

Comparing Proc SQL to Standard SAS* Coding Techniques

Howard Levine, Levine Software Systems, Inc.

INTRODUCTION

Proc SQL is a major addition to the Base product of the SAS System. It allows significantly greater flexibility in accessing data bases than many of the other procedures and data steps allow. Many tasks that can be performed using procedures and data step can be accomplished with fewer, simpler statements with SQL. Also, some things that would require complex data step programming, sophisticated macros, and SAS formats can be accomplished very simply with Proc SQL. This paper will show coding examples for a few selected situations to compare standard SAS code with SQL for performing the same task.

Types of operations that will be compared:

- * Printing SAS data sets
- * Sorting data sets
- * Summarizing data sets
- * Simple merge of two data sets
- * Merge of three data sets without all three having a common key
- * "Fuzzy" merges when the look-up table is a range rather than a set of single values.

The primary issue that will be addressed is simplicity of code. Computer performance information will not be stressed because it can vary from one platform to another. Also, the presence or absence of data set indexes can affect performance - sometimes in unexpectedly negative ways with data steps.

In order to see how different tasks are performed, we will use several data sets as examples. They are described in the appendices.

PRINTING DATA

This is done easily with either Proc PRINT or Proc SQL. In addition, Proc REPORT can be used for printing. Proc PRINT has more options and features for printing than are available with Proc SQL, but SQL can print calculated values and sort the report without running Proc SORT. Proc REPORT is the most versatile and powerful of the data printing tools available, but it is not typically used to print quick, ad hoc reports. To print all of the variables in the CITIES data set using Proc PRINT, the following code could be used:

```
PROC PRINT DATA = DATA.CITIES;  
RUN;
```

In SQL the following code would be used:

```
PROC SQL;  
  SELECT *  
  FROM DATA.CITIES;
```

SORTING DATA

Sorting data sets is simple using either Proc SORT or Proc SQL.

The conventional code for sorting data set data.cities by city name and storing the sorted file as another data set is shown below:

```
PROC      SORT      DATA=DATA.CITIES
OUT=SORTED;

      BY CITY;

RUN;
```

In contrast, the SQL code looks like this:

```
PROC SQL NOPRINT;

      CREATE TABLE SORTED AS

      SELECT *

      FROM DATA.CITIES

      ORDER BY CITY;
```

In this case, the standard SAS code is simpler to use than SQL. There is no difference in computer efficiency between the two methods.

Sorting in SQL makes more sense if you wish to sort by a value that is a function of one or more variables. For example, if an employee compensation data set contains the following variables: name, salary, and bonuses; then you may want to sort the data set by total compensation rather than just salary. With conventional SAS code, you would have to write a data step and run proc sort as shown below:

```
DATA COMP1;

      SET DATA.EMPCOMP;

      COMP = SALARY + BONUS;

RUN;

PROC SORT DATA=COMP1;

      BY DESCENDING COMP;

RUN;
```

This SQL code does the same thing:

```
PROC SQL NOPRINT;

      CREATE TABLE COMP1 AS

      SELECT *,SALARY+BONUS AS COMP

      FROM DATA.EMPCOMP

      ORDER BY COMP DESC;
```

The SQL is almost as complex as the Standard SAS code. But the SQL does not require the programmer to create an intermediate data set with the calculated value. SQL can work more efficiently in this situation because it can reduce the number of times the data is handled.

SUMMARIZING DATA

SQL can also handle some data summarizing functions such as counting items, averaging, and summing. It can essentially do the same things as the SUMMARY or MEANS procedure.

For example, if we wanted to print the total of salaries paid to all employees, this can be done with PROC MEANS with the following code:

```
PROC MEANS DATA=DATA.EMPCOMP SUM;

    VAR SALARY;

RUN;
```

With SQL, the code would look like this:

```
PROC SQL;

    SELECT SUM(SALARY) TOTAL
    SALARY
    FROM DATA.EMPCOMP;
```

In both examples, the result will be similar and there will be no significant difference in processing time. There is no significant difference in complexity of code.

