Paper 042-29

Top Ten Reasons to Use PROC SQL

**Weiming Hu, Center for Health Research Kaiser Permanente,
Portland, Oregon, USA**

ABSTRACT

Among SAS® users, it seems there are two groups of people, those who love PROC SQL and those who hate PROC SQL. Personally I fell in love with SAS SQL right after I was introduced to it in 1994. I see SAS PROC SQL as being complementary to existing SAS procedures and the DATA Step. I find a lot of tasks can be done more easily in PROC SQL, and sometimes they are not even possible in other SAS procedures. This paper is intended to share my experience with other SAS users, especially non-SQL users who want to learn SQL. The purpose is to explore alternative methodologies, as we know that in SAS there are hundreds of ways to get the same thing done.

INTRODUCTION

Structured Query Language (SQL) is a language primarily used for retrieving data from relational databases. Ever since SAS implemented SQL (PROC SQL) in Version 6.0, it opened a whole new arena for SAS users. I see the distinct advantage of PROC SQL as making SAS programming easier and SAS code more intuitive. A simple SQL statement sometimes is equivalent to several SAS procedures and DATA Steps. In this paper, I will go over some usages of PROC SQL, which either simplify SAS programming or produce output that is not available in the non-SQL part of SAS. This paper will not assess efficiency or other system issues (readers can look in the previous SUGI papers to see the comparisons). More often than not, PROC SQL will come out as a winner.[1] Because computers now run much faster, and memory is much cheaper, I would rather focus on writing the SAS code and getting the job done and not worry about CPU time. Since I work in a health service research environment, many of the examples given below will be health care related **.**

Top 10 reasons to use SAS PROC SQL

10. Join tables

This is probably the most common usage of PROC SQL. The following SQL examples join two or more tables. All rows from the left are returned with information added from the right tables on the match column(s). The advantages are:

- No sorting needed.
- Two tables can join on different variable names.

In example A, HRN (Health Record Number) and CHART are patients' unique ID variables. The SQL statement is the equivalent of two PROC SORT procedures and one DATA Step merge. Example B demonstrates that you can join different tables based on different variables. (Caution: HRN or CHART or CASEID needs to be unique in each of the tables, otherwise you may end up with more rows than you anticipated.)

A) Two tables

```
PROC SQL; CREATE TABLE MERGED AS
 SELECT *
 FROM SAMPLE AS A LEFT JOIN VISITS AS B
 ON A.HRN=B.CHART
;QUIT;
```

```
B) Several tables

        PROC SQL; CREATE TABLE SRD_CAT AS
         SELECT *
         FROM SAMPLE AS A LEFT JOIN DEMO     AS B ON A.HRN=B.CHART
                          LEFT JOIN LUNG_CX  AS C ON A.HRN=C.HRN
                          LEFT JOIN CVD      AS D ON A.HRN=D.HRN
                          LEFT JOIN COPD     AS E ON A.HRN=E.HRN
                          LEFT JOIN ELIG     AS F ON A.CASEID=F.CASEID
        ;QUIT;
```

9. Build macro value list

The following SQL example assigns a whole column of values to a macro variable. This can be useful in two situations:

- Outputting information to title/footnote statements using macro variables when you don't know the new values (department code) in advance.
- Using the macro variable as the value for your IN statement (see below).

The limit for the length of the macro variable is quite long. Under Windows SAS version 8, for 8 digit HRN, the macro variable can hold 7200+ HRNs.

```
PROC SQL NOPRINT;
  SELECT QUOTE(TRIM(DEPT)) INTO :DEPT_LIST SEPARATED BY ', '
  FROM A
;QUIT;

 %PUT DEPT_LIST: &DEPT_LIST;
 DEPT_LIST: "EAP", "HEHA", "XYZ"
```

8. Access other databases

Below is SQL code to query an Oracle table from PC SAS. (OO.CMS_MEMBER is our membership file with 400,000+ rows). PROC SQL is the only way you can join a SAS table and an Oracle table. Method 1 uses join while Method 2 uses subquery. As we can see from the chart on the next page Method 2 is super fast for a small sample.

```
*--METHOD 1--;
PROC SQL;
 CREATE TABLE TTT AS
 SELECT B.HRN, FAMACT, RELTN
 FROM XSAMPLE  AS A INNER JOIN OO.CMS_MEMBER (dbkey=hrn dbindex=yes) AS B
 ON A.HRN =B.HRN
;QUIT;

*--METHOD 2--;
PROC SQL NOPRINT; SELECT HRN INTO :HRN_LIST SEPARATED BY ',' FROM xsample; QUIT;
PROC SQL;
 CREATE TABLE TTT AS
 SELECT HRN, FAMACT, RELTN
 FROM OO.CMS_MEMBER (dbkey=hrn dbindex=yes)
 WHERE HRN IN (&HRN_LIST)
;QUIT;
```

Note: The maximum for &HRN_LIST under Windows SAS version 8, for 8 digits HRN, is about 1000 HRNs; in other words, the SQL limit comes before SAS Macro.

