Paper AD09_05

Managing a Many to Many Merge using Formats, Direct Access, and Implied Hashing on Datasets Exceeding One Million Observations

Mark E. Asiala, Bureau of the Census, Washington, D.C.
John C. Gober Bureau of the Census, Washington, D.C.

ABSTRACT

Managing a many to many merge in SAS® and most other languages has always been the bane of programmers everywhere. The SAS merge statement can handle one-to-many and many-to-one merges, but was never designed to meet the intricacies and variations of a many-to-many merge. True, there are other options available to the programmer, like PROC SQL or arrays but these either require extensive disk resources or code that is often times confusing and elusive. The Bureau recently needed an application to apply missing geography to approximately 11 million records based on 27 million eligible ZIP codes. Since this was a time sensitive project, not only speed of execution but also available space was a critical consideration when design was considered. This paper is the collaboration of two programmers who independently came up with similar techniques.

INTRODUCTION

This works stems from an application to address an imputation problem. A large database of addresses contains almost 168 million records. For our work, it is necessary that each record have a complete set of geographic identifiers, which help determine where in a county a particular address is located. One source of updates to the database, however, does not provide these geographic identifiers and they must be determined based on matches to street name and address ranges. This match provides geocodes to many but not all of the update records. The records which do not receive geocodes, however, are still valid addresses but are missing the geographic identifiers below the county level. Our operation requires that these codes exist and thus we try to provide the best imputation of those codes based on the good geocoded records. In all, 11.3 million of these ungeocoded addresses exist and must be geocoded with valid geographic identifiers.

The method of imputation uses a donor pool of the good geocoded addresses to create lookup tables from them. Since we know that the county and ZIP Code fields are two reliable pieces of information present on both the ungeocoded and geocoded addresses, these fields provide the basis for the key on the lookup tables. For each combination of county and ZIP, however, multiple values for the geocode variables can exist and thus multiple records can be present for each combination of key variables on the lookup table. Since we can also have multiple ungeocoded records with the same combination of county and ZIP, merging a file of ungeocoded records to the lookup table creates a many-to-many merge.

In general, a many-to-many merge is either not well defined or produces a large Cartesian product of the records for each key. For each record, a many-to-many merge is actually a one-to-many situation which is the reason it is not well defined. Our process, however, is made to be well-defined by adding a test on a set of auxiliary variables that allows the selection of a unique match for each one-to-many situation. Through this test, the process is transformed into a logical many-to-one merge but is not recognized as such in most predefined procedures in SAS.

In our application, this test is performed by adding a cumulative percent distribution variable on the lookup table that reflects the percent of all geocoded addresses within the same key that is reflected up to that point much like the cumulative percent column in a PROC FREQ. Thus, this variable takes on increasing values from 0 to 1 where the last record will have a value of 1. Each ungeocoded record, on the other hand, will be assigned a random number between 0 and 1 inclusively. The merge is then performed by combining the set of ungeocoded records with the lookup table on the key variables and then selecting the first record on the lookup table whose cumulative distribution value is greater than or equal to the random number associated with the ungeocoded record. This method thus defines a many-to-one merge through the use of criteria on a set of auxiliary variables.

One additional wrinkle to the problem is that if a match from a ungeocoded record to the lookup table fails using county and ZIP+4 as the key, additional lookups are performed using the first eight digits of ZIP+4, the first seven digits, and so on until either a match is found or no match is found using county and the five digit ZIP Code. If that fails, then a match using county only is used via a county lookup table. While the number of unique keys is reduced

at each step, the number of records per key increases. This can impact the relative success of different techniques in terms of overall memory and disk usage.

WHY THE MERGE STATEMENT WILL NOT WORK

Consider the following records from the geography dataset missing the block information:

housenum	zip	block	rn
0001	22308		0.73902
0002	22308		0.27248
0003	22308		0.70953
0004	22308		0.31916

Now consider the following records from the block imputation dataset:

zip	block	_cprob_
22308	10	0.18358
22308	45	0.36317
22308	20	0.57940
22308	15	0.69553
22308	5	0.89117
22308	32	1.00000

In this scenario we wish to assign the first block code from the block dataset that has the first cprob greater than the rn from the geography table. Below was the undesired results using the merge statement with a by condition:

housenum	_zip_	rn	block	_cprob_
0001	22308	0.73902	10	0.18358
0002	22308	0.27248	45	0.36317
0003	22308	0.70953	20	0.57940
0004	22308	0.31916	15	0.69553
0004	22308	0.31916	5	0.89117
0004	22308	0.31916	32	1.00000

Because of the internal structure of the SAS merge statement each record of the geography and the block dataset was only read once. There was no Cartesian raking of information. Therefore it was not possible way to correctly incorporate the requirement ‘IF CPROB GE RN’ or any other needed statements into our code.

