#### Paper 071-2009

# Choosing the Right Technique to Merge Large Data Sets Efficiently Qingfeng Liang, Community Care Behavioral Health Organization, Pittsburgh, PA

#### **ABSTRACT**

Merging two data sets horizontally is a routine data manipulation task that SAS® programmers perform almost daily. CPU time, I/O, and memory usage are factors that SAS® programmers need to consider before data sets with large volumes are merged.

This paper outlines different SAS® merging techniques for both a many-to-one match, and a many-to-many match. For a many-to-one match, MERGE, PROC SQL JOIN, FORMAT, ARRAY, and the HASH object were investigated, and their performances were benchmarked. For a many-to-many match, PROC SQL join and MATCH-CROSSING methods were discussed, and their performances were compared.

This paper shows that the HASH object is the best choice to perform a many-to-one match no matter single value or multiple values are retrieved from lookup table. ARRAY and FORMAT are also good choices with some restrictions. PROC SQL JOIN is the best choice for a many-to-many match.

#### INTRODUCTION

The volumes of data SAS® programmers deal with today are getting larger than several years ago, especially in health insurance area. When these large data sets are required to merge together, it adds significant complications to data manipulation process. Therefore, performing efficient merging becomes very critical, especially with an implementation of data warehouse.

Merging data sets is to combine two or more data sets horizontally by matching the keys from both data sets. This process is also referred as performing a table lookup, one data set serves as base table and the other is the lookup table. The records from the base table are matched to the records on a lookup table based on the common keys in both tables. There are four types of match merges: one-to-one match merge, one-to-many match merge, many-to-one match merge and many-to-many match merge.

The simplest case of a merge is a one-on-one match merge which for both the base table and lookup table each has the same unique identifier and the records are matched from both tables based on BY values of unique identifier. Many-to-one match merge is defined as records in a base table with duplicate BY values are matched to the unique BY values in lookup table. For example, if we want to look at utilization of members' outpatient service by demographic information, we need to merge outpatient services claims which a member could have multiple services during measurement period with member demographic information which each member is unique. One-to-many match merge is the same as many-to-one match merge if you treat the table with a unique key as lookup table rather than base table.

Many-to-many match merge refers to the BY value is not unique for both base table and lookup table. Many-to-many match could be problematic but it is very useful, especially in healthcare area. Whether insurance claims are eligible to be paid, claims have to be merged with an enrollment table using member ID as the key to see whether the service happened within the eligible coverage period. There are times when the member ID is not unique in both tables. A member could have more than one claim during a certain time period and he/she can also have more than one enrollment record in enrollment file. In this case, many-to-many match is very helpful and handy.

### **MANY-TO-ONE MATCH MERGE**

One-to-one match, one-to-many match and many-to-one match are dense matches and can be applied to the same merging technique. In this paper, many-to-one match merge will use to demonstrate different techniques and all the

rows from base table will be in the final combined data set and some rows from the look up table may not appear in the final table if they didn't match any rows from the base table.

Table 1 and Table 2 show the layouts of the data sets which are used for benchmarking merging techniques. Table 1 "Claims" has 9,654,283 rows with duplicate BY values (ID) and the file size is about 230 MB. Table 2 "Enrollment\_nodup" has 333,803 rows with unique BY value (ID) and the file size is about 13 MB.

**Table 1: Claims** 

ID	Start	End	Service Code	Provider
M1000001	2/1/2008	2/3/2008	Н0001	P1000002
M1000001	1/13/2008	1/19/2008	H0002	P1000003
M1000001	1/1/2008	1/3/2008	Н1010	P1000004
M1000001	4/13/2008	4/19/2008	T2000	P1000002
M10000008	1/1/2008	1/3/2008	Н0001	P1000003
M1000005	2/13/2008	3/19/2008	H0002	P1000004
M1000005	1/1/2008	1/3/2008	Н1010	P1000003
M10000009	2/10/2008	2/19/2008	T2000	P1000004
•••				

Table 2: Enrollment\_nodup

ID	Group_ID	
M1000001	ABC	
M1000002	BCD	
M1000003	CDE	
M1000004	DEF	
	•••	

The final combined table has 9,654,283 records. All many-to-one match merge techniques demonstrated in the paper will generate exactly the same result for the final combined table. CPU time, Input\Output operation and memory usage are used to measure the performance.

