Paper BB-13-2014

Some useful SQL Procedures in SAS® – Applications in complex scenarios

Soma Ghosh, UnitedHealth Group, Minneapolis, MN

ABSTRACT

SQL Procedure (PROC SQL) is a very powerful feature provided by Base SAS. Apart from any DBMS or RDBMS, with SAS we see this implementation of Structured Query Language. For new SAS users, who have prior experience with SQL, it is very easy to migrate to SAS based projects by using PROC SQL. PROC SQL can be used as an alternative to DATA steps. This can be used to import data from SAS Datasets and any other external Database Systems. With PROC SQL we can create Tables, Views, and Indexes. In addition to this, it supports the theory of CRUD (Create, Read, Update and Delete) features.

PROC SQL makes some of the things easy to implement, like: ordering, grouping with having clause in a query, using aggregate functions, merging tables using joins, working on multiple query results with Union All, using Oracle Analytical functions with pass through queries.

PROC SQL can:

- Create different types of reports
- Summarize the statistical data
- Merge data from tables and views
- Create views and indexes
- Create and Modify SAS datasets
- Delete and Drop SAS datasets

INTRODUCTION

In this paper, I am going to cover some very useful PROC SQL statements which could be used in any SAS based project. Also this paper will cover some PROC SQL statements which could be easier and can work with fewer resources in compare to DATA steps. The queries defined here can motivate one to utilize expertise on other Structured Query languages.

Throughout this paper I will be referencing to the columns from sashelp.class via a temporary SAS datasets called “SAMPLE_CLASS” or from SASHELP.DEMOGRAPHICS via a temporary SAS datasets called “SAMPLE_DEMO”, which I will be creating in the work directory. [SAS provides over 200 data sets in the Sashelp library. These data sets are available to use for examples and for testing code.]

Here are some terminologies to differentiate SAS vs SQL:

SAS	SQl
Data Set	Table
Observation	Row
Variable	Column
Merge	Join
Extract	Query
Rename	As
Proc Print	Select
Proc / By	Group By

Creating temporary SAS datasets [called “SAMPLE_CLASS” & “SAMPLE_DEMO”].

```
PROC SQL;
 CREATE TABLE SAMPLE_CLASS AS SELECT * FROM SASHELP.CLASS;
QUIT;

PROC SQL;
 CREATE TABLE SAMPLE_DEMO AS SELECT * FROM SASHELP.DEMOGRAPHICS;
QUIT;
```

Pass through SQL Query:

Using Pass through SQL Query we can connect to any DBMS/RDBMS to pull data. This is a very useful feature, where we can use queries which are supported by the RDBMS/DBMS system from where the data are being imported. It is very important to pass the connectivity information accurately otherwise SAS fails to recognize whether it is reading from an external database and not from a disk file.

```
PROC SQL;
CONNECT TO ORACLE(USER=<USERNAME> ORAPW=<PASSWORD> PATH="<DATABASE INSTANCE NAME>");
CREATE TABLE INIT_PULL AS SELECT *
    FROM CONNECTION TO ORACLE
     (
    SELECT DISTINCT
      TAB1.COLUMN1
      ,TAB1.COLUMN2
      ,TAB2.COLUMN2
             FROM              SCHEMA.TABLE1 TAB1
     INNER JOIN     SCHEMA.TABLE2 TAB2 ON TAB1.COLUMN1=TAB2.COLUMN1
             WHERE   TAB1.COLUMN2 IN ( %BQUOTE('&VALUE') )
        );
       QUIT;
```

Above query will pull the data from Oracle by joining two tables and based on a criteria [**WHERE TAB1.COLUMN2 IN (%BQUOTE('&VALUE'))**]. Procedure for adding criteria is same like oracle. Date based criteria can also be included as we do in any SQL query. Example [**WHERE TAB1.DATECOL >=TO_DATE(‘01/01/2014’,’MM/DD/YYYY’)**].

With Pass through SQL Query, we can also use Oracle Analytical function, if connecting to Oracle 9i or higher.

An example is given below with a code snippet.

```
SELECT DISTINCT
  TAB1.COLUMN1
  ,TAB1.COLUMN2
  ,TAB2.COLUMN2
 ,SUM(TAB2.COLUMN4) OVER (PARTITION BY TAB1.COLUMN1) AS VARCOLUMN
```

In the above example, VARCOLUMN will store the sum of TAB2.COLUMN4 grouped by TAB1.COLUMN1

Using Rank with PROC SQL

We saw in previous example that, we can use Oracle Analytical functions with SAS as pass through query. But if we are importing data from a SAS Dataset, then we cannot use Oracle Analytical functions. However, some complex queries can be handled by procedure SQL.

