Paper 268-29

Introduction to Proc SQL

Katie Minten Ronk, Systems Seminar Consultants, Madison, WI

ABSTRACT

PROC SQL is a powerful Base SAS® Procedure that combines the functionality of DATA and PROC steps into a single step. PROC SQL can sort, summarize, subset, join (merge), and concatenate datasets, create new variables, and print the results or create a new table or view all in one step!

PROC SQL can be used to retrieve, update, and report on information from SAS data sets or other database products. This paper will concentrate on SQL's syntax and how to access information from existing SAS data sets. Some of the topics covered in this brief introduction include:

- Writing SQL code using various styles of the SELECT statement.
- Dynamically creating new variables on the SELECT statement.
- Using CASE/WHEN clauses for conditionally processing the data.
- Joining data from two or more data sets (like a MERGE!).
- Concatenating query results together.

WHY LEARN PROC SQL?

PROC SQL can not only retrieve information without having to learn SAS syntax, but it can often do this with fewer and shorter statements than traditional SAS code. Additionally, SQL often uses fewer resources than conventional DATA and PROC steps. Further, the knowledge learned is transferable to other SQL packages.

AN EXAMPLE OF PROC SQL SYNTAX

Every PROC SQL query must have at least one SELECT statement. The purpose of the SELECT statement is to name the columns that will appear on the report and the order in which they will appear (similar to a VAR statement on PROC PRINT). The FROM clause names the data set from which the information will be extracted from (similar to the SET statement). One advantage of SQL is that new variables can be dynamically created on the SELECT statement, which is a feature we do not normally associate with a SAS Procedure:

```
PROC SQL;
  SELECT STATE, SALES,
    (SALES * .05) AS TAX
  FROM USSALES;
QUIT;
    (no output shown for this code)
```

THE SELECT STATEMENT SYNTAX

The purpose of the SELECT statement is to describe how the report will look. It consists of the SELECT clause and several sub-clauses. The sub-clauses name the input dataset, select rows meeting certain conditions (subsetting), group (or aggregate) the data, and order (or sort) the data:

```
PROC SQL options;
  SELECT column(s)
   FROM table-name | view-name
   WHERE expression
   GROUP BY column(s)
   HAVING expression
   ORDER BY column(s);
QUIT;
```

A SIMPLE PROC SQL

An asterisk on the SELECT statement will select all columns from the data set. By default a row will wrap when there is too much information to fit across the page. Column headings will be separated from the data with a line and no observation number will appear:

```
PROC SQL;
```

```
    SELECT *
    FROM USSALES;
   QUIT;
      (see output #1 for results)
```

A COMPLEX PROC SQL

The SELECT statement in it's simplest form, needs a SELECT and a FROM clause.   The SELECT statement can also have all six possible clauses represented in a query:

```
  proc sql;
    SELECT state, sum(sales) as TOTSALES
    FROM ussales
    WHERE state in ('WI','MI','IL')
    GROUP BY state
    HAVING  sum(sales) > 40000
    ORDER BY state desc;
quit;
      (see output #2 for results)
```

These statements will be reviewed in detail later in the paper.

LIMITING INFORMATION ON THE SELECT

To specify that only certain variables should appear on the report, the variables are listed and separated on the SELECT statement. The SELECT statement does NOT limit the number of variables read. The NUMBER option will print a column on the report labeled 'ROW' which contains the observation number:

```
  PROC SQL NUMBER;
    SELECT STATE, SALES
    FROM USSALES;
  QUIT;
      (see output #3 for results)
```

CREATING NEW VARIABLES

Variables can be dynamically created in PROC SQL. Dynamically created variables can be given a variable name, label, or neither. If a dynamically created variable is not given a name or a label, it will appear on the report as a column with no column heading. Any of the DATA step functions can be used in an expression to create a new variable except LAG, DIF, and SOUND. Notice the commas separating the columns:

```
  PROC SQL;
    SELECT SUBSTR(STORENO,1,3) LABEL='REGION',
      SALES, (SALES * .05) AS TAX,
      (SALES * .05) * .01
    FROM USSALES;
  QUIT;
      (see output #4 for results)
```