## **MANY-TO-ONE MATCH MERGE TECHNIQUES**

#### 1. PROC SQL JOIN

```
/* Left join will be used to take all records from the base table*/
proc sql noprint;
  create table sql_merge as
  select a.*, b.group_id
  from claims a left join enrollment_nodup b
  on a.id=b.id;
quit;
```

Codes are ANSI standard SQL syntax and easy to follow, but it can not be used in DATA step. Both data sets do not have to be sorted before join. Multiple values can be retrieved from lookup table.

# 2. MERGE

```
/* Sort base table and lookup table BY ID first */
proc sort data= enrollment_nodup;
  by id;
run;
proc sort data=claims;
  by id;
run;
```

```
data match_merge;
  merge claims(in=a) enrollment;
  by id;
  if a;
run;
```

Both base table and lookup table have to be sorted by BY values before they can be combined. Codes are straight forward. Multiple values can be returned from lookup table.

#### 3. ARRAY

```
/* Find the range for ID in order to define Array */
proc sql;
   select min(id) into: min_id from enrollment;
   select max(id) into: max_id from enrollment;
quit;

data array_merge;
   keep id group_id;
   /* Load lookup table into Array */
   if _N_=1 then do i=1 to numobs;
      set enrollment nobs=numobs;
      array groups{&min_id : &max_id} $ 25 _temporary_;
      groups{id}=group_id;
end;
set claims;
   group_id=groups{id};
run;
```

Codes are getting complex for merging using array in DATA step. First, the elements of array need to be determined using macro before the array is defined. Second, the lookup table has to be loaded into the array using DO loop. Third, the numeric key from base table is required to match the element from the array. Data sets can be unsorted when the array is applied to combine data sets. But the array can only be used in data step and only one value can be returned from lookup table. If more than one values are required from lookup table, multiple arrays have to be defined. Array can only store either numeric data items or character data items.

#### 4. FORMAT

First, the lookup table needs to be loaded in FORMAT and it could occupy a lot of resources if the number of records from the lookup table is huge. Second, the key in base table will be assigned to the created format using PUT statement. The data sets do not have to be sorted and FORMAT can be used in procedures. Codes are not difficult but you do need to know how to load a data set into FORMAT. FORMAT can only store one data item per key. Multiple formats need to be defined if more than one value will be retrieved from the lookup table. If that is the case, FORMAT is not an efficient way to do merge.

#### 5. HASH OBJECT

```
data hash merge(drop=pp);
  declare Hash group (dataset: "enrollment_nodup");
  pp=group.definekey('id');
  pp=group.definedata('group id');
  pp=group.definedone();
  do until(eof1);
    set enrollment_nodup end=eof1;
      pp=group.add();
  end;
  do until(eof2);
    set claims end=eof2;
      call missing(group_id);
      pp=group.find();
      if pp =0 then output;
  end;
run;
```

First, a HASH OBJECT is defined and named "group" and it is referred to data set "enrollment\_nodup". Second, "id" is specified as the key of the HASH OBJECT and "group\_id" is defined as data element. Third, the lookup table "enrollment\_nodup" will be loaded into Hash object. Fourth, group\_id is assigned to missing initially and the FIND method uses the value of the key (ID) from base to determine if there is a match to the key in Hash object. If a match is found, the data from two tables haven been combined and outputted to the data set "hash\_merge".

HASH OBJECT can be used only in DATA step and data sets do not have to be sorted. HASH OBJECT offers some of advantages and flexibilities to merge data sets. HASH OBJECT can use character variable, numeric variable, or a combination of variables as the key, and can store both multiple character variables and multiple numeric variables per key.

#### PERFORMANCE COMPARISON

CPU time, I/O operation and Memory usage were used to benchmark performance. CPU time is the amount of time the Central Processing Unit uses to perform merging task, and includes user CPU time and system CPU time. I/O operation measures the read and the write operations performed as data from storage to memory or from memory to storage. It measures the difference between real time and CPU time. Memory usage is the size of the work area required to hold data, buffers, etc.

The performances of five techniques outlined above have been benchmarked on a laptop with Window XP SP3 operating system, Intel Core2Duo 2.2 GHz CPU, 3 GB memory, 100 GB hard drive and SAS 9.1 installed. Five tests had been benchmarked on each technique and the average of five tests is used to measure the performance.