Example: Rank:

```
PROC SQL;
   SELECT NAME, SEX, HEIGHT, (SELECT COUNT(DISTINCT B.HEIGHT)
   FROM SAMPLE_CLASS B
   WHERE A.SEX=B.SEX AND B.HEIGHT >= A.HEIGHT
    ) AS RANK
   FROM SAMPLE_CLASS A
   GROUP BY SEX
   ORDER BY SEX, RANK ;
QUIT;
```

The above example will rank based on Gender and Height. Top rank 1 will be assigned to the tallest person. Output is shown in Figure 1.

Name	Sex	Height	RANK
Mary	F	66.5	1
Barbara	F	65.3	2
Judy	F	64.3	3
Carol	F	62.8	4
Janet	F	62.5	5
Jane	F	59.8	6
Alice	F	56.5	7
Louise	F	56.3	8
Joyce	F	51.3	9
Philip	M	72	1
Alfred	M	69	2
Ronald	M	67	3
William	M	66.5	4
Robert	M	64.8	5
Henry	M	63.5	6
Jeffrey	M	62.5	7
John	M	59	8
Thomas	M	57.5	9
James	M	57.3	10

Figure 1: Result from the above query

Query with In-Line View

```
PROC SQL;
TITLE "REGION POPULATION";
   SELECT REGION,
          AVERAGE FORMAT=10.0 LABEL='AVERAGE POPULATION',
          MAX FORMAT=10.0 LABEL='MAXIMUM POPULATION'
```

```
FROM (SELECT REGION,
             AVG(POP) AS AVERAGE,
             MAX(POP) AS MAX,
             SUM(POP) AS TOTAL
             FROM SAMPLE_DEMO
             GROUP BY REGION)
ORDER BY AVERAGE;
```

In the above query, there is an in-line view used as a Sub-Query.

Output will look like the table given below:

REGION POPULATION

Region	AVERAGE POPULATION	MAXIMUM POPULATION
AFR	16045289	131529669
EUR	19304233	143201572
AMR	25323829	298212895
EMR	25619117	157935075
WPR	61785086	1323344591
SEAR	150593529	1103370802

Figure 2: Average and Maximum Population based on Region

Dropping duplicate rows:

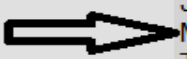
Deleting rows with duplicate key variables from a SAS data set can be done by SAS SQL Procedure as well.

```
PROC SQL;
    CREATE TABLE SAMPLE_OUT
         AS SELECT * FROM SAMPLE_CLASS GROUP BY SEX,AGE
                   HAVING MIN(WEIGHT) = WEIGHT;
 QUIT;
```

In the above example, result from the query is stored in a temporary (in the work directory) SAS dataset called “SAMPLE_OUT”. Resultant dataset will only include one row by Gender and the Age group. It will take the person with min weight and delete the rest.

Sample of Input and Output dataset:

Name	Sex	Age	Height	Weight
Joyce	F	11	51.3	50.5
Jane	F	12	59.8	84.5
Louise	F	12	56.3	77
Barbara	F	13	65.3	98
Alice	F	13	56.5	84
Carol	F	14	62.8	102.5
Judy	F	14	64.3	90
Mary	F	15	66.5	112
Janet	F	15	62.5	112.5
Thomas	M	11	57.5	85
James	M	12	57.3	83
John	M	12	59	99.5
Robert	M	12	64.8	128
Jeffrey	M	13	62.5	84
Alfred	M	14	69	112.5
Henry	M	14	63.5	102.5
William	M	15	66.5	112
Ronald	M	15	67	133
Philip	M	16	72	150



Name	Sex	Age	Height	Weight
Joyce	F	11	51.3	50.5
Louise	F	12	56.3	77
Alice	F	13	56.5	84
Judy	F	14	64.3	90
Mary	F	15	66.5	112
Thomas	M	11	57.5	85
James	M	12	57.3	83
Jeffrey	M	13	62.5	84
Henry	M	14	63.5	102.5
William	M	15	66.5	112
Philip	M	16	72	150

Figure 3: Result of a duplicate data deletion

One very important feature found in SAS SQL Procedure, which is unavailable in PL/SQL is: With Procedure SQL, we can have as many number of rows we want in the SELECT statement, irrespective of using those columns under Group By clause.

