An Introduction to PROC SQL
Kim L. Kolbe Ritzow, Systems Seminar Consultants, Kalamazoo, MI

Abstract
This paper will discuss the basic syntax of PROC SQL and how it can be used to create some simple reports. It will also discuss the CREATE statement, the HAVING clause, the CASE statement and the various types of CASE expressions that can be used.

This paper will not address how PROC SQL can be used to retrieve information from other database management systems. It will deal solely with the types of reports that can be generated in SQL when retrieving information from SAS data sets.

Background
PROC SQL is a new Base SAS® PROC starting with Version 6.06 of SAS Software®. It has both DATA and PROC step functionalities which can be used to retrieve, update, and report on information in SAS data files (be it traditional SAS data sets or views). Its syntax is similar to the ANSI standard for SQL with a few key differences. It is in some cases more efficient than traditional SAS code because of its dense language structure.

Why Learn PROC SQL?
PROC SQL can not only retrieve information without having to learn SAS syntax, but it can also oftentimes do this with fewer and shorter statements than traditional SAS code. Additionally, on average it uses fewer resources than conventional DATA and PROC steps. This means PROC SQL is usually a more efficient alternative to traditional SAS code. Further, the knowledge learned is transferrable.

An Example of PROC SQL's Syntax
Every PROC SQL 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 come from (similar to the SET statement). One advantages 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 USALES;
QUIT;
```

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

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

A Simple PROC SQL
An "*" on the SELECT statement will select all columns. By default a row will wrap when there is too much information to fit across the page. Also by default, column headings will be separated from the data with a line and no observation number will appear:

```
PROC SQL;
  SELECT *
  FROM USALES;
QUIT;
```

(see output #1 for results)

Limiting Information on the SELECT
Multiple requests are delimited by commas 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
QUIT;
```
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 associated with it. Any of the DATA step functions can be used in an expression to create a new variable except LAG, DIF, and SOUND:

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

Outout #2

Options on the PROC SQL Statement
There are several useful options that can be used on the PROC SQL statement to help control the appearance of the report. Be careful, once coded, these options will apply to all SELECT statements within PROC SQL unless a RESET statement is used:

```
PROC SQL INOBS= 100 OUTOBS=9 DOUBLE;
SELECT STORE, (SALES * .05) AS TAX
FROM USSALES;
QUIT;
```

Outout #3

The FLOW Option and Using RESET
The FLOW option allows text to continue in its column rather than wrapping the text on to the next line. If a value is not specified on the FLOW option, SAS will "flow" the value to the length of the column. The RESET statement changes options within the same step without respecifying the procedure. The option FLOW=30 40 floats the width of the column between the values specified to produce a better layout:

```
PROC SQL;
SELECT STATE, STORENAM, COMMENT FROM USSALES;
RESET FLOW=30 40 DOUBLE NUMBER
```

Outout #4

```
PROC SQL;
SELECT STATE, STORENAM, COMMENT FROM USSALES;
QUIT;  
```

Outout #5

Note: multiple SELECT clauses can be coded under a single PROC SQL. Each SELECT clause will generate a separate report.

The CALCULATED Option on the SELECT Statement
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 in the same SELECT statement as it is used:

```
PROC SQL INOBS=9;
SELECT STATE, (SALES * .05) AS TAX,
   (SALES * .05) * .01 AS REBATE
FROM USSALES;
QUIT;
```

Outout #6

```
PROC SQL;
SELECT STATE, SALES FORMAT=DOLLAR10.2
   LABEL='AMOUNT OF SALES',
   (SALES * .05) AS TAX
FROM USSALES;
QUIT;
```

Outout #7

Options on the PROC SQL Statement
There are several useful options that can be used on the PROC SQL statement to help control the appearance of the report. Be careful, once coded, these options will apply to all SELECT statements within PROC SQL unless a RESET statement is used:

```
PROC SQL INOBS= 100 OUTOBS=9 DOUBLE;
SELECT STORE, (SALES * .05) AS TAX
FROM USSALES;
QUIT;
```

Outout #4

```
PROC SQL;
SELECT STATE, STORENAM, COMMENT FROM USSALES;
RESET FLOW=30 40 DOUBLE NUMBER
```

Outout #5

Note: multiple SELECT clauses can be coded under a single PROC SQL. Each SELECT clause will generate a separate report.