Comparison of JOIN and SUBQUERY

Sample size		CPU	clock
100	method1	0.36	16.70
	method2	0.05	1.04
500	method1	1.17	1:07.60
	method2	0.14	1:36.00
1000	method1	2.87	2:22.37
	method2	0.29	3:56.61

7. Textwrapping

When you have a long character variable (such as a COMMENT field in the questionnaire), and you want to print the values using PROC PRINT, you will get the warning message:

WARNING: Data too long for column "COMMENT"; truncated to 124 characters to fit.

A simple solution is to use PROC SQL with the flow option. An alternative would be to use PROC REPORT. (Note: Flow=30 has an effect on all character variables.)

```
PROC SQL FLOW=30;
 SELECT HRN, COMMENT
 FROM A
;QUIT;
```

```
     HRN  COMMENT
-------------------------------------
12345678  LONGTEXTTTTTTTTTTTTTTTTTTTT
          TTTTTTTTTTTTTTTTTTTTTTTTTT
          TTTTTTTTTTTTTTTTTTTTTTTTTT
          TTTTTTTTTTTTTTTTTTTTTTTTTT
87654321  LONGTEXTTTTTTTTTTTTTTTTTTTT
          TTTTTTTTTTTTTTTTTTTTTTTTTT
          TTTTTTTTTTTTTTTTTTTTTTTTTT
          TTTTTTTTTTTTTTTTTTTTTTTTTT
```

6. Count frequencies

Using PROC SQL, you can quickly count non-missing values for several variables and output the result on one line. (PROC FREQ would produce several output tables with the output sometimes continuing on to the next page.)

```
PROC SQL; SELECT COUNT(*) AS TOTAL,
       COUNT(DIAG0001) AS DX1,
       COUNT(DIAG0002) AS DX2,
       COUNT(DIAG0003) AS DX3,
       COUNT(DIAG0004) AS DX4
FROM INP; QUIT;
```

TOTAL	DX1	DX2	DX3	DX4
1562	1562	1421	1163	814

5. Matching multiple tables at different levels

Below I join 3 tables. The task is to get the inpatient diagnoses residing in OO.ADT_DIAG (MAIN_KEY as match variable) for my SAMPLE table (HRN as match variable). The middle table (OO.ADT_REG) serves as a link for the other 2 tables because it contains both matching variables (HRN and MAIN_KEY).

```
PROC SQL;
 CREATE TABLE ADT AS
 SELECT A.HRN, DIAG
 FROM SAMPLE AS A , OO.ADT_REG (dbkey=HRN dbnullkeys=no) AS B,
                    OO.ADT_DIAG (dbkey=MAIN_KEY dbnullkeys=no) AS C
 WHERE A.HRN = B.HRN and B.MAIN_KEY=C.MAIN_KEY
;QUIT;
```

4. Insert records to a table

Below is code to calculate the mid-year membership count from an eligibility file for each year from 1986 to 2002. (This is just for demonstration purposes - the code is not very efficient). I have a macro %DO-%END loop where each iteration will produce two macro variables and PROC SQL will insert a new record into the table. Compared to BASE SAS processing, using SQL saves a step. (In BASE SAS you would have to create a one-record table, then append it to the master table.)

```
%MACRO MULTI_YR (BY=, EY= );

  *---creating empty table---;
 DATA MYEARPOP; MYEAR=.; POP=.; DELETE; RUN;

 %DO I= &BY %TO &EY;

  PROC SQL NOPRINT;
   SELECT COUNT(*) INTO :RECORDS
   FROM CCPSSD.KPOPGAP2
   WHERE FDATE LE "01jul&I"D LE TDATE;
  ;QUIT;       %put &records;

  PROC SQL; INSERT INTO MYEARPOP
   SET MYEAR=&I,POP=&RECORDS
  ;QUIT;

 %END;

%MEND MULTI_YR;

%MULTI_YR(BY=1986,EY=2002 );
```