In the example given below, output dataset SAMPLE_DATA will have all the columns from SAMPL_DEMO for the region AMR, with highest population under each CONT.

```
PROC SQL;
    CREATE TABLE  SAMPLE_DATA AS SELECT *
    FROM
     SAMPLE_DEMO
                    WHERE REGION="AMR"
                     GROUP BY CONT
                HAVING POP=MAX(POP)
 ;
QUIT;
```

Using CASE statement like other SQ languages in SAS PROCEDURE SQL:

CASE can be used very efficiently like any other Structured Query languages to check any criteria. It can be used with multiple CASE..THEN..END or with one CASE Statement.

Examples given below:

```
PROC SQL;
    CREATE TABLE  SAMPLE_DATA AS SELECT A.*,
             CASE WHEN A.CONT =91 THEN "NORTH AMR" ELSE
                    CASE WHEN A.CONT =92 THEN "SOUTH AMR" ELSE
                                                        "OTHER"
                  END
                  END
```

```
     AS REG_DEF
     FROM
     SAMPLE_DEMO A
           WHERE A.REGION="AMR";
QUIT;

PROC SQL;
    CREATE TABLE  SAMPLE_DATA AS SELECT A.*,
            CASE WHEN A.CONT =91 THEN "NORTH AMR"
                              WHEN A.CONT =92 THEN "SOUTH AMR" ELSE
                "OTHER" END AS REG_DEF
    FROM
     SAMPLE_DEMO A
                   WHERE A.REGION="AMR";
QUIT;
```

Above examples will show same output. With the existing rows in the data set, a new row will be added at the end with the name REG_DEF, which will show “NORTH AMR”, “SOUTH_AMR”, “OTHER” based on the value available in the variable CONT.

Union All in PROC SQL

This can be used to concatenate data from multiple datasets. This feature is common like other SQL. ***Note: selected column names and data type should match across all the tables used to unite together.***

```
PROC SQL;
    CREATE TABLE  SAMPLE_DATA AS
        SELECT A.* FROM TAB1 A
                   UNION ALL
        SELECT B.* FROM TAB2 B
;
QUIT;
```

With **OUTER UNION CORR**, all the selected columns from both the tables will appear. Which means it will show the matching columns only once and then all unmatched columns.

First it will show the data from Table 1 and unmatched columns from Table 2 will show null values.

Next, it will insert rows from Table 2, where unmatched columns from Table 1 will have null values and matched columns from Table 1 will show data. Along with those rows, unmatched columns from Table 2 will show data.

With **OUTER UNION**, all the selected columns from both the tables will appear which means it will show the matching columns only once and then all unmatched columns.

First it will show the data from Table 1 and unmatched columns from Table 2 will show null values.

Next, it will insert rows from Table 2, where columns from Table 1 will have null values and unmatched columns from Table 2 will show data.

Creating Views with SAS PROC SQL

A SAS view is like a SAS data set and it pulls data from other files. It is like a copy of a SAS dataset. A SAS view stores descriptive information, which are basically the type of data and lengths of the variables. SAS views are of type as member. In most cases, SAS view is just like a SAS data file.

Structure of SAS Views

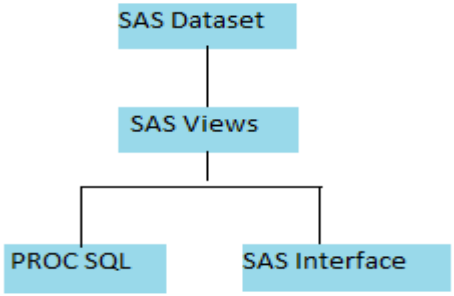


Figure 4: SAS Views structure

```
PROC SQL;

CREATE VIEW SAMPLE_VIEW AS
         SELECT *
            FROM SAMPLE_DEMO
            WHERE REGION="AMR";
QUIT;
```

Creating Index with SAS PROC SQL

Indexing is a very important feature to improve the performance, especially when the dataset is large in size, with huge number of observations. We can add index on multiple columns based on the criteria defined in SQL query. If there are criteria included for four columns, it would be better in terms of performance if all those four columns are indexed. Concept of index in SAS PROC SQL is similar to any DBMS/RDMS indexing theory.