However, if the sum of total compensation is desired, using Proc SQL instead of Proc MEANS can save one pass through the data. Since a new variable COMP must be calculated, Proc MEANS would require a prior data step. Proc SQL would not.

THREE WAY MERGE

Another area that merits attention is the three way merge. This involves having two tables that are not directly related to one another being related indirectly through a third table. An example is the following three tables having the columns shown:

- 1) TABLE1 with CITY and STATE

- 2) TABLE2 with STATE and COUNTRY

- 3) TABLE3 with COUNTRY and LANDMASS.

If all cities are located within states, all states are located in a country, and all countries are located on a landmass (continent), then TABLE1 is indirectly related to TABLE3 through TABLE2. To illustrate this, if you want to find out what landmass a city is on, TABLE1 must be merged with TABLE2 to get country information. Only then can landmass information be obtained from TABLE3. The code required for associating each city with its landmass is shown both in conventional SAS code using sorts and merges, "sophisticated" SAS code using Proc FORMAT and the PUT function, and with Proc SQL.

- 1) Sorts and Merges

```
PROC SORT DATA=TABLE1;
    BY STATE;
RUN;

PROC SORT DATA=TABLE2;
    BY STATE;
RUN;

DATA TEMP1;
    MERGE TABLE1 TABLE2;
    BY STATE;
RUN;

PROC SORT DATA=TEMP1;
    BY COUNTRY;
RUN;

PROC SORT DATA=TABLE3;
    BY COUNTRY;
RUN;

DATA RESULT;
    MERGE TEMP1 TABLE3;
    BY COUNTRY;
RUN;
```

2) Proc FORMAT and PUT function

```
DATA FMT2;
    SET TABLE2;
    RETAIN FMTNAME 'NATION' TYPE
    'C';
RUN;

PROC FORMAT CNTLIN = FMT2 (RENAME =
    (START = STATE LABEL = COUNTRY));
RUN;
```

```
DATA FMT3;
    SET TABLE3;
    RETAIN FMTNAME 'LANDMS' TYPE
    'C';
RUN;

PROC FORMAT CNTLIN = FMT3 (RENAME =
    (START = COUNTRY LABEL = LANDMASS));
;
RUN;
```

```
DATA RESULT;
    SET TABLE1;

    COUNTRY = PUT(STATE,$NATION.);

    LANDMASS =
    PUT(COUNTRY,$LANDMS.);
RUN;
```

3) Proc SQL

```
PROC SQL NOPRINT;

    CREATE TABLE RESULT AS

    SELECT TABLE1.CITY,
    TABLE1.STATE, TABLE2.COUNTRY,
    TABLE3.LANDMASS*

    FROM TABLE1, TABLE2, TABLE3

    WHERE TABLE1.STATE =
    TABLE2.STATE AND TABLE2.COUNTRY =
    TABLE3.COUNTRY;
```

In terms of coding simplicity, Proc SQL is the best of the three techniques unless the SAS formats have already been defined. With the formats already defined, Method #2 is the best in terms simplicity of the code. In terms of processing efficiency, Proc SQL will usually be more efficient if all of the data sets are indexed appropriately. Techniques with SAS formats are usually very efficient if the formats have already been created. With unindexed data sets, sort order and size of the data sets will affect which method is the most efficient with respect to computer resources. The frequency with which the data sets are updated will also affect your decision about what technique is best because the cost of sorting, indexing, and creating formats are fixed costs that only need to be incurred when there is a change in the data sets.

FUZZY MERGES

A fuzzy merge is when data sets are "merged" based on values falling within a certain range rather than matching exactly. More generally, a fuzzy merge can be thought of as merge based on inequalities. With standard SAS code, these can be handled by using formats. Conventional match merging with a BY statement will not work. Fuzzy merges are not a problem for SQL. The following example will illustrate how the coding techniques differ.

