Paper 104-26

Merging Tables in DATA Step vs. PROC SQL: Convenience and Efficiency Issues

Gajanan Bhat, PAREXEL International, Waltham, MA
Raj Suligavi, 4 C Solutions, Inc., East Moline, IL

ABSTRACT

The main objective of this paper is to discuss the convenience and efficiency of processing various types of data merges using DATA Step and PROC SQL. The paper shows strengths and weaknesses of both approaches in various types of merges and how and where to use them. DATA Step is an easy, convenient and safer approach for some types of merges but may not be efficient and powerful for some other types of data merges. PROC SQL is more powerful for some types of merges such as match merge. However, caution must be taken to use the latter approach. This paper also demonstrates the efficiency of both approaches in terms of time and memory use.

INTRODUCTION

Efficient programming is the one that gets more results from fewer resources. The efficiency is measured mainly in two ways, computer efficiency, human efficiency, and the trade-off between the two. Computer efficiency involves CPU time, I/O time, memory, and data storage. Human efficiency involves programming time and techniques of programming that decrease the time for design, coding, testing, debugging, and mainly processing of the programs to get the desired results.

One of the important tasks performed using SAS system in various industries for data retrieval and analysis is pulling together multiple sets of data. Merge is one of the important techniques of combining data sets using DATA Step. However, merge can take a considerable amount of processing time as data sets get larger. Merging two data sets require both data sets to be sorted first and read. Alternatively, PROC SQL is also used to accomplish the task of combining data sets/tables that a merge can do. The purpose of this paper is to discuss the efficiencies and ease of use of both techniques. The paper demonstrates some few types of merges/joins and analyzes the processing time and the complexity of coding.

METHOD

This paper, with two scenarios, demonstrates mainly three types of merges: simple merge, merge with index or keyed access method, and the PROC SQL join. The two scenarios that were used to illustrate the efficiencies of various methods are explained as follows. Time trial presented here uses mainly two data sets, VISITS and AE. VISITS has data on patient ID (PATNO), visit ID (VISID), phase within a visit (PHASE), and visit date (VISDT) along with other information on patients and his visits. AE, along with lot of other information, has PATNO, VISID, PHASE, and adverse event onset date (AEDT).

Scenario I – Merging Data sets using common BY variables

This scenario involves merging VISITS and AE using variables common to both data sets. To get the information related to patient visit with AE onset, VISITS has been merged with AE using PATNO, VISID, and PHASE that are available in both data sets. The methods compared here are as follows.

Simple Merge

The most common method of selecting observations from one data set based on observation from a second data set is by using the MERGE-expression. This requires that both data sets to be sorted. This also requires reading in every data record from both data sets even if one of them has only

ten or twenty records compared to millions in the second data set. The following code for MERGE-expression.

```
PROC SORT DATA=VISITS;
  BY PATNO VISID PHASE;
RUN;
PROC SORT DATA=AE;
  BY PATNO VISID PHASE;
RUN;
DATA AENEW;
   MERGE AE (in=in1) VISITS;
   BY PATNO VISID PHASE;
   IF in1;
RUN;
```

Merge with Index

Indexed SAS data sets can provide significant performance improvement for large data sets. For relatively small data sets, sequential processing is often just as efficient as indexed processing due to the overhead involved in indexing. When you have a data set that frequently changes, the overhead associated with rebuilding an index after each update can outweigh the processing advantages you gain from accessing the data through an index. The following segment of code illustrated the use of keyed access method.

```
PROC DATASETS LIBRARY=WORK;
     MODIFY VISITS,
     INDEX CREATE PATID=(PATNO VISID
PHASE) /UNIQUE;
RUN;
DATA AENEW;
   SET AE;
   SET VISITS KEY=PATID/UNIQUE;
   IF (_IORC_ NE  %SYSRC(_DSENOM)) THEN
OUTPUT;
   ELSE DELETE;
RUN;
```

SQL Procedure

PROC SQL offers a simpler coding in various situations such as combining more than two data sets, match on variables that are not exactly the same, calculate using intermediate results. In some merge situations, PROC SQL makes the code simpler faster than in DATA Step. The following is the code to accomplish what was done in the above categories.

```
PROC SQL;
   CREATE TABLE AENEW AS
   SELECT *
   FROM AE as AE, VISITS as VS
   WHERE AE.PATNO=VS.PATNO and
AE.VISID=VS.VISID and AE.PHASE=VS.PHASE;
QUIT;
```

Scenario II - Merging Data sets using not common BY variables

Assume that AE data set does not have variables VISID and PHASE. In this case, only PATNO is common to both data sets. Merging two data sets can only be done using PATNO and comparing the dates of present and previous visits with AE onset date. So the three techniques of merge described above was revised as follows.

Simple Merge

Since PATNO is the only common variable to merge, merge was accomplished by using PATNO and comparison of dates of previous and current visits with the AE onset date.

```
DATA AENEW;
 SET AE;
   DO I=1 TO NUMOBS;
      SET VISITS (RENAME=(PATNO=PID))
            NOBS=NUMOBS POINT=I;
      IF PATNO=PID AND AEDT>PVISDT
            AND AEDT<=VISDT THEN OUTPUT;
 END;
RUN;
```

Where, PVISDT is the date of previous visit for a patient.