3. COALESCE function

In our administrative tables, the Social Security Number (SSN) field is not very well populated. We need to go after different sources: membership tables (old, current, and daily) and other utilization tables. PROC SQL makes the selection process very easy, where the COALESCE function will pick the first non-missing value.

```
PROC SQL; CREATE TABLE _SSNINFO AS
 SELECT S.*, COALESCE (C.SSN, A.SSN, B.SSN, D.SSN) AS SSN,
 FROM _SAMPLE AS S LEFT JOIN _CMS     AS A ON S.HRN=A.HRN
                   LEFT JOIN _MG      AS B ON S.HRN=B.HRN
                   LEFT JOIN _CMSDL   AS C ON S.HRN=C.HRN
                   LEFT JOIN _DOCPLUS AS D ON S.HRN=D.HRN
;QUIT;
```

2. Summarize data

You can use SQL functions to summarize data. PROC SQL is more intuitive than PROC MEANS or PROC SUMMARY, where SAS will create an output table that always contains more rows and columns than you need and you have to choose the right _TYPE_ value. PROC SQL below summarizes the total number of cases by age group for each year.

```
PROC SQL;
 SELECT CHMPYEAR, AGEGRP, SUM(CASES) AS CASES
 FROM V6DIR.SVI4_AGE
 GROUP BY 1,2
;QUIT;
```

1. Fuzzy merge

Fuzzy merge is the process of matching records where the condition of a match is based on close-but-not equivalent condition. In survival analysis where we need to know whether a member is dead, one important step is to match our sample to the records in the death tape by SSN, birthday, and name. We compare matching variables and assign points for every match. The score of 16 would be a perfect match. In real life, this is not always the case, so our rule is that a score of 13 and above will be considered a match. For any score between 9 and 12 we will do a manual check to determine whether it is a match.

```
PROC SQL;
 CREATE TABLE REVIEW AS
 SELECT *
 FROM SAMPLE, OW_DEATH
 WHERE SUM( ((KBMON  =SBMON)*2), ((KBDAY  =SBDAY)*1),
          ((KBYEAR =SBYEAR)*2),
          ((KFSNDX =SFSNDX)*1), ((KFNAME =SFNAME)*1),
          ((KMNAME =SMNAME)*1),
          ((KLSNDX =SLSNDX)*1), ((KLNAME =SLNAME)*1),
          ((KSEX   =SSEX)*2) ,  ((KSSN   =SSSN and KSSN ne ' ')*4)  )
          >=9;
QUIT;
```

CONCLUSION

PROC SQL is a powerful tool. It can make your life much easier. For novice SQL users, I would like to offer my caveats:

1. Be careful about many to many table joins in SQL. When joining tables that have multiple records per matching ids, the output table may be a Cartesian product. For example, 3 rows joining 5 rows of same id variable will produce 15 rows, as compared to the DATA Step MERGE where only 5 rows will be created.

2. PROC SQL is code-saving, but not always time-saving.

REFERENCES

The reader is directed to the following papers for additional and important information on SAS PROC SQL.

1. Steven Feder. “Comparative Efficiency of SQL and Base Code When Reading from Database Tables and Existing Data Sets.” Proceedings of SAS User Group International Conference, Paper 76-28, Seattle, Washington, USA. 2003.

2. Ian Whitelock. “PROC SQL – Is It a required Tool for Good SAS programming?” Proceedings of SAS User Group International Conference, Paper 60-26, Long Beach, CA, USA 2001.

3. Kevin J. Smith, et al. “PROC SQL vs. Merge. The Miller Lite Questions of 2002 and Beyond.” Proceedings of SAS User Group International Conference, Paper 96-28, Seattle, Washington, USA. 2003.

TRADEMARKS

ACKNOWLEDGEMENTS

The author would like to express his appreciation to Pamela and Steve Balch for their assistance in this paper.

CONTACT INFORMATION

You can send your comments, questions, and inquiries to:

Weiming Hu, Senior Research Analyst
Center for Health Research
Kaiser Permanente Northwest Region
3800 N Interstate Ave.
Portland, OR 97227-1110
Tel.: 503.335.6770
Fax: 503.335.2424
Email: Weiming.r.hu@kpchr.org