The CALCULATED Option on the SELECT Statement
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 in the same SELECT statement as it is used:

```
PROC SQL INOBS=9;
SELECT STATE, (SALES * .05) AS TAX,
   (SALES * .05) * .01 AS REBATE
FROM USSALES;
QUIT;
```

Outout #6

```
PROC SQL;
SELECT STATE, SALES FORMAT=DOLLAR10.2
   LABEL='AMOUNT OF SALES',
   (SALES * .05) AS TAX
FROM USSALES;
QUIT;
```

Outout #7
The CASE Expression on the SELECT
The CASE Expression allows conditional processing within PROC SQL:

PROC SQL;
SELECT STATE,
CASE
  WHEN SALES < 10000 THEN 'LOW'
  WHEN SALES < 15000 THEN 'AVG'
  WHEN SALES < 20000 THEN 'HIGH'
  ELSE 'VERY HIGH'
END AS SALES_CAT
FROM USSALES;
QUIT;
(see results #8 for results)

The END is required when using the CASE. Coding the WHEN in descending order of probability will improve efficiency because it will stop checking when it finds the first value to be true. You do not have to worry about the length of the newly created variable’s value being assigned with WHENs like you do with the IF.

The CASE Expression can be coded many different ways; perhaps this WHEN clause looks more familiar to you. It does the exact same thing as the previous syntax:

PROC SQL;
SELECT STATE,
CASE
  WHEN 0 <= SALES <= 10000 THEN 'LOW'
  WHEN 10001 <= SALES <= 15000 THEN 'AVG'
  WHEN 15001 <= SALES <= 20000 THEN 'HIGH'
  ELSE 'VERY HIGH'
END AS SALES_CAT
FROM USSALES;
QUIT;
(output is same as output #8)

Basically, you can do all the same things on a CASE statement as you can on an IF. 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 SALES_CAT
FROM USSALES;
QUIT;
(see output #9 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 #10 for results)

Other summary functions available are the AVG/MED, COUNT/FREQ/N, MAX, MIN, NMISS, STD, SUM, and VAR.

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

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

Sometimes remerging is good, as in the case when the SELECT statement does not contain any character variables, or in the case of calculating a percentage:

PROC SQL;
SELECT SUM(SALES) AS TOTSALES FROM USSALES;
QUIT;
(see output #12 for results)

PROC SQL;
SELECT STATE, SALES,
(SALES/SUM(SALES)) AS PCTSALES
FROM USSALES;
QUIT;
(output is same as output #8)
Always check your output carefully when the remerging note appears in your log to determine if you have gotten the desired results.

Sorting the Data in PROC SQL
The ORDER BY clause will return the data in sorted order:

PROC SQL;
SELECT STATE, SALES
FROM USSALES
ORDER BY STATE, SALES DESC;
QUIT;
(see output #14 for results)

Much like PROC SORT, if the data are 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 is more efficient when you need to sort on a dynamically created variable:

PROC SQL;
SELECT SUBSTR(STORENO,1,3)
  LABEL = 'REGION',
  (SALES * .05) AS TAX
FROM USSALES
ORDER BY 1 ASC, TAX DESC;
QUIT;
(see output #15 for results)

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

Subsetting Using the WHERE
The WHERE statement will subset rows before they are read:

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)

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 #16 for results)

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 #17 for results)

Also be aware that they WHERE statement cannot be used with the GROUP BY:

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

In order to subset data when grouping is in effect, the HAVING statement 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 #19 for results)

The HAVING clause is needed even if it is not referring to a computed variable:

PROC SQL;
SELECT STATE,
SUM(SALES) AS TOTSALES
FROM USSALES
GROUP BY STATE
HAVING STATE IN ('IL','WI');
QUIT;

(see output #20 for results)

The CREATE Statement
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 #21 for results)

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

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 oftentimes do it more efficiently because of its dense language structure.

While not discussed in this paper, 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.

Trademark Notice
SAS is a registered trademark of the SAS Institute Inc., Cary, NC, USA and other countries.