THE CALCULATED OPTION ON THE SELECT

Starting with Version 6.07, the CALCULATED component refers to a previously calculated variable so recalculation is not necessary. The CALCULATED component must refer to a variable created within the same SELECT statement:

```
  PROC SQL;
    SELECT STATE, (SALES * .05) AS TAX,
      (SALES * .05) * .01 AS REBATE
    FROM USSALES;
  - or -
    SELECT STATE, (SALES * .05) AS TAX,
      CALCULATED TAX * .01 AS REBATE
    FROM USSALES;
```

```
   QUIT;
       (see output #5 for results)
```

USING LABELS AND FORMATS

SAS-defined or user-defined formats can be used to improve the appearance of the body of a report. LABELs give the ability to define longer column headings:

```
   TITLE 'REPORT OF THE U.S. SALES';
   FOOTNOTE 'PREPARED BY THE MARKETING DEPT.';
   PROC SQL;
     SELECT STATE, SALES
         FORMAT=DOLLAR10.2
         LABEL='AMOUNT OF SALES',
       (SALES * .05) AS TAX
         FORMAT=DOLLAR7.2
         LABEL='5% TAX'
     FROM USSALES;
   QUIT;
       (see output #6 for results)
```

THE CASE EXPRESSION ON THE SELECT

The CASE Expression allows conditional processing within PROC SQL:

```
   PROC SQL;
     SELECT STATE,
       CASE
         WHEN SALES BETWEEN 0 AND 10000 THEN 'LOW'
         WHEN SALES BETWEEN 10001 AND 15000 THEN 'AVG'
         WHEN SALES BETWEEN 15001 AND 20000 THEN 'HIGH'
         ELSE 'VERY HIGH'
       END AS SALESCAT
     FROM USSALES;
   QUIT;
       (see results #7 for results)
```

The END is required when using the CASE. Coding the WHEN in descending order of probability will improve efficiency because SAS will stop checking the CASE conditions as soon as it finds the first true value. Also note that the length of SALESCAT will be the longest value (the length of VERY HIGH or nine characters). No special length statement is required as it is in the data step.

Another interesting thing about CASE-WHEN logic is that the same operators that are available on the WHERE statement, are also available in CASE-WHEN logic. These operators are:

- All operators that IF uses (= , <, >, NOT, NE, AND, OR, IN, etc)
- BETWEEN AND
- CONTAINS or '?'
- IS NULL or IS MISSING
- = *
- LIKE

ANOTHER CASE

The CASE statement has much of the same functionality as an IF statement. Here is yet another variation on the CASE expression:

```
   PROC SQL;
     SELECT STATE,
       CASE
         WHEN SALES > 20000 AND STORENO
           IN ('33281','31983') THEN 'CHECKIT'
         ELSE 'OKAY'
```

```
      END AS SALESCAT
    FROM USSALES;
  QUIT;
      (see output #8 for results)
```

ADDITIONAL SELECT STATEMENT CLAUSES

The GROUP BY clause can be used to summarize or aggregate data. Summary functions (also referred to as aggregate functions) are used on the SELECT statement for each of the analysis variables:

```
  PROC SQL;
    SELECT STATE, SUM(SALES) AS TOTSALES
    FROM USSALES
    GROUP BY STATE;
  QUIT;
      (see output #9 for results)
```

Other summary functions available are the AVG/MEAN, COUNT/FREQ/N, MAX, MIN, NMISS, STD, SUM, and VAR. This capability Is similar to PROC SUMMARY with a CLASS statement.

REMERGING

Remerging occurs when a summary function is used without a GROUP BY. The result is a grand total shown on every line:

```
  PROC SQL;
    SELECT STATE, SUM(SALES) AS TOTSALES
    FROM USSALES;
  QUIT;
      (see output #10 for results)
```

REMERGING FOR TOTALS

Sometimes remerging is good, as in the case when the SELECT statement does not contain any other variables:

```
  PROC SQL;
    SELECT SUM(SALES) AS TOTSALES
    FROM USSALES;
  QUIT;
      (see output #11 for results)
```

CALCULATING PERCENTAGE

Remerging can also be used to calculate percentages:

```
  PROC SQL;
    SELECT STATE, SALES,
      (SALES/SUM(SALES)) AS PCTSALES
            FORMAT=PERCENT7.2
    FROM USSALES;
  QUIT;
      (see output #12 for results)
```

Check your output carefully when the remerging note appears in your log to determine if the results are what you expect.