DESCRIPTION OF THE TECHNIQUE

The original technique used for this problem involved PROC SQL and applied a WHERE clause to the result of a Cartesian product of the full join. This method was very resource intensive and broke down for the lookup based on county only since the Cartesian product was quite large. A change to the procedure for this year, required that the process be run at the national level rather than the state level. This made the original technique, which worked at the state level for the smaller imputation tables, no longer viable for the current design which would create a Cartesian product of up to an estimated 2 billion observations for the county level imputation. No matter what resources we made available or SAS options we invoked or tweaked, the SAS process always failed to complete. Below is an example of our SQL code:

```
proc sql;
   create table work.matchzip9 as
   select a.*, b.cprob, b.block
   from work.geography as a,
            work.blockimpute as b
   where      a.zip    = b.zip        and
              b.cprob >= a.rn;
quit;

proc sort data=work.matchzip5 out=work.matchzip9;
   by zip housenum cprob;
run;
```

```
data work.matchzip9;
   set work.matchzip9;
   by zip housenum cprob;
   if first.housenum;
run;
```

Again this code gives the desired results but was too resource intensive for our system to handle.

Direct access merges were being used in several other parts of the process with good success both in terms of feasibility and efficiency and this technique was borrowed for this application.

Direct access can be performed using either indexes or direct observation number lookups. In this application, however, the use of an index is not well suited because of the many-to-many relationship. This leaves the use of direct access using the POINT= option. The authors each independently created a solution to the problem, which involves essentially the same technique where one can be viewed as a refinement of the other.

The base technique is to use a format that performs the role of mapping from a unique key to the first observation in the lookup table containing that key. Using the direct access method, an ungeocoded record can be read in and then pointed to the first record for the value of the key via the format if a match on that key can be found. If the first record meets the criteria then it is output with the imputed values for the geocodes obtained from the lookup table. If the record does not meet the criteria, then the read from the lookup table loops to the next record present on the table and continues until it finds a record that satisfies the criteria. By design, the last record for a unique key will always satisfy the criteria and thus a record can always be selected if the match on the key exists. Any ungeocoded records remaining are output to a separate dataset for lookups to tables using fewer digits of the ZIP Code.

The nature of this method changes as fewer and fewer digits of the ZIP Code are used. At the beginning, when all nine digits of the ZIP Code are used in conjunction with county, there are relatively few records on the lookup table for each key but there are many keys. This requires a larger format to be set up, but the search within the lookup table can be quicker since the sequential search is shorter. Later in the process when matches are made based on county and five digits of the ZIP Code or simply county, there are fewer keys but a much larger number of records on the lookup table per key. This makes the format smaller but the sequential search longer. This makes the simple implementation of this technique take progressively longer as the sequential search portion of the technique becomes less and less efficient.

```
*create format to generate starting points;
*for each county;
data zip9_format;
   length start  $14 label  8;
   fmtname ='zip9fmt';
   type='I';
   set blockimpute end=last;
   by zip9 fipst fcnty;
   start  = fipst || fcnty || zip9;
   label = _n_;
   if first.fcnty then
   do;
      n + 1;
      output;
   end;
   if last then
   do;
      start = 'other';
      label = 0;
      n + 1;
      output;
   end;
run;

proc format cntlin=zip9_format;
run;
```

```
*assign missing geography;
data sampwtw.match9(keep=…)
        nomatch9(keep=housenum fipst fcnty zip9 zip8 rn);
   length starta $14 zip8   $8 found 3;
   retain found 0 pt    0 retain seed  0;
   drop found;
   found = 0;
   set geography;
   by fipst;
   starta = fipst || fcnty || zip9;
   if first.fipst then seed = input('8' || fipst ||"&yyyy" || "12",9.);
   pt = input(starta,zip9fmt.);
   if pt ne 0 then
   do until (found = 1 );
      set blockimpute
             (keep=…rename=(…))
             point=pt;
        geosf   = '9';
        *assignment statements go here;
        …
        if cprob >= rn then
        do;
           found = 1;
           output sampwtw.&od18_impzip9;
        end;
       pt = pt + 1;
   end;
   else
   do;
       zip8 = substr(zip9,1,8);
       call ranuni(seed,rn);
      output nomatch9;
   end;
run;
*end nine digit assignment;
*This code repeats four more times for zip8, zip7, zip6, zip5;
```

