this order is "**S**o **f**ew **w**orkers **g**o home **o**n time."

A SELECT statement is referred to as a "query". The parts of a SELECT statement are referred to as "clauses" (e.g., "the WHERE clause", "the ORDER BY clause", etc.). We combine clauses to form a statement or a query.

The statement PROC SQL; is used prior to a SELECT statement. The statement QUIT; is used following a SELECT statement. Note that it is a QUIT; statement and not a RUN; statement.

```
proc sql;
  select …
    ;
quit;
```

Multiple SELECT statements are allowed within a single PROC SQL.

```
proc sql;
  select …
    ;
  select …
    ;
quit;
```

Various summary statistics can be calculated, including but not limited to:

- MAX(var)
- MEAN(var)
- COUNT(*)

Functions like MAX and MEAN can be used at the observation level by specifying multiple parameters (e.g., `x=max(var1,var2,var3);`), or they can be used at the dataset level by specifying a single parameter (e.g., `select max(var1)`). Summary statistics can be calculated within subgroups (i.e., by using GROUP BY) or at the dataset level (i.e., by omitting the GROUP BY).

Beauty being in the eye of the beholder, we meekly offer the following recommendations related to query readability: (a) keep each clause on a separate line, (b) indent keywords the same amount, and (c) put the semicolon on its own line.

```
proc sql;
  select …
    from …
    where …
    group by …
    order by …
    ;
quit;
```

BUILDING ON THE FUNDAMENTALS

The fundamentals in hand, we now build on them.

DISTINCT

The DISTINCT keyword allows us to see all of the unique values of a given variable.

```
proc sql;
  select distinct make
    from sashelp.cars
    ;
quit;
```

```
Make
--------------
Acura
Audi
BMW
Buick
Cadillac
```

Application: Use this technique when you want to see all unique values for a variable, but you aren't concerned with counting the number of observations corresponding to each unique value.

Vs. Traditional Methods: Similar output might be generated by a NODUPKEY SORT followed by a PROC PRINT. The main benefit of the SQL is less typing. PROC FREQ could also be used, but it carries a little additional overhead because it will be calculating counts as well.

SQLOBS

The automatic macro variable SQLOBS is updated after every SQL query. This macro variable indicates the number of records returned by the most recent query.

```
proc sql;
  select distinct make
    from sashelp.cars
    ;
quit;
%PUT &SQLOBS.;
```

```
Log:
107  %PUT &SQLOBS.;
38
```

Vs. Traditional Methods: Similar output might be generated by a NODUPKEY SORT followed by a CALL SYMPUT on the END= record within a DATA _NULL_. The main benefit of the SQL is again less typing.

Derivations

Derivations can be performed in the SELECT clause.

```
proc sql;
  select make, model, msrp*0.85
    from sashelp.cars
    ;
quit;
```

Make	Model	
Volvo	C70 HPT convertible 2dr	**36180.25**
Volvo	S80 T6 4dr	**38428.5**
Volvo	V40	**22214.75**
Volvo	XC70	**29873.25**

Vs. Traditional Methods: Similar output might be generated by a DATA step followed by a PROC PRINT. Again with the less typing thing.

Naming Columns

You might have noticed by now that the results of derivations and summary functions do not automatically come endowed with column headers. The keyword AS is used to assign names to derived columns. Note that the new variable name comes after the derivation.

```
proc sql;
  select make, model, msrp*0.85 as wholesale
    from sashelp.cars
    ;
quit;
```

Make	Model	**wholesale**
Acura	MDX	31403.25
Acura	RSX Type S 2dr	20247
Acura	TSX 4dr	22941.5
Acura	TL 4dr	28215.75
Acura	3.5 RL 4dr	37191.75

Takeaway Message: Compared to the dataset syntax for assigning values to new variables, we replace = with AS, and we move the new variable name from the left side to the right side.

CREATE TABLE

Results from queries can be saved into datasets. The keyword is CREATE TABLE.

```
proc sql;
  create table means_by_make as
  select make, mean(msrp) as mean_msrp
    from sashelp.cars
    group by make
    ;
quit;
```

```
Log:
NOTE: Table WORK.MEANS_BY_MAKE created, with 38 rows and 2 columns.
```

VIEWTABLE: Work.Means_by_make

	Make	mean_msrp
1	Acura	42938.571429
2	Audi	43307.894737
3	BMW	43285.25
4	Buick	30537.777778
5	Cadillac	50474.375
6	Chevrolet	26587.037037
7	Chrysler	27252

Potential Gotcha: The keyword AS is required prior to SELECT.

Vs. Traditional Methods: Similar output might be generated by a PROC SORT (by MAKE) followed by a PROC MEANS and ODS OUTPUT. The main benefit of the SQL is that you do not have to pre-sort your dataset.