Merging with Index

The method creates a new data set AEBYPID with counters for first as well as last record of each patient in AE, and then merges AE data set with VISITS using AEBYPID data set. The code is as follows:

```
PROC SORT DATA=AE OUT=AE;
  BY PATNO AEDT;
```

```
RUN;
PROC SORT DATA=VISITS OUT=VISITS;
  BY PATNO VISID PHASE;
RUN;
DATA FIRST LAST;
  SET AE;
  BY PATNO;
  IF FIRST.PATNO THEN DO;
     FIRSTOB=_N_; OUTPUT FIRST;
  END;
  ELSE IF LAST.PATNO THEN DO;
     LASTOB=_N_; OUTPUT FIRST;
  END;
RUN;
DATA PID_NUM (INDEX=(PATNO));
  MERGE FIRST (KEEP=PATNO FIRSTOB) LAST
(KEEP=PATNO LASTOB);
  BY PATNO;
  IF LASTOB=. THEN LASTOB=FIRSTOB;
  IF LASTOB < FIRSTOB THEN DELETE;
  IF FIRSTOB ^=. AND LASTOB ^=.;
RUN;

DATA AENEW;
  SET VISITS;
  SET PID_NUM KEY=PATNO/UNIQUE;
  IF (_IORC_ NE %SYSRC(_DSENOM)) THEN
OUTPUT;
  ELSE DELETE;
  DO I=FIRSTOB TO LASTOB;
     SET IN.AE POINT=I;
     IF AEDT > PVISDT AND AEDT <= VISDT
THEN OUTPUT;
  END;
RUN;
```

SQL Procedure

The Procedure is similar to the SQL procedure in the Scenario I with an additional condition of comparing AE onset date (AEDT) with visit date (VISDT) and previous visit date (PVISDT). The code is as follows:

```
PROC SQL;
   CREATE TABLE AENEW AS
   SELECT *
   FROM AE AS A, VISITS AS B
   WHERE A.PATNO=B.PATNO AND A.AEDT >=
PVISDT AND A.AEDT <= B.VISDT;
QUIT;
```

RESULTS

Consider the following two tables: VISITS and AE.

Figure 1. VISITS Data Set

PATNO	VISIT	PHASE	VISDT
1	1	A	1/05/96
1	1	B	2/04/96
1	1	C	3/06/96
1	2	A	4/05/96
1	2	B	5/05/96
1	3	A	6/04/96
1	3	B	7/04/96
1	3	C	8/03/96
1	3	D	9/02/96
2	1	A	1/10/96
2	1	B	2/09/96
2	2	A	3/11/96
2	2	B	4/10/96
2	2	C	5/10/96
2	3	A	6/09/96

Figure 2. AE Data Set

PATNO	AEDT
1	1/10/96
1	1/12/96
1	1/15/96
1	5/12/96
1	8/16/96
2	3/15/96
2	5/20/96

Visits are defined for each patient with the variable identifiers VISIT, PHASE, and VISDT. Each visit is further divided into different phases (A, B, and C). Each record has a visit date. Scenario I used PATNO, VISID, and PHASE to merge the VISITS with AE data set. However, for the purpose of demonstration, assuming that there are no VISID and PHASE variables in AE, scenario II uses the only PATNO as common variables to both data sets and merge the records by comparing AEDT with VISDT. Time trial was conducted on a VAX environment to compare the techniques in both the scenarios with 100, 500, 1000, 5000, 10000, and 20000 records

in AE data set. The results of CPU time are in Table 1, and 2.

Table 1. CPU time (sec.) of Processing - Scenario I

No. of Obs.	Simple Merge	Index Merge	SQL Procedure
100	2.60	2.33	0.35
500	2.83	2.41	0.37
1000	3.90	2.51	0.54
5000	5.25	2.92	1.04
10000	8.88	3.30	1.28
20000	10.35	3.81	2.39

Table 2. CPU Time (sec.) of Processing - Scenario II

No. of Obs.	Simple Merge	Index Merge	SQL Procedure
100	0.2	2.8	1.8
500	3.4	3.2	2.1
1000	4.9	3.3	2.9
5000	160.0	5.8	6.3
10000	1477.5	8.6	10.7
20000	7718.1	12.3	12.5

CONCLUSION

Based on the time trials, Merge with index is the better option when the data set is relatively constant. When data set changes frequently, due to indexing overhead, this may not be the viable option. Efficiency of index merge depends on the coding efficiency. Depending on the situation as described in the above two cases, there are different ways in using this method. In both scenarios, choice of simple merge is ruled since. PROC SQL is a good option in both scenarios with simplicity in coding. SQL also provides other potential advantages for communicating outside the SAS as SQL query is the standard method for specifying data requests from one database to another. Thus, the factors in deciding which method to use are: size of the data set, how static are the data set, the expertise of the programmer, whether you work with other data base systems.

AUTHORS AND CONTACT INFORMATION

Gajanan Bhat
PAREXEL International Corporation
195 West Street
Waltham, MA 01720
Gajanan.Bhat@PAREXEL.com

Raj Suligavi
4 C Solutions, Inc.
East Moline, IL 61244
Rajs@4cs.com