SORTING THE DATA IN PROC SQL

The ORDER BY clause will return the data in sorted order: Much like PROC SORT, if the data is already in sorted order, PROC SQL will print a message in the LOG stating the sorting utility was not used. When sorting on an existing column, PROC SQL and PROC SORT are nearly comparable in terms of efficiency. SQL may be more efficient when you need to sort on a dynamically created variable:

```
  PROC SQL;
    SELECT STATE, SALES
    FROM USSALES
    ORDER BY STATE, SALES DESC;
  QUIT;
      (see output #13 for results)
```

SORT ON NEW COLUMN

On the ORDER BY or GROUP BY clauses, columns can be referred to by their name or by their position on the SELECT cause. The option 'ASC' (ascending) on the ORDER BY clause is the default, it does not need to be specified.

```
  PROC SQL;
    SELECT SUBSTR(STORENO,1,3)
      LABEL='REGION',
      (SALES * .05) AS TAX
    FROM USSALES
    ORDER BY 1 ASC, TAX DESC;
  QUIT;
      (see output #14 for results)
```

SUBSETTING USING THE WHERE

The WHERE statement will process a subset of data rows before they are processed:

```
  PROC SQL;
    SELECT *
    FROM USSALES
    WHERE STATE IN ('OH','IN','IL');

    SELECT *
    FROM USSALES
    WHERE NSTATE IN (10,20,30);

    SELECT *
    FROM USSALES
    WHERE STATE IN ('OH','IN','IL') AND SALES > 500;
  QUIT;
      (no output shown for this example)
```

INCORRECT WHERE CLAUSE

Be careful of the WHERE clause, it cannot reference a computed variable:

```
  PROC SQL;
    SELECT STATE, SALES,
      (SALES * .05) AS TAX
    FROM USSALES
    WHERE STATE IN ('OH','IN','IL') AND TAX > 10 ;
  QUIT;
      (see output #15 for results)
```

WHERE ON COMPUTED COLUMN

To use computed variables on the WHERE clause they must be recomputed:

```
  PROC SQL;
    SELECT STATE, SALES,
      (SALES * .05) AS TAX
    FROM USSALES
    WHERE STATE IN ('OH','IL','IN')
      AND (SALES * .05) > 10;
  QUIT;
      (see output #16 for results)
```

SELECTION ON GROUP COLUMN

The WHERE clause cannot be used with the GROUP BY:

```
   PROC SQL;
     SELECT STATE, STORE,
       SUM(SALES) AS TOTSALES
     FROM USSALES
     GROUP BY STATE, STORE
     WHERE TOTSALES > 500;
   QUIT;
       (see output #17 for results)
```

USE HAVING CLAUSE

In order to subset data when grouping is in effect, the HAVING clause must be used:

```
   PROC SQL;
     SELECT STATE, STORENO,
       SUM(SALES) AS TOTSALES
     FROM USSALES
     GROUP BY STATE, STORENO
     HAVING SUM(SALES) > 500;
   QUIT;
       (see output #18 for results)
```

CREATING NEW TABLES OR VIEWS

The CREATE statement provides the ability to create a new data set as output in lieu of a report (which is what happens when a SELECT is present without a CREATE statement). The CREATE statement can either build a TABLE (a traditional SAS dataset, like what is built on a SAS DATA statement) or a VIEW (not covered in this paper):

```
   PROC SQL;
     CREATE TABLE TESTA AS
     SELECT STATE, SALES
     FROM USSALES
     WHERE STATE IN ('IL','OH');

     SELECT * FROM TESTA;
   QUIT;
       (see output #19 for results)
```

The name given on the create statement can either be temporary or permanent. Only one table or view can be created by a CREATE statement. The second SELECT statement (without a CREATE) is used to generate the report.

JOINING DATASETS USING PROC SQL

A join is used to combine information from multiple files. One advantage of using PROC SQL to join files is that it does not require sorting the datasets prior to joining as is required with a DATA step merge.