The refinement to this technique is to keep these two components of the problem more in balance throughout the different lookup tables. As the number of records per key increases, it becomes advantageous to create a hashing within the key to reduce the number of records within each smallest interval of the hash. This adaptation was experimented with using hashes of size 2**n for increasing sizes of n. Trial and error determined which hash size was the most efficient. Looking at the final hash sizes, they are at values where the overall format size is on the same order of magnitude. A summary of the hashing parameters is given below:

Table1: Summary of Merging and Hash Usage

Level	#Lookup Table Obs	#Unique Keys in Lookup	Avg Max Seq Read w/o Hashing	# Hash Sub-divisons	#Hash Format Obs.	Avg Max Seq. Read with Hashing	Time to Create Format	Time to do the Merge	#Obs Read in	#Obs Matched
9	0.707	0.599	1.2	1	0.599	1.2	0:13	1:10	11.31	1.38
8	2.754	0.731	3.8	1	0.731	3.8	0:20	1:10	9.94	3.49
7	3.867	0.285	13.6	2	0.571	6.8	0:20	0:45	6.45	1.93
6	5.260	0.096	54.8	4	0.383	13.7	0:19	0:31	4.52	0.79
5	5.703	0.043	132.6	64	2.748	2.1	1:10	0:58	3.73	2.68
Cty	5.590	0.003	1,863.3	256	0.804	7.0	1:05	1:05	1.05	1.05

The choice of the number of hash subdivisions was more complicated than simply optimizing the average sequential read which would have suggested creating more subdivisions at an earlier point in the geocode imputation process. An additional factor was the number of records that matched to the lookup table. In the table above, it would seem that a much finer hash would be more efficient for the level 6 (match on 6 digits of ZIP code) based on the average maximum sequential read of 13.7. However, there is a much lower match rate for this level so much of that finer mesh would never be used and the additional time to set up the format is greater than the time saved in the merge

step. This is true since the non-matches to the keys in the format never actually read in observations from the lookup table so the impact of the large maximum sequential read is less than if we matched at a higher rate.