PROC SQL can create single and composite index. Examples provided below

Single Index

```
PROC SQL;
CREATE INDEX AGE ON WORK.SAMPLE_CLASS(AGE);
QUIT;
```

Composite Index

```
PROC SQL;
CREATE INDEX CALC ON WORK.SAMPLE_CLASS(WEIGHT,HEIGHT);
QUIT;
```

To check that each value of the indexed column is unique, UNIQUE keyword can be used.

```
PROC SQL;
CREATE UNIQUE INDEX CALC ON WORK.SAMPLE_CLASS(WEIGHT,HEIGHT);
QUIT;
```

JOINing (MERGE) Data sets using PROC SQLs

Like other Structured Query Languages, join can be used with PROC SQL to retrieve data from multiple tables based on some columns, which are common to all the linked/joined tables/datasets.

Join could be inner or left/outer like any other Structured Query Language.

Types of Join	Result
Inner	Pulls only those rows, where the values from column used in join match with each other
Outer	Pulls rows match across the tables and also rows from the table used in join

Figure 5: Join Types

```
SELECT COLUMN1
      ,COLUMN2
FROM TABLE1
INNER JOIN TABLE2
ON TABLE1.COLUMN_NAME=TABLE2.COLUMN_NAME;
```

With join we can show the lag values as well with PROC SQL. Here is an example:

```
PROC SQL;
CREATE TABLE SAMPLE_LAG AS SELECT *,MONOTONIC() AS RNUM FROM SAMPLE_CLASS;
QUIT;

PROC SQL;
   CREATE TABLE SAMPLE_LAG_DATA(DROP=RNUM) AS SELECT A.*,
          (SELECT B.AGE FROM SAMPLE_LAG AS B
            WHERE A.RNUM - B.RNUM = 1) AS NEXT_AGE
    FROM SAMPLE_LAG AS A
    ORDER BY A.RNUM
;
RUN;
```

The above query will create a new column called NEXT_AGE by moving the value from the column age from one row to its next.

Understanding Advantages of PROC SQL Joins

Here are some of the key features of PROC SQL joins.

Operator/Type	Features	Example

Left Join	In PROC SQL, it is not necessary to sort the dataset before joining.	`PROC SQL;`
		`SELECT A.VAL1, A.VAL2, B.VAL4`
		`FROM TAB1 A`
		`LEFT JOIN`
		`TAB2 B`
		`ON A.VAL1 = B.VAL1;`

	In PROC SQL, it is not necessary that the column names to be same with the criteria/expressions.	`PROC SQL;`
		`SELECT A.NAME,`
		`A.WEIGHT`
		`FROM TAB1 A, TAB2 B`
		`WHERE A.MATCHCOL =`
		`B.MATCHCOL;`

Inner / Outer Join	In PROC SQL, inner and outer join can be defined.	`PROC SQL;`
		`SELECT A.NAME,`
		`A.WEIGHT`
		`FROM TAB1 A INNER JOIN TAB2 B`
		`ON A.MATCHCOL =`
		`B.MATCHCOL;`

>=<	In PROC SQL, we can use any other operation like >, <, etc.	`PROC SQL;`
		`SELECT A.NAME, A.SEX,`
		`A.WEIGHT`
		`FROM TAB1 A,`
		`TAB2 B`
		`WHERE A.ID = B.IDN`
		`AND A.WEIGHT>B.WEIGHT;`

Figure 6: Table shows Advantages of PROC SQL

At the end of this paper, here is a very important functionality which can be used with PROC SQL: It is **dynamically storing multiple values in one variable.**

This concept is similar to an Array, where multiple values can be stored in a single variable. This is very important in any programming language where we have more than one value under a single variable. In the example given below we will see, how we can assign values into one variable with PROC SQL.

```
%MACRO DIN_VAL;

PROC SQL;
 CREATE TABLE SAMPLE AS SELECT * FROM SASHELP.DEMOGRAPHICS;QUIT;
PROC SQL;
    CREATE TABLE  SAMPLE_DATA AS SELECT COUNT(*) AS CNT
    FROM
     SASHELP.DEMOGRAPHICS
                WHERE REGION="AMR"
                     GROUP BY CONT
              HAVING POP=MAX(POP); QUIT;
%LET OBS_CNT=0;
 PROC SQL NOPRINT;
         SELECT CNT INTO:OBS_CNT FROM SAMPLE_DATA;
 QUIT;
```

```
%IF %EVAL(&OBS_CNT GT 0) %THEN %DO;
%DO ITERATION=1 %TO &&OBS_CNT.;
PROC SQL NOPRINT;
    SELECT DISTINCT "'"||NAME||"'" ,"'"||ISONAME||"'",
    POP,POPAGR,POPURBAN,GNI INTO
    :NAME&ITERATION. ,
    :ISONAME&ITERATION.,
    :POP&ITERATION.,
    :POPAGR&ITERATION.,
    :POPURBAN&ITERATION.,
    :GNI&ITERATION.
 FROM SAMPLE WHERE MONOTONIC ()=%SYSFUNC(FLOOR(%EVAL((&ITERATION.))));
QUIT;
%PRINT_REPORT(NAME=&&NAME&ITERATION.,ISONAME=&&ISONAME&ITERATION.,
POP=&&POP&ITERATION., POPAGR=&&POPAGR&ITERATION. ,POPURBAN=&&POPURBAN&ITERATION.
,GNI=&&GNI&ITERATION. );

%END;
%END;

%MEND DIN_VAL;
%DIN_VAL;
```