A key difference between using SAS formats and Proc SQL for performing fuzzy merges is that SAS formats do not allow a key value to match more than one item on the look-up table. Proc SQL allows the data set used as a look-up table to have overlapping value ranges. For example, the ranges 10-20 and 15-25 can occur using Proc SQL, but cannot occur with SAS formats. Overlapping ranges means that each record from a main data file may have two records brought in from the look-up table. This is important to understand because it can be a source of difficult to understand errors, but we will not use overlapping ranges in any examples.

Using the employee compensation data base from another example, we will categorize salaries into LOW, MEDIUM, and HIGH. We create a data set SALCAT with the following values:

```
FMTNAME = 'SALCAT' and
TYPE = 'N' for all observations
```

START END	LABEL
-----------	-------

0	14000	LOW
14001	50000	MEDIUM
50001	1000000	HIGH

To create a new data set with both the salary and salary category, a SAS format can be created and used with the PUT function. The code for this is shown below:

```
PROC FORMAT CNTLIN=SALCAT;
RUN;
```

```
DATA NEWCOMP;
```

```
    SET DATA.EMPCOMP;
```

```
    SALCAT = PUT (SALARY,
SALCAT.);
```

```
RUN;
```

The SQL code is shown below:

```
PROC SQL NOPRINT;
```

```
    CREATE TABLE NEWCOMP AS
```

```
    SELECT A.SALARY, A.BONUS,
B.LABEL AS SALCAT
```

```
    FROM DATA.SALARY A. SALCAT B
```

```
    WHERE SALARY BETWEEN B.START
AND B.END;
```

In this case, the conventional SAS code is easier to read and code than the SQL. However, if the data set NEWCOMP has been saved as a view rather than a table in Proc SQL, it would be possible to have access to the data without actually making a new data set. This would you to take advantage of the latest changes in salary categories without having to explicitly run a program to update a SAS format and the NEWCOMP data set.

CONCLUSION

This has been a brief overview of how SQL code and conventional SAS coding techniques compare. Proc SQL is shown to be a versatile data management tool that can expand the capabilities of the SAS System, simplify some code, and duplicate some of the tasks performed by conventional SAS code.

For further information, feel free to contact the author at

5525 Queen Avenue S.

Minneapolis, MN 55410

612-926-6967

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA

References:

Gerend, Patricia S., "Normalized Data: Implications for the use of SAS Software, SQL, and Views," *Proceedings of the Sixteenth Annual SAS Users' Group International Conference*, 1991, pp. 536-540

Kretzman, Peter, "Old Wine, New Wineskins: Why the SQL Procedure Will Change the Way You Think," *Proceedings of the Sixteenth Annual SAS Users' Group International Conference*, 1991, pp. 271-276

SAS Institute Inc., *SAS Guide to the SQL Procedure*, 1989

APPENDIX CONTAINING DATA SET PRINT OUTS

DATA.CITIES

CITY

MINNEAPOLIS
ST. PAUL
BUFFALO
CLEVELAND
TORONTO

TABLE3

LANDMASS COUNTRY

NORTH AMERICA UNITED STATES
NORTH AMERICA CANADA
EUROPE SWITZERLAND

DATA.EMPCOMP

EMPNO SALARY BONUS

4001	73985	22008
4002	59929	939
4003	68641	21927
4004	17191	3169
4005	19917	6656

TABLE1

CITY STATE

MINNEAPOLIS MINNESOTA
ST. PAUL MINNESOTA
BUFFALO NEW YORK
CLEVELAND OHIO
TORONTO ONTARIO
ZURICH ZURICH CANTON

TABLE2

STATE COUNTRY

NEW YORK UNITED STATES
OHIO UNITED STATES
MINNESOTA UNITED STATES
ONTARIO CANADA
ZURICH CANTON SWITZERLAND