HASH OBJECT has the best performance on CPU time, the next to the best is ARRAY and FORMAT. HASH OBJECT also outperforms the other techniques on I/O operation by large margin. For memory usage, ARRAY needs much more memory than other techniques. FORMAT uses minimum amount of memory and HASH OBJECT is the second to the best. HASH OBJECT is clearly the winner on many-to-one match merge. ARRAY and FORMAT are also good choices with some restrictions. Only one value per key is stored in either single ARRAY or FORMAT, and ARRAY also uses a large amount of memory.

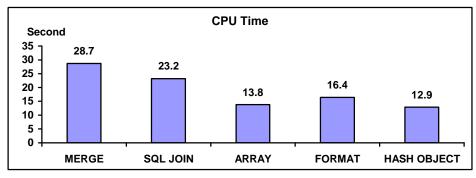


Figure 1: Performance of CPU time on many-to-one match merge

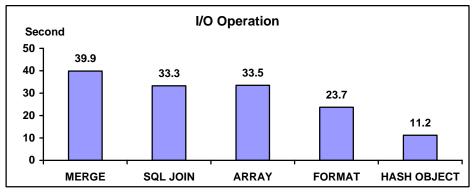


Figure 2: Performance of I/O operation time on many-to-one match merge

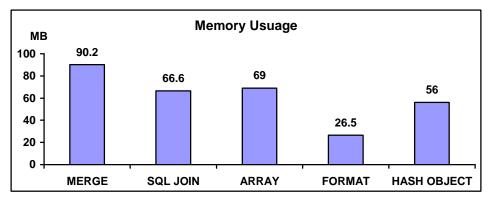


Figure 3: Performance of memory usage on many-to-one match merge

#### **OTHER TECHNIQUES**

There are two other techniques that are not discussed in the paper. These techniques can produce the same merging result on many-to-one match merge. One is to use IF-THEN/ELSE statement within DATA step. It is not efficient when there are so many conditions to be tested. It almost becomes impossible to list all conditions using IF-THEN/ELSE statements. The other technique is to use SET statement like following:

```
data set_merge(drop=mid);
  set claims;
  do i=1 to num;
    set enrollment_nodup(rename=(id=mid)) nobs=num point=i;
    if id=mid then output;
  end;
run;
```

It is useful when small size data sets are merged and its performance is not acceptable when combining large size data sets.

## **MANY-TO-MANY MATCH MERGE**

Tables used to demonstrate many-to-many match merge have the same layouts as the data sets used for many-to-one match merge, but Table 4 "Enrollment" has the repeated BY values "ID". Table 3 "Claims" has 9,654,283 rows

and Table 4 "Enrollment" has 333,803 rows. The final table for many-to-many match merge will has 35,718,496 rows.

Table 3: Claims

ID	Start	End	Service Code	Provider
M1000001	2/1/2008	2/3/2008	н0001	P1000002
M1000001	1/13/2008	1/19/2008	H0002	P1000003
M1000001	1/1/2008	1/3/2008	Н1010	P1000004
M1000001	4/13/2008	4/19/2008	T2000	P1000002
M10000008	1/1/2008	1/3/2008	Н0001	P1000003
M1000005	2/13/2008	3/19/2008	H0002	P1000004
M1000005	1/1/2008	1/3/2008	Н1010	P1000003
M10000009	2/10/2008	2/19/2008	T2000	P1000004
•••				

**Table 4: Enrollment** 

ID	Group_ID	
M1000001	ABC	
M1000001	BCD	
M1000001	CDE	
M1000002	DEF	
M10000002	KFD	

#### MANY-TO-MANY MATCH MERGE TECHNIQUES

#### 1. PROC SQL JOIN

```
proc sql noprint;
  create table sql_merge as
  select a.*, b.group_id
  from claims a, enrollment b
  where a.id=b.id;
quit;
```

The syntax for PROC SQL JOIN is standard and straight forward. Data sets do not have to be sorted or indexed. Multiple values can be returned from lookup table.