Useful Publications
SAS Institute Inc., Getting Started with the SQL Procedure, Version 6, First Edition


Kolbe Ritzow, Kim, "Joining Data with SQL", Proceedings of the 6th Annual MidWest SAS® Users Group Conference

Lafler, Kirk Paul, "Diving into SAS® Software with the SQL Procedure", Proceedings of the 20th Annual SAS® Users Group International Conference


Any questions or comments regarding the paper may be directed to the author:

Kim L. Kolbe Ritzow
Systems Seminar Consultants
Kalamazoo Office
927 Lakeway Drive
Kalamazoo, MI 49001
Phone: (616) 345-5636
Fax: (616) 345-5793
E-mail: KRITZOW@AOL.COM
(or) KRITZOW@MCIMAIL.COM
Output #1 (partial):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES STORENO</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI 10103.23</td>
<td>32331</td>
<td>SALES WERE SLOW BECAUSE OF COMPETITOR'S SALE</td>
</tr>
<tr>
<td>WI 9103.23</td>
<td>32320</td>
<td>SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER</td>
</tr>
<tr>
<td>WI 15032.11</td>
<td>32311</td>
<td>AVERAGE SALES ACTIVITY REPORTED</td>
</tr>
</tbody>
</table>

Output #2 (partial):

```
<table>
<thead>
<tr>
<th>ROW</th>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WI</td>
<td>10103.23</td>
</tr>
<tr>
<td>2</td>
<td>WI</td>
<td>9103.23</td>
</tr>
<tr>
<td>3</td>
<td>WI</td>
<td>15032.11</td>
</tr>
</tbody>
</table>
```

Output #3 (partial):

```
<table>
<thead>
<tr>
<th>REGION</th>
<th>SALES</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>323</td>
<td>10103.23</td>
<td>505.1615</td>
</tr>
<tr>
<td>323</td>
<td>9103.23</td>
<td>455.1615</td>
</tr>
<tr>
<td>323</td>
<td>15032.11</td>
<td>751.6055</td>
</tr>
<tr>
<td>332</td>
<td>33209.23</td>
<td>1660.462</td>
</tr>
</tbody>
</table>
```

Output #4 (partial):

```
<table>
<thead>
<tr>
<th>STATE</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>505.1615</td>
</tr>
<tr>
<td>WI</td>
<td>455.1615</td>
</tr>
<tr>
<td>WI</td>
<td>751.6055</td>
</tr>
<tr>
<td>MI</td>
<td>1660.462</td>
</tr>
</tbody>
</table>
```

Output #5 (partial):

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENAM</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>RON'S VALUE RITE STORE</td>
<td>SALES WERE SLOW BECAUSE OF COMPETITOR'S SALE</td>
</tr>
<tr>
<td>WI</td>
<td>PRICED SMART GROCERS</td>
<td>SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>ROW</th>
<th>STATE</th>
<th>STORENAM</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WI</td>
<td>RON'S VALUE RITE STORE</td>
<td>SALES WERE SLOW BECAUSE OF COMPETITOR'S SALE</td>
</tr>
<tr>
<td>2</td>
<td>WI</td>
<td>PRICED SMART GROCERS</td>
<td>SALES SLOWER THAN NORMAL BECAUSE OF BAD WEATHER</td>
</tr>
</tbody>
</table>
```
Output #6 (partial):

<table>
<thead>
<tr>
<th>STATE</th>
<th>TAX REBATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>505.1615</td>
</tr>
<tr>
<td>WI</td>
<td>455.1615</td>
</tr>
<tr>
<td>WI</td>
<td>751.6055</td>
</tr>
<tr>
<td>WI</td>
<td>1660.462</td>
</tr>
</tbody>
</table>

Output #7 (partial):

REPORT OF THE U.S. SALES

<table>
<thead>
<tr>
<th>AMOUNT OF STATE SALES</th>
<th>5% TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI $10,103.23</td>
<td>$505.16</td>
</tr>
<tr>
<td>WI $9,103.23</td>
<td>$455.16</td>
</tr>
<tr>
<td>WI $15,032.11</td>
<td>$751.61</td>
</tr>
<tr>
<td>HI $33,209.23</td>
<td>$1660.46</td>
</tr>
</tbody>
</table>

PREPARED BY THE MARKETING DEPT.

Output #8 (partial):

<table>
<thead>
<tr>
<th>STATE SALES</th>
<th>CAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>AVG</td>
</tr>
<tr>
<td>WI</td>
<td>LOW</td>
</tr>
<tr>
<td>WI</td>
<td>HIGH</td>
</tr>
<tr>
<td>MI</td>
<td>VERY HIGH</td>
</tr>
</tbody>
</table>

Output #9 (partial):

<table>
<thead>
<tr>
<th>STATE SALES</th>
<th>CAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>OKAY</td>
</tr>
<tr>
<td>WI</td>
<td>OKAY</td>
</tr>
<tr>
<td>WI</td>
<td>OKAY</td>
</tr>
<tr>
<td>MI</td>
<td>CHECKIT</td>
</tr>
</tbody>
</table>

Output #10:

<table>
<thead>
<tr>
<th>STATE TOTALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
</tr>
<tr>
<td>MI</td>
</tr>
<tr>
<td>WI</td>
</tr>
</tbody>
</table>

Output #11 (partial):

<table>
<thead>
<tr>
<th>STATE TOTALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
</tr>
<tr>
<td>WI</td>
</tr>
<tr>
<td>WI</td>
</tr>
<tr>
<td>MI</td>
</tr>
</tbody>
</table>

Output #12:

<table>
<thead>
<tr>
<th>TOTALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>172556.8</td>
</tr>
</tbody>
</table>
Beginning Tutorials

Output #13 (partial):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>PCTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>10103.23</td>
<td>5.86%</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>5.28%</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
<td>8.71%</td>
</tr>
<tr>
<td>MI</td>
<td>33209.23</td>
<td>19.2%</td>
</tr>
</tbody>
</table>

Output #14 (partial):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>10103.23</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
</tr>
<tr>
<td>MI</td>
<td>33209.23</td>
</tr>
</tbody>
</table>

Output #15 (partial):

<table>
<thead>
<tr>
<th>REGION</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>312</td>
<td>516.6055</td>
</tr>
<tr>
<td>313</td>
<td>1604.161</td>
</tr>
<tr>
<td>313</td>
<td>1111.156</td>
</tr>
<tr>
<td>319</td>
<td>1016.906</td>
</tr>
</tbody>
</table>

Output #16 (The resulting SAS LOG- partial):

