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

ABSTRACT
This paper outlines different SAS® merging techniques for both the many-to-one match merge and the many-to-many match merge. For the many-to-one match, PROC SQL JOIN, MERGE, FORMAT, ARRAY, and HASH OBJECT were investigated. For the many-to-many match, PROC SQL join and MATCH-CROSSING methods were discussed.

CPU time, I/O, and memory usage are factors that SAS® programmers need to consider whether data sets with large volumes are merged efficiently. These factors have been benchmarked for all match merging techniques mentioned above when the volumes of data are increasing.

This paper shows that the HASH object is the winner to perform a many-to-one match for large data sets in terms of efficiency. PROC SQL JOIN is the best choice for a many-to-many match when the size of data is getting larger.

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 value 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 lookup 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 85,370,162 rows with duplicate BY values (ID) and the file size is about 1.93 GB. Table 2 “Enrollment_nodup” has 1,069,517 rows with unique BY value (ID) and the file size is about 42 MB.
The source table “Claims” is used as base source to construct new source tables with different sizes. Different percentages of records (1%, 5%, 10%, 25%, 50%, 75%) were randomly extracted from and saved as new source tables. The sizes of each new source table are shown in Table 3. Each merging technique will run 10 times for each different size of source tables. CPU time, Input/Output operation and memory usage are used to measure the performance.

<table>
<thead>
<tr>
<th>Portion</th>
<th>Rows</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>819,487</td>
<td>19</td>
</tr>
<tr>
<td>5%</td>
<td>4,097,436</td>
<td>93</td>
</tr>
<tr>
<td>10%</td>
<td>8,553,016</td>
<td>198</td>
</tr>
<tr>
<td>25%</td>
<td>21,342,540</td>
<td>496</td>
</tr>
<tr>
<td>50%</td>
<td>42,685,081</td>
<td>992</td>
</tr>
<tr>
<td>75%</td>
<td>64,027,621</td>
<td>1,450</td>
</tr>
<tr>
<td>100%</td>
<td>85,370,162</td>
<td>1,930</td>
</tr>
</tbody>
</table>

### MANY-TO-ONE MATCH MERGE TECHNIQUES

**1. PROC SQL JOIN**

```sql
/* 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(in=b);
      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

/* create temporary data to load into FORMAT */
DATA temp_fmt;
   RETAIN fmtname "groupfmt";
   SET enrollment(KEEP=id group_id
       RENAME=(id=start group_id=label));
RUN;
PROC FORMAT LIBRARY=work CNTLIN=temp_fmt;
RUN;

DATA format_merge;
   SET claims;
   group_id=PUT(id, groupfmt.);
RUN;
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

```sas
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;
    ELSE 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 have 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 10 tests is used to measure the performance under different size of source tables.

HASH OBJECT has the best performance on CPU time; the next to the best is ARRAY and FORMAT. HASH OBJECT performs even better when the size of the base source table is increasing. HASH OBJECT also outperforms the other techniques on I/O operation when the size of base source table is getting larger and larger. For memory usage, HASH OBJECT uses more memory than other techniques. 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.
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 5 “Enrollment” has the repeated BY values “ID”. Source tables are the same as those for many-to-one merges. Table 4 “Enrollment” has 2,549,799 rows with 95 MB.

<table>
<thead>
<tr>
<th>ID</th>
<th>Start</th>
<th>End</th>
<th>Service Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>M10000001</td>
<td>2/1/2008</td>
<td>2/3/2008</td>
<td>H0001</td>
</tr>
<tr>
<td>M10000001</td>
<td>1/13/2008</td>
<td>1/19/2008</td>
<td>H0002</td>
</tr>
<tr>
<td>M10000001</td>
<td>1/1/2008</td>
<td>1/3/2008</td>
<td>H1010</td>
</tr>
<tr>
<td>M10000008</td>
<td>1/1/2008</td>
<td>1/3/2008</td>
<td>H0001</td>
</tr>
<tr>
<td>M10000005</td>
<td>2/13/2008</td>
<td>3/19/2008</td>
<td>H0002</td>
</tr>
<tr>
<td>M10000005</td>
<td>1/1/2008</td>
<td>1/3/2008</td>
<td>H1010</td>
</tr>
<tr>
<td>M10000009</td>
<td>2/10/2008</td>
<td>2/19/2008</td>
<td>T2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Group_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>M10000001</td>
<td>ABC</td>
</tr>
<tr>
<td>M10000001</td>
<td>BCD</td>
</tr>
<tr>
<td>M10000001</td>
<td>CDE</td>
</tr>
<tr>
<td>M10000002</td>
<td>DEF</td>
</tr>
<tr>
<td>M10000002</td>
<td>KFD</td>
</tr>
</tbody>
</table>

Source tables shown in Table 3 were used to test the performances of each many-to-many techniques. Each many-to-many merging technique will run 10 times for each different size of source tables. CPU time, Input/Output operation and memory usage were recorded to demonstrate the performance.

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 regarding the different sizes of source tables. 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.

```sas
DATA 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

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

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:

Qingfeng Liang, Lead Programmer
Community Care Behavioral Health Organization
Suite 700, Chatham One Center
Pittsburgh, PA 15241
Tel: 412-402-8748
Fax: 412-454-2177
E-mail: liangQ@ccbh.com
Web: www.ccbh.com