Methods For Combining SAS®
Data Sets Of Varying Sizes: Which Method To Use When?
Hugh Roghmann, Acumen-LLC, Burlingame, CA

ABSTRACT
When combining two SAS data sets based on values of a single key variable, there are several coding methods to choose from. Some of these include a match merge, a PROC SQL join, key-indexing, and hashing. This paper differentiates the relative efficiency of these four techniques in one common situation with regard to one changing parameter: the proportion of key values in a smaller data set that have matching values in a larger 'lookup' data set. The data-combining situation I will concentrate on involves the combining of two SAS data sets that share a one-to-one relationship between key values. I find that the index-based methods of PROC SQL and key-indexing can be significantly faster than the sequential match merge method when the smaller data set is less than about 7% the sample of the larger lookup data set. For samples greater than about 7%, the match merge usually runs significantly faster than all other methods. I also find that performance of index-based methods varies dramatically according to whether SAS has recently worked with the two input data sets recently, which appears to cause data sets to become cached into memory.

INTRODUCTION
The comparison of these four data combining methods was done by running many trials of different SAS DATA steps and PROC SQL steps and then comparing the mean elapsed times of the different types of trials. Other than the type of programming method used for the combining of the data sets, the only other varying factor will be the relative sizes of the smaller and larger data sets in terms of the number of observations in each. The real times elapsed during the various steps will be graphically compared for varying relative sizes of the two input data sets. Graphs are located at the end of the paper in the Appendix.

TERMINOLOGY AND CONTEXT
The smaller data set will be referred to as either the Primary or Sample data set. The larger data set will be referred to either as the Lookup or Master data set. The programming exercise is to add a variable to the Sample/Primary data set that exists in the larger Master/Lookup data set. In general, the Primary data set is the one we are using for our analysis, while the Master data set holds information on everyone in our population. In the example used in this paper, I use the Social Security Number (SSN) of the people in the Primary data set to match to the correct individuals in the Master data set so that I can obtain the dates of birth of the people in my Primary sample. The Master data set contains demographic information such as date of birth, sex, race, etc. The question is whether we should program this task using a match-merge, Proc SQL, key-indexing techniques, or perhaps the new hashing features found in version 9 of the SAS System. Our main performance criterion is efficiency, measured by the elapsed time of completion of the DATA step or PROC SQL step.

THE METHODS
The following SAS code was run for each of the following methods:

1. match merge

   data temp;
   merge d.sample&sample._not_indexed (in=a)
       d.lookup_population_not_indexed (in=b); by SSN;
   if (a and b) then output;
   run;

2. SQL join

   proc sql noprint;
   create table temp as
   select a.SSN, b.bdate
   from d.sample&sample._indexed as a,
       d.lookup_population_indexed as b
   where a.SSN=b.SSN;


3. key indexing

```sas
data temp;
  set d.sample&sample._not_indexed;
  set d.lookup_population_indexed key=SSN;
  [error checking code not shown here for brevity]
run;
```

4. hashing, as implemented in SAS V9

```sas
data temp (keep=SSN bdate);
  length SSN $9;
  declare AssociativeArray hh (dataset: "d.sample&sample._not_indexed", hashexp:16 );
  rc = hh.DefineKey ('SSN');
  rc = hh.DefineDone ();
  **read lookup;
  do until (eof2);
    set d.lookup_population_not_indexed ( keep=SSN bdate) end=eof2;
    rc = hh.find();
    if rc=0 then output;
  end;
  stop;
run;
```

The data sets used follow the naming scheme shown below in Table 1, so it is clear to the reader when a data set used has an index and when it does not.

I ran all trials on a dual-Pentium III 1400Mhz computer with 3.8GB of RAM running Microsoft Windows XP Professional and SAS Software Version 9.00. Perl was used both to run numerous SAS programs in batch mode, and to extract reported times from the SAS log files.

---

**DATA CONSTRUCTION**

For the many trials I run in this exercise, five directories of parallel same-named data sets were created. The Master data sets contain only randomly generated Social Security numbers and birth dates for large numbers of fictitious people. In the five directories, the Master data sets have 1, 10, 30, 50 and 100 million observations of unique artificial SSNs. Also, in each of these five directories, there are 13 Primary data sets containing SSNs only that are .01%, .05%, 1%, 3%, 5%, 7%, 10%, 20%, 30%, 40%, 60%, 80%, and 100% random samples of the Master data set. Random sampling was done using the SAS RANUNI function. All data sets are ordered by SSN. By construction, all SSNs in the Primary data sets will always be a random subset of those found in the Master data set. For each of the 14 data sets in each of the five directories, two separate copies are maintained, one with a simple index based on SSN and one without. Trials were run using the data sets in only one of the five directories at a time.