ASTERISK

In order to select all variables in a dataset, use an asterisk.

```
proc sql;
  create table caddies as
  select *
    from sashelp.cars
    where make = 'Cadillac'
    ;
quit;
```

```
Log:
NOTE: Table WORK.CADDIES created, with 8 rows and 15 columns.
```

Vs. Traditional Methods: Similar output might be generated by a DATA step. It's a tie.

INTO

Results from queries can be saved into macro variables. The keyword is INTO. The following application returns the number of models for the make "Cadillac".

```
proc sql;
  select count(*)
    into :NMODELS
    from sashelp.cars
    where make = 'Cadillac'
    ;
quit;
%PUT NMODELS = [&NMODELS.];
```

```
Log:
121  %PUT NMODELS = [&NMODELS.];
NMODELS = [       8]
```

Potential Gotcha: The macro variable name must be preceded by a colon.

Potential Gotcha: SAS will sometimes pad macro variables with leading or trailing spaces. When sending results to the log with a %PUT statement, enclose the macro variable reference in brackets so that padding is more easily detected.

Vs. Traditional Methods: Similar output might be generated by a DATA step with WHERE= and END= on the SET statement and a CALL SYMPUT. The main benefit of the SQL is readability.

Sets of Macro Variables

INTO allows for the creation of a set of macro variables with a dash.

```
proc sql;
  select distinct make
    into :MAKE1 - :MAKE1000
    from sashelp.cars
    ;
  %
quit;
%PUT MAKE1 = [&MAKE1.];
%PUT MAKE2 = [&MAKE2.];
…
```

```
Log:
180  %PUT MAKE1 = [&MAKE1.];
MAKE1 = [Acura]
181  %PUT MAKE2 = [&MAKE2.];
MAKE2 = [Audi]
```

Potential Gotcha: The fact that we have to write down a specific number after the dash (i.e., :MAKE1000) is very unsatisfying. Alas, a number must be specified. As a safety precaution, always specify a number that is a couple of orders of magnitude larger than what you might reasonably expect to find in your data.

NOPRINT

Results from queries can be suppressed from the output window. The keyword is NOPRINT.

```
proc sql noprint;
  select count(*)
    into :NMODELS
    from sashelp.cars
    ;
quit;
```

Application: Use this technique when you want to know how many of something you have, but you don't necessarily need to see the count in the output window.

MORE CONSOLIDATION AND ELABORATION

In BUILDING ON THE FUNDAMENTALS we introduced the following:

- DISTINCT (removing duplicates)
- SQLOBS (number of observations returned by the query)
- Derivations (in SELECT)
- Naming Columns (with AS)
- CREATE TABLE (creating datasets)
- ASTERISK (selecting all variables in a dataset)
- INTO (creating one macro variable)
- Sets of Macro Variables (creating a set of macro variables)
- NOPRINT (suppressing results in the output window)

CREATE TABLE does not send results to the output window, but to a dataset. Therefore NOPRINT is unnecessary when CREATE TABLE is used.

Multiple variables can be specified following DISTINCT, GROUP BY, or ORDER BY.

```
proc sql;
  select make, model, msrp
    from sashelp.cars
    order by make, model
    ;
quit;
```

Because the value of the SQLOBS macro variable is automatically updated after every query, you should immediately save the result, lest the desired value be overwritten by your next query.

```
proc sql;
  select …
    ;
  %LET N1 = &SQLOBS.;
  select …
    ;
  %LET N2 = &SQLOBS.;
quit;
```

CONCLUSION

PROC SQL is an extremely powerful procedure that combines parts and pieces of the DATA step, PROC PRINT, PROC SORT, PROC FREQ, PROC MEANS, and the Macro Language. If a SAS user is not properly introduced to PROC SQL it can easily intimidate. We hope that the preceding introduction was quick enough so as not to bore you and gentle enough so as not to scare you. We leave you with the following summarization of all of the topics that have been introduced in this paper.

```
proc sql <noprint>;
  create table table-name as
  select <*> <distinct> <var <as newvar>> <,…var <as newvar>>
    into :mac
    from table-name
    where expression
    group by var <,…var>
    order by var <,…var>
    ;
  %LET N = &SQLOBS.;
quit;
```

REFERENCES

SAS® Institute Inc., 2009. SAS® 9.2 SQL Procedure User's Guide. Cary, NC: SAS® Institute Inc.

RECOMMENDED READING

Gusinow, Rosalind, "Introduction to PROC SQL", Proceedings of the 25th Annual SAS® Users Group International Conference

DeFoor, Jimmy, "Proc SQL – A Primer for SAS® Programmers", Proceedings of the 31st Annual SAS® Users Group International Conference

CONTACT INFORMATION

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

Shane Rosanbalm
Rho, Inc
6330 Quadrangle Dr., Ste. 500
Chapel Hill, NC 27517
919.595.6273
E-mail: shane_rosanbalm@rhoworld.com
Web: www.rhoworld.com