```sql
PROC SQL;
SELECT STATE, SALES, (SALES * .05) AS TAX
FROM USSALES
WHERE STATE IN ('OH', 'IN', 'IL') AND TAX > 10;
ERROR: THE FOLLOWING COLUMNS WERE NOT FOUND IN THE CONTRIBUTING TABLES: TAX.
NOTE: PROC SQL SET OPTION NOEXEC AND WILL CONTINUE TO CHECK THE SYNTAX OF STATEMENTS.
```

Output #17 (partial):

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
<th>TAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>WI</td>
<td>10103.23</td>
<td>505.1615</td>
</tr>
<tr>
<td>WI</td>
<td>9103.23</td>
<td>455.1615</td>
</tr>
<tr>
<td>WI</td>
<td>15032.11</td>
<td>751.6055</td>
</tr>
<tr>
<td>IL</td>
<td>20338.12</td>
<td>1016.906</td>
</tr>
</tbody>
</table>

Output #18 (The resulting SAS LOG- partial):

```sql
PROC SQL;
SELECT STATE, STORE, SUM(SALES) AS TOTSAlES
FROM USSAlES
GROUP BY STATE
WHERE TOTSALES > 500;
+ 22
ERROR 202-322: THE OPTION OR PARAMETER IS NOT RECOGNIZED.
```
Output #19 (partial):

<table>
<thead>
<tr>
<th>STATE</th>
<th>STORENO</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>31212</td>
<td>10332.11</td>
</tr>
<tr>
<td>IL</td>
<td>31373</td>
<td>22223.12</td>
</tr>
<tr>
<td>IL</td>
<td>31381</td>
<td>32083.22</td>
</tr>
<tr>
<td>IL</td>
<td>31983</td>
<td>20338.12</td>
</tr>
<tr>
<td>WI</td>
<td>33281</td>
<td>33209.23</td>
</tr>
</tbody>
</table>

Output #20:

<table>
<thead>
<tr>
<th>STATE</th>
<th>TOTSALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>84976.57</td>
</tr>
<tr>
<td>WI</td>
<td>34238.57</td>
</tr>
</tbody>
</table>

Output #21:

<table>
<thead>
<tr>
<th>STATE</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL</td>
<td>20338.12</td>
</tr>
<tr>
<td>IL</td>
<td>10332.11</td>
</tr>
<tr>
<td>IL</td>
<td>32083.22</td>
</tr>
<tr>
<td>IL</td>
<td>22223.12</td>
</tr>
</tbody>
</table>