For each respective population size in each directory, I used the following data set naming scheme:

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup_population_indexed.sas7bdat</td>
<td>Master data set with an index</td>
</tr>
<tr>
<td>Lookup_population_not_indexed.sas7bdat</td>
<td>Master data set without an index</td>
</tr>
<tr>
<td>Sample0001_indexed.sas7bdat</td>
<td>Primary .01% sub-sample data set with an index</td>
</tr>
<tr>
<td>Sample0001_not_indexed.sas7bdat</td>
<td>Primary .01% sub-sample data set without an index</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Sample100_indexed.sas7bdat</td>
<td>Primary 100% sub-sample data set with an index</td>
</tr>
<tr>
<td>Sample100_not_indexed.sas7bdat</td>
<td>Primary 100% sub-sample data set without an index</td>
</tr>
</tbody>
</table>
CONCEPTUAL MOTIVATION

I have chosen these four methods for comparison because I am interested in comparing sequential and direct-access methods for combining data sets. The match merge is the only example of a sequential data combining method of the four, while PROC SQL join, key-indexing, and hashing are in my mind all forms of direct-addressing methods. I have read several times the general statement that direct access methods are faster when a very small data set is being combined with a much larger data set, while sequential access methods should be superior when the input data sets are more comparable in size. I have always been curious to test this statement and find out what particular sizes of input data sets run faster using which methods and by how much.

To try to conceptualize the difference between a direct-addressing method and a sequential method, I try to think about what work the SAS System must do to find a matching key value in the Master data set, given a particular key value in the smaller Primary data set. With a sequential method (match-merge), successive observations in Master are read until a matching value is found. If the number of observations in Master is large relative to the number of observations in Primary, many false comparisons are done for every successful key-value comparison. Sequential methods go to each observation of the each data set checking each value to see if there is a match, reading all observations in both Primary and Master.

Direct access methods, on the other hand, do not process each observation sequentially but, given a key value in Primary, use some addressing scheme to go directly to the observation in Master with matching key value. Direct addressing might use indexes, arrays, or hashes in order to know the address or observation number of a particular value. Maintaining and using this addressing structure entails some additional processing work that sequential methods do not perform, but this addressing structure enables the SAS System to go directly to a set of values in Master given a key value in Primary. Key-indexing and PROC SQL use indices for their direct-addressing structures, and require an index to be already created prior to the DATA step doing the combining. While not using indices, I also think of hashing as a direct-address lookup method, but one that uses memory to create the addressing structure ‘on the fly’ in the same DATA step of your SAS program.

So, if there are a very small number of observations in Primary, a direct-addressing method should be faster than a sequential method because it only has to use the index on Master to look up a small number of matching key values. Using an addressing structure adds time which makes accessing an observation in Master take longer than accessing a single observation without an index, but it wastes no time going to read observations of non-matching key values. A match-merge, on the other hand, would take more time doing mostly false comparisons reading all the observations in Master for the small number of matches it finds. However, as the Primary data set contains a larger and larger number of observations, the sequential method becomes relatively faster as the time using an index to match more values increases and the wasted time of doing false comparisons in the match merge becomes smaller.

RESULTS

Graph 1A shows ‘Second and Later’ run times of the various methods as series, with elapsed time on the Y-axis, and the sample size on the X-axis. Second run times are recorded after the same two input data sets are combined with the same method in consecutive iteration. If the step is iterated 10 times, only the mean of the last 9 runs is reported. Thus, when a 1% sample Primary data set (N=50,000) was merged with a Master data set of 50 million records in size, the match merge DATA step took on average 36 seconds to complete. The same thing was accomplished in 14 and 17 seconds on average using Proc SQL and key-indexing, respectively. At a sample size of about 7%, the sequential and direct-access lookup techniques had about the same performance.