Note: Print_Report is a user defined macro and can be defined based on the functionality required to perform.

Apart from all these, we can have add on features of using SAS functions for formatting character, number date variables in PROC SQL Queries.

Some Tips:

Following features can also be used with PROC SQL:

Monotonic (): Provides row number. This function can be used as a column in the select statement or can also be used in the where criteria.

Example: `PROC SQL; SELECT *, MONOTONIC () AS ROW_NUM FROM SAMPLE_CLASS;`

OUTOBS= option: OUTOBS can be used to restrict the number of rows that are displayed. To restrict the number of rows that PROC SQL consider as input from any single source, we can use the INOBS= option

Example: `PROC SQL OUTOBS=n;` where *n* specifies the number of rows.

DISTINCT: Used to eliminate duplicate rows.

Example: `SELECT DISTINCT NAME, SEX, AGE FROM SAMPLE_CLASS.`

One semicolon only is required at the end of the query

SAS OPERATORS:

In PROC SQL queries, we can also use the following conditional operators. All of these operators can also be used in other SAS procedures.

Operator	Logic	Example
BETWEEN – AND	Checks if values are within the range	`WHERE WEIGHT BETWEEN 100 AND 200`
CONTAINS	Checks for values, that stores a specific text	`WHERE NAME CONTAINS 'P'`
?	Checks for values, that stores a specific text	`WHERE NAME ? 'P'`
IS MISSING or **IS NULL**	Checks for missing values or null values	`WHERE WEIGHT IS MISSING` `WHERE WEIGHT IS NULL`
LIKE (with **%**, _)	Checks for given pattern matching	`WHERE ADDRESS LIKE '%PLACE'`
=*	Checks if the pattern provided as criteria, sound like a specified value	`WHERE NAME=* 'Alfred'`
IN	Checks at least one of the values, that match	`WHERE NAME IN ('Alfred','Henry','Mary')`

ANY	Checks where the given criteria meets with respect to any one of the values returned by a sub-query	`SELECT *`
		`FROM SAMPLE_CLASS`
		`WHERE SEX='M' AND WEIGHT < ANY`
		`(SELECT WEIGHT`
		`FROM SAMPLE_CLASS`
		`WHERE SEX='F')`

ALL	Checks if a given criteria meets in respect to all the values returned by a sub-query	`SELECT *`
		`FROM SAMPLE_CLASS`
		`WHERE SEX='M' AND WEIGHT < ALL`
		`(SELECT WEIGHT`
		`FROM SAMPLE_CLASS`
		`WHERE SEX='F')`

EXISTS	Checks if the value(s) returned by a sub-query, is/are available or not.	`SELECT * FROM SAMPLE_CLASS A WHERE EXISTS`
		`(SELECT *`
		`FROM SASHELP.CLASSFIT B`
		`WHERE A.NAME=`
		`B.NAME)`

CONCLUSION

With SAS SQL Procedures we can write queries to perform all the required jobs in SAS and can efficiently use existing functions available in Oracle, Teradata, etc.

PROC SQL is a very powerful feature and can be used is every SAS program and can replace DATA steps if required based on user's expertise's.

REFERENCES

Chao Huang, Yu Fu. 'Top 10 SQL Tricks in SAS®'. SAS Global Forum 2014 Proceeding.
http://support.sas.com/resources/papers/proceedings14/1561-2014.pdf

CONTACT INFORMATION

The author can be contacted at soma.ghosh@optum.com and ghosh.soma@gmail.com

TRADEMARK CITATION