A Cartesian Join combines all rows from one file with all rows from another file. This type of join is difficult to perform using traditional SAS code.

```
   PROC SQL;
     SELECT *
     FROM GIRLS, BOYS;
   QUIT;
       (see output #20 for results)
```

INNER JOIN

A Conventional or Inner Join combines datasets only if an observation is in both datasets. This type of join is similar to a DATA step merge using the IN Data Set Option and IF logic requiring that the observation's key is on both data sets (IF ONA AND ONB).

```
PROC SQL;
```

```
   SELECT *
   FROM GIRLS, BOYS
   WHERE GIRLS.STATE=BOYS.STATE;
QUIT;
       (see output #21 for results)
```

JOINING THREE OR MORE TABLES

An Associative Join combines information from three or more tables. Performing this operation using traditional SAS code would require several PROC SORTs and several DATA step merges. The same result can be achieved with one PROC SQL:

```
PROC SQL;
  SELECT B.FNAME, B.LNAME, CLAIMS,
       E.STORENO, STATE
  FROM BENEFITS B, EMPLOYEE E,
       FEBSALES F
  WHERE B.FNAME=E.FNAME AND
        B.LNAME=E.LNAME AND
        E.STORENO=F.STORENO AND
           CLAIMS >  1000;
QUIT;
    (see output #22 for dataset list and results)
```

CONCATENATING QUERY RESULTS

Query results can be concatenated with the UNION operator.
The UNION operator keeps only unique observations. To keep all observations, the UNION ALL operator can be used. Traditional SAS syntax would require the creation of multiple tables and then either a SET concatenation or a PROC APPEND. Again, the results can be achieved with one PROC SQL:

```
PROC SQL;
  CREATE TABLE YTDSALES AS
  SELECT TRANCODE, STORENO, SALES
  FROM JANSALES

  UNION
  SELECT TRANCODE, STORENO,
         SALES * .99
  FROM FEBSALES;
QUIT;

(no output shown for this example)
```

IN SUMMARY

PROC SQL is a powerful data analysis tool. It can perform many of the same operations as found in traditional SAS code, but can often be more efficient because of its dense language structure.

PROC SQL can be an effective tool for joining data, particularly when doing associative, or three-way joins. For more information regarding SQL joins, reference the papers noted in the bibliography.

CONTACT INFORMATION

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

Katie Minten Ronk
Systems Seminar Consultants
2997 Yarmouth Greenway Drive
Madison, WI 53713
Phone: (608) 278-9964
Fax: (608) 278-0065

Email: kronk@sys-seminar.com
Web: www.sys-seminar.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or 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.

OUTPUT #1 (PARTIAL):

```
STATE      SALES   STORENO
COMMENT
STORENAM
--------------------------------------------------
WI      10103.23   32331
SALES WERE SLOW BECAUSE OF COMPETITORS SALE
RON'S VALUE RITE STORE

WI       9103.23   32320
SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER
PRICED SMART GROCERS

WI      15032.11   32311
AVERAGE SALES ACTIVITY REPORTED
VALUE CITY
```

OUTPUT #2 (PARTIAL):

```
STATE   TOTSALES
________________
MI      53341.66
IL      84976.57
```

OUTPUT #3 (PARTIAL):

```
ROW   STATE      SALES
 ---------------------
  1   WI      10103.23
  2   WI       9103.23
  3   WI      15032.11
```

OUTPUT #4 (PARTIAL):

```
REGION      SALES        TAX
------------------------------------
323      10103.23   505.1615   5.051615
323       9103.23   455.1615   4.551615
323      15032.11   751.6055   7.516055
332      33209.23   1660.462   16.60461
```

OUTPUT #5 (PARTIAL):

```
STATE        TAX    REBATE
-------------------------
WI      505.1615   5.051615
WI      455.1615   4.551615
WI      751.6055   7.516055
MI      1660.462   16.60461
```

OUTPUT #6 (PARTIAL):

```
         REPORT OF THE U.S. SALES

                AMOUNT OF
         STATE        SALES   5% TAX
         -------------------------
         WI      $10,103.23  $505.16
         WI       $9,103.23  $455.16
         WI      $15,032.11  $751.61
         MI      $33,209.23  1660.46


       PREPARED BY THE MARKETING DEPT.
```

OUTPUT #7 (PARTIAL):

```
 STATE  SALESCAT
 ----------------
 WI      AVG
 WI      LOW
 WI      HIGH
 MI      VERY HIGH
```

OUTPUT #8 (PARTIAL):

```
STATE  SALESCAT
---------------
WI      OKAY
WI      OKAY
WI      OKAY
MI      CHECKIT
```

OUTPUT #9:

```
 STATE  TOTSALES
 ---------------
 IL      84976.57
 MI      53341.66
 WI      34238.57
```

OUTPUT #10 (PARTIAL):

```
STATE  TOTSALES
---------------
WI      172556.8
WI      172556.8
WI      172556.8
MI      172556.8
```