Graph 1B shows ‘First Run’ times for the same combinations of input data sets as in Graph1A. In this case, the match merge ran significantly faster than the index-based methods for nearly all sample sizes. First runs are produced when SAS has not recently worked with either of the two input data sets. To ensure this is the case, a reboot was done prior to running these steps. It is my understanding that the difference between real times achieved in Graphs 1A and 1B is due to the Input/Output (I/O) time it takes SAS to read the data sets off disk into memory. Since real elapsed time can be decomposed into I/O and CPU time, significantly lower reported CPU times than real times indicate a significant I/O component was involved. This can be confirmed by examining CPU times rather than real times for first runs, as is done in Graph 2A. Comparison of Graphs 1A and 2A show that CPU and Real times are much more similar for second runs than first runs.
In order to limit the number of graphs and results reported in this paper, all results reported here concentrate on the 50 million observation size Lookup data set. Findings were essentially similar for the 1,10, 30, and 100 million observation size lookup data sets.

My overall findings are summarized in the following 2 points:

1. When the sample size to be matched in the lookup data set is smaller than approximately 5-7%, the indexed-based methods were capable of running in significantly faster times than the match-merge. The smaller the sample size, the relatively faster the index-based method performed. On the other hand, for sample sizes increasingly larger than 7%, the match merge increasingly ran faster than the index-based methods.

2. Real run times varied drastically depending on whether SAS had recently worked with the input data sets, especially when an index-based method was being used. This performance chasm can likely be attributed to whether the input data sets have been cached into memory, or whether SAS was reading these data sets off disk for the first time.

CONCLUSIONS

If you regularly combine relatively small data sets with much larger ones, and each input data set is can be ordered by unique values of a key identifier, you may want to consider creating an index and using Proc SQL or key-indexing rather than running a match merge. Otherwise, a simple match merge performs very well. I found that a sample size of 7% was an approximate break-even point when the Master data set had 50 million observations.

However, serious consideration should be given as to whether the step you are planning is likely to perform as a ‘second or later’ run, or a ‘first’ run. If your planned step makes use of an index, I urge you to think about whether the step you are about to run will effectively be a first or second run to SAS. You may be able to order the steps in your programming so that your steps that access a very large data set are consecutive, rather than interrupted by the reading of other large data sets, or the performing other memory-intensive steps in between accesses of the single large data set. You also may want to consider running a relatively quick data _null_ step to read indexed input data sets before attempting to combine them. In the limited number of trials I was able to run, I observed that the combined elapsed times of preliminary data _null_ steps and a ‘first trial’ data combining step were much less than ‘first trial’ data combining step running alone.

I must strongly emphasize that my results are merely one set of results for a specific data combining situation, and should not be inferred to apply to different data combining situations. Data sets were already in SSN sort order, and there are already copies with and without indexes. None of my trials included the time taken to create indices or sort data sets. This was done intentionally in an effort to compare the best possible performances these methods could offer, with the assumption that once you decide upon the appropriate method, you can decide whether to maintain an index on your Master and Primary data sets, and that you maintain most of your data sets ordered by key personal identifiers. I also assume that you frequently combine data sets of certain set sizes. I expect that choosing a situation with previously sorted data sets confers the greatest advantage on the match merge method, and is perhaps is most unfair to the V9 hashing method, which requires neither pre-sorting nor the pre-existence of an index.

I also found that a match merge does not perform as well when the Primary and Master datasets are indexed. And Proc SQL performs very poorly if there isn’t an index. Key-indexing is not even possible if no index exists on the Master data set.

REFERENCES


ACKNOWLEDGMENTS
I would like to thank West Addison for his helpful insights into these trials, and for valuable comments on earlier drafts of this paper.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:
Hugh Roghmann
Acumen-LLC
1040 Rollins Rd. Suite 110
Burlingame, CA
Work Phone: 650-558-3980 x23
Fax:650-558-0180
Email: hroghmann@acumen-llc.com
Web: http://www.acumen-llc.com/

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 trademarks of their respective companies.
Graph 1A: Real Times by Sample Size
50,000,000 Observations in Lookup Data Set
Second and Later Trials Only

Times reported are means of 9 trials for samples \( \leq 10\% \), 2 trials for \( > 10\% \).
Second and later trials occur after SAS has recently read both input data sets.

Graph 1B: Real Times by Sample Size
50,000,000 Observations in Lookup Data Set
First Trials Only

Times reported are means of 4 trials.
First trials are first reads of unique SAS data sets following a reboot.
Graph 2A: CPU Times by Sample Size
50,000,000 Observations in Lookup Data Set
First Trials Only

Times reported are means of 4 trials.
First trials are first reads of unique SAS data sets following a reboot.