The code for the direct access merge with the use of hashing is given below. Even though the code looks more complicated than the previous examples, it has extensive macro usage to reduce replication of code and a macro to create a hashing table for an arbitrary number of hash subdivisions. The use of hashing reduced the run times by a factor of 5–10 or more for the 5-digit ZIP Code and County levels.

```
%*************************************************************************;
%*** WE FIRST DEFINE A GENERAL MACRO TO CREATE A FORMAT THAT PLACES A NUMBER ***;
%*** IN A GIVEN NTILE: E.G., ONE-HALF, ONE-QUARTER, ONE-SIXTY-FOURTH         ***;
%*************************************************************************;

%MACRO DEFINE_NTILES(N=,FMTNAME=);
%* N DEFINES THE NUMBER OF PARTITIONS WE DIVIDE THE INTERVAL 0-1 INTO *;
%* FMTNAME IS THE DESIRED NAME OF THE FORMAT *;

%LET INC = %SYSFUNC(ROUND(%SYSEVALF(1.00 / &N),0.00000001));

%*** WE TEST IF N IS GT 1 OTHERWISE WE DO NOT NEED THIS FORMAT ***;
%IF &N GT 1 %THEN
   %LET LEN = %LENGTH(TRIM(LEFT(&N)));
%ELSE
   %PUT ERROR: N MUST BE GREATER THAN 1;

DATA NTILES;
   IF _N_ EQ 1 THEN DO;
      RETAIN FMTNAME "&FMTNAME" TYPE 'N' FUZZ 0;
      LENGTH START END 8 SEXCL EEXCL $1 LABEL $&LEN;
      DO I=0 TO %EVAL(&N - 1);

         *** START GOES OUT IN MULTIPLES OF INCREMENTS;
         START = &INC * I;

         *** INCLUDE THE START POINT ONLY FOR FIRST INTERVAL;
         IF I EQ 0 THEN
            SEXCL = 'N';
         ELSE
            SEXCL = 'Y';

         *** ALWAYS INCLUDE THE END POINT OF THE INTERVAL;
         EEXCL = 'N';

         *** IF THIS IS THE FINAL INTERVAL MAKE SURE IT ENDS AT 1.00;
         IF I EQ %EVAL(&N - 1) THEN
            END = 1.00;
         ELSE
            END    = START + &INC;

         *** WE LABEL THE N-TILES;
         LABEL = PUT(I,Z&LEN..);
         OUTPUT;
         END;
      END;
   ELSE
      STOP;
   DROP I;
   FORMAT START END 10.8;
RUN;
```

```
PROC FORMAT FMTLIB CNTLIN=NTILES;
   SELECT &FMTNAME;
RUN;
%MEND DEFINE_NTILES;

%*****************************************;
%*** NEXT WE DEFINE THE PRINCIPLE MACRO ***;
%*****************************************;

%MACRO CREATE_ZIPX_UPDATE(X,N=,FMTNAME=);
%* X IS THE NUMBER OF DIGITS OF THE ZIP CODE TO USE IN THE IMPUTATION *;
%* N IS THE NUMBER OF PARTITIONS TO HASH INTO *;
%* FMTNAME IS THE FORMAT NAME OF THE NTILE FORMAT *;

%*** LEN IS THE LENGTH OF THE DECIMAL NUMBER OF PARTITIONS ***;
%IF &N GT 1 %THEN
   %LET LEN= %LENGTH(TRIM(LEFT(&N)));
%ELSE
   %LET LEN = 0;

%*** XM IS THE NUMBER OF ZIP CODE DIGITS MINUS ONE ***;
%LET XM = %EVAL(&X -1);

%*** DEFINE NTILES ***;
%IF &N GT 1 %THEN
   %DEFINE_NTILES(N=&N,FMTNAME=&FMTNAME);

%*** CREATE ZIPX_KEY FILE TO DEFINE THE FORMAT ***;
DATA ZIP&X._KEY;
   SET DBCOMP.ZIPBLK&X (KEEP=ZIP&X FIPST FCNTY CPROB) END=EOF;
   BY ZIP&X FIPST FCNTY;
   RETAIN FMTNAME "Z&X.KEY" TYPE 'I';
   LENGTH START $%EVAL(&X + 5 + &LEN);

   %IF &N EQ 1 %THEN %DO;
      %*** NO PARTITIONS JUST CREATE FORMAT BY THE KEY VARIABLES ***;
      IF FIRST.FCNTY THEN DO;
         LABEL = _N_;
         START = ZIP&X||FIPST||FCNTY;
         OUTPUT;
         END;
      IF EOF THEN DO;
         START  = 'OTHER';
         LABEL = -1;
         OUTPUT;
         END;
      %END;
   %ELSE %DO;
      %** NEED TO MAKE A FORMAT ENTRY FOR EACH KEY VARIABLE BY NO. OF PARTITIONS **;
      RETAIN Q;
      IF FIRST.FCNTY THEN DO;
         Q = 0; *** PREVIOUS QUARTILE ***;
         R = 0; *** CURRENT QUARTILE ***;
         LABEL = _N_;
         START = ZIP&X||FIPST||FCNTY||PUT(Q,Z&LEN..);
         OUTPUT;
         END;
      R = INPUT(PUT(CPROB,&FMTNAME..),&LEN..);
```

```
      *** EXECUTE IF WE ARE IN A NEW QUARTILE ***;
      IF R GT Q THEN DO;

         *** OUTPUT RECORDS TO COVER THE WHOLE INTERVAL ***;