OUTPUT #11:

```
TOTSALES
--------
172556.8
```

OUTPUT #12 (PARTIAL):
(log message shown)

```
          STATE      SALES   PCTSALES

          ________________________
          WI      10103.23     5.86%
          WI       9103.23     5.28%
          WI      15032.11     8.71%
          MI      33209.23     19.2%
NOTE: The query requires remerging summary
      Statistics back with the original data.
```

OUTPUT #13 (PARTIAL):

```
          STATE      SALES
          ---------------
          IL      32083.22
          IL      22223.12
          IL      20338.12
          IL      10332.11
          MI      33209.23
```

OUTPUT #14 (PARTIAL):

```
          REGION        TAX
          ----------------
          312      516.6055
          313      1604.161
          313      1111.156
          319      1016.906
```

OUTPUT #15 (THE RESULTING SAS LOG- PARTIAL):

```
27          PROC SQL;
28            SELECT STATE,SALES, (SALES * .05) AS TAX
29            FROM USSALES
30            WHERE STATE IN ('OH','IN','IL') AND TAX > 10;
ERROR: THE FOLLOWING COLUMNS WERE NOT FOUND IN THE
       CONTRIBUTING TABLES: TAX.
NOTE:  The SAS System stopped processing this step because
 of errors.
```

OUTPUT #16 (PARTIAL):

```
STATE      SALES        TAX
-------------------------
WI      10103.23   505.1615
WI       9103.23   455.1615
WI      15032.11   751.6055
IL      20338.12   1016.906
```

OUTPUT #17 (THE RESULTING SAS LOG- PARTIAL):

```
167    GROUP BY STATE, STORE
168    WHERE TOTSALES > 500;
-----
22
202
ERROR 22-322: Expecting one of the following: (, **, *, /, +, -
               !!, ||, <, <=, <>, =, >, >=, EQ, GE, GT, LE, LT,
               NE, ^=, ~=, &, AND, !, OR, |, ',', HAVING, ORDER
               The statement is being ignored.

ERROR 202-322: The option or parameter is not recognized.
```

OUTPUT #18 (PARTIAL):

```
STATE  STORENO  TOTSALES
------------------------
IL     31212    10332.11
IL     31373    22223.12
IL     31381    32083.22
IL     31983    20338.12
MI     33281    33209.23
```

OUTPUT #19:

```
STATE     SALES
---------------
IL     20338.12
IL     10332.11
IL     32083.22
IL     22223.12
```

OUTPUT #20(PARTIAL):

```
NAME          STATE  NAME          STATE
----------------------------------------
NANCY            WI  NED              WI
NANCY            WI  GENE             NY
NANCY            WI  ADAM             CA
JEAN             MN  NED              WI
JEAN             MN  GENE             NY
JEAN             MN  ADAM             CA
AMELIA           IL  NED              WI
AMELIA           IL  GENE             NY
AMELIA           IL  ADAM             CA
```

OUTPUT #21 (PARTIAL):

```
NAME      STATE     NAME      STATE
-----------------------------------
NANCY     WI        NED       WI
```

OUTPUT #22:

```
EMPLOYEE                                  FEBSALES                                 BENEFITS
OBS    FNAME     LNAME      STORENO       OBS    STATE      SALES     STORENO      OBS    FNAME     LNAME      CLAIMS

1      ANN       BECKER     33281         1      MI      31209.23     33281        1      ANN       BECKER       2003
2      CHRIS     DOBSON     33281         2      MI      15132.43     33312        2      CHRIS     DOBSON        100
3      EARL      FISHER     33281         3      IL      25338.12     31983        3      ALLEN     PARK        10392
4      ALLEN     PARK       31373         4      IL      26223.12     31373        4      BETTY     JOHNSON      3832
5      BETTY     JOHNSON    31373
6      KAREN     ADAMS      31373
```

```
FNAME     LNAME           CLAIMS  STORENO  STATE
---------------------------------------------
ANN       BECKER            2003  33281     MI
ALLEN     PARK             10392  31373     IL
BETTY     JOHNSON           3832  31373     IL
```