## 2. MATCH-CROSSING

MATCH-CROSSING is data step method and its details are explained by Aster & Seidman in their book "Professional SAS Programming Secrets".

```
/* sort base table and lookup table BY ID */
proc sort data=enrollment;
  by id;
run;

proc sort data=claims;
  by id;
run;

/* Create data set to hold the start and end point for each BY group */
data enroll_index;
  keep id _from _to;
```

```
retain _from;
  set enrollment(keep= id);
    by id;
    if first.id then _from=_N_;
    if last.id then do;
      _to=_N_;
      output;
     end;
run;
/* use the data set from previous step to combine base table and lookup table */
data crossing_merge(drop=_from _to);
  merge claims(in=a) enroll_index(in=b);
   by id;
    if a and b;
    do i=_from to _to;
      set enrollment point=i;
      output;
    end;
run;
```

First, base table and lookup table have to be sorted. Second, intermediate data set is created to hold beginning and end points for each BY group. Third, the base table is merged with the intermediate data set using BY values, for each BY value, pointer for the base table is to retrieve the values from the lookup table in DO loop.

## PERFORMANCE COMPARISON

The same laptop was used to benchmark the performances of the techniques outlined above. PROC SQL JOIN outperforms the MATCH-CROSSING method in all three performance measures. The syntax is also much simpler for PROC SQL JOIN. Therefore, PROC SQL JOIN is the best choice for performing many-to-many match merges.

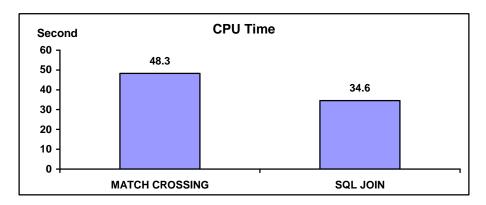


Figure 4: Performance of CPU time on many-to-many match merge

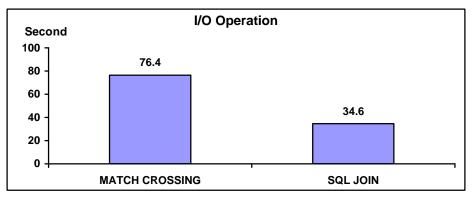


Figure 5: Performance of I/O operation on many-to-many match merge

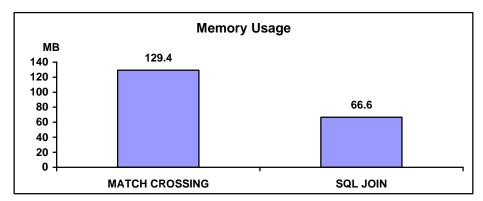


Figure 6: Performance of memory usage on many-to-many match merge

#### **OTHER TECHNIQUES**

A technique used SET statement with DO loop can do many-to-many match merge.

```
set_match(drop=mid);
  set claims;
  do i=1 to num;
    set enrollment(rename=(id=mid)) nobs=num point=i;
    if id=mid then output;
  end;
run;
```

The syntax is not as complex as MATCH-CROSSING method but the performance is too slow to be benchmarked when merging data set with large volumes.

#### CONCLUSION

Considering performances, the complexity of syntax on many-to-one match merge and flexibility, HASH OBJECT is highly recommended. It not only outperforms the other techniques but also has great flexibilities. It is time for SAS programmers to move from traditional MERGE and PROC SQL JOIN, ARRAY and FORMAT to HASH OBJECT. Although ARRAY and FORMAT have acceptable performances but there are some restrictions to apply them. For many-to-many match merge, PROC SQL JOIN is clearly the winner in terms of performance and simplicity.

# **REFERENCES**

Aster, Rick and Seidman, Rhena 1991, Professional SAS Programming Secrets, New York, McGraw Hill, Inc.

McAllaster, MAJ Doug 2002. DATA STEP vs PROC SQL in a Many-to-Many Match Merge

Stroupe, Jane and Jolley, Linda 2008. Using Table Lookup Techniques Efficiently. Proceeding of SAS Global Forum 2008

Loren, Judy 2008 How Do I Love Hash Tables? Let Me Count The Ways! Proceeding of NESUG 2008.

#### **ACKNOWLEDGEMENTS**

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

#### **CONTACT INFORMATION**

Your comments and questions are valued and encouraged. Contact the author at:

Name: Qingfeng Liang, Lead Programmer

Enterprise: Community Care Behavioral Health Organization

Address: Suite 700, Chatham One Center City, State ZIP: Pittsburgh, PA 15241

Work Phone: 412-402-8748

Fax: 412-454-2177 E-mail: liangQ@ccbh.com Web: www.ccbh.com