         DO I=Q+1 TO R;
            LABEL = _N_;
            START = ZIP&X||FIPST||FCNTY||PUT(I,Z&LEN..);
            OUTPUT;
            END;

         *** SET PREVIOUS QUARTILE TO CURRENT QUARTILE ***;
         Q = R;
         END;

      IF EOF THEN DO;
         *** ADD NON-MATCH VALUE ***;
         START  = 'OTHER';
         LABEL = -1;
         OUTPUT;
         END;
      %END;
   KEEP FMTNAME TYPE START LABEL;
RUN;

PROC FORMAT CNTLIN=ZIP&X._KEY;
RUN;

%*** IMPZIP_X IS THE OUTPUT DATASET OF MATCHES TO THE LOOKUP TABLE ***;
%*** ZIPREC_XM IS THE OUTPUT DATASET OF NON-MATCHES TO THE LOOKUP TABLE ***;

DATA   DBCOMP.IMPZIP&X      (KEEP=MAFID FIPST FCNTY BST BCNTY
                               BTRAT BLOCK BLKS1 BLKS2 GEOSF)
       DBCOMP.ZIPREC&XM     (KEEP=MAFID ZIP&X ZIP&XM FIPST FCNTY RN);

   *** DEFINE THE SEED FOR X-1 ***;
   IF FIRST.FIPST THEN
      SEED = INPUT("&XM"||FIPST||"&YYYY.12",9.);
   RETAIN SEED;

   *** WE NEED TO OBS STATEMENTS DEPENDING ON WHETHER HASHING IS USED ***;
   %IF &N EQ 1 %THEN %DO;
      OBS= INPUT(ZIP&X||FIPST||FCNTY,Z&X.KEY.);
      %END;
   %ELSE %DO;
      OBS= INPUT(ZIP&X||FIPST||FCNTY||PUT(RN,&FMTNAME..),Z&X.KEY.);
      %END;

   IF OBS EQ -1 THEN DO;               *** THIS IS THE PATH FOR A NON-MATCH ***;
      *** DEFINE THE RN FOR X-1 ***;
      CALL RANUNI(SEED, RN);

      *** DEFINE ZIP<X-1> ***;
      LENGTH ZIP&XM $&XM;
      ZIP&XM = SUBSTR(ZIP&X,1,&XM);

      *** OUTPUT THE RECORD FOR THE NEXT PASS **;
      OUTPUT DBCOMP.ZIPREC&XM;
      END;
```

```
   ELSE DO;                              *** THIS IS THE PATH FOR A MATCH ***;
      DO UNTIL (CPROB GE RN);
         SET DBCOMP.ZIPBLK&X (DROP=PROB) POINT=OBS;
         OBS + 1;
         END;
      GEOSF = "&X";

      *** OUTPUT RECORD WITH GEOCODES ADDED ***;
      OUTPUT DBCOMP.IMPZIP&X;
      END;
RUN;
%MEND CREATE_ZIPX_UPDATE;

%CREATE_ZIPX_UPDATE(X=9,N=1);                  %* NO WITHIN KEY HASHING *;
%CREATE_ZIPX_UPDATE(X=8,N=1);                  %* NO WITHIN KEY HASHING *;
%CREATE_ZIPX_UPDATE(X=7,N=2, FMTNAME=HALF);    %* SPLIT THE RANGE INTO TWO PIECES *;
%CREATE_ZIPX_UPDATE(X=6,N=4, FMTNAME=QUART);   %* SPLIT THE RANGE INTO FOUR PIECES *;
%CREATE_ZIPX_UPDATE(X=5,N=64,FMTNAME=FCUBE);   %* SPLIT THE RANGE INTO 64 PIECES *;

%*** LATER WE DO THE COUNTY LEVEL IMPUTATION IN THE SAME MANNER WITH N=256 ***;
```

CONCLUSION

Many-to-many merges with auxiliary selection criteria pose significant complications to the base SAS procedures both in terms of functionality and resources. In this case, custom procedures within SAS helped each author devise a technique that was both functional and an efficient use of resources. The use of direct access and formats provided a workable solution while the use of hashing greatly added to its efficiency. Important considerations in the use of hash tables include the size of the table and its ability to be stored in memory, the maximum size of sequential reads that is possible from different hash size options, and the number of anticipated matches between the two datasets. It was clear from our work that all three items must be considered when trying to tune this technique.

The use of custom techniques such as these also appears to continue to be beneficial even as more robust procedures are being added to SAS. While SAS offers a vast array of predefined procedures and statements, its ability to allow solutions “outside-the-box” makes it an excellent software tool.

WEB PRESENCE

http://www.census.gov

TRADEMARKS

CONTACT INFORMATION

Mark E. Asiala, Decennial Statistical Studies Division
U.S. Bureau of the Census
4700 Silver Hill RD
Washington DC 20233
301-763-3605

John Charles Gober, American Community Survey Office
U.S. Bureau of the Census
4700 Silver Hill RD
Washington DC 20233
301-763-5964