Using SAS Indexes with Large Databases

Alex Vinokurov, Omnicare Clinical Research, King of Prussia, PA
Lawrence Helbers, Omnicare Clinical Research, King of Prussia, PA

ABSTRACT

This paper presents a primer on using SAS® indexes to directly access large databases in a variety of sort orders. It discusses applicable uses of an index, creation of indexes and the SAS code needed to utilize indexes in selecting data subsets, merging, and SAS procedures. The techniques discussed produced substantial performance improvements in prescription database analyses with over 68 million observations.

INTRODUCTION

Have you had to develop an application where you want to access just the records you want quickly from a much larger database? SAS indexes make that possible. This paper is designed to show you how and when to use indexes with SAS. These methods can have great advantages over traditional sequential access of data.

The paper demonstrates some techniques not documented in SAS-online and hardcopy manuals, but which have been described at the SAS website and recent User-Group presentations. The purpose of the paper is to help spread the word. We relate our own experiences on MS Windows® and SUN Unix® platforms using SAS v6.12 and SAS v8.02.

WHAT'S AN INDEX?

An index is a means of selecting data from a database using a key (index) and direct access input/output operations. SAS supports simple (single variable) and composite (multiple variable) indexes and can store multiple indexes together in its supporting (.si2 in SAS for Windows) index file.

A sales database would be a good example, where different applications may need to access the database by Customer, Salesperson, Invoice Number, Entity, Region, Product, and Date. In our prescription database application we need to access the Rx data by patient, NDC_Number (drug packaging id), drug compound code, and year/month. Different indexes can be created to access the data in each desired order without additional sorting, once the index is created. All indexes would reside in a single .si2 file. The year/month index is a composite index with multiple variables, while the others are simple indexes.

SAS uses the index to find where the record you want is stored and then uses direct access input/output to access that record in the database (the .sd2 file in SAS for Windows). Some overhead is involved in finding the records you want and then accessing them directly, so there must be other savings to make this procedure efficient. For example, you may only need to access a small percentage of the records or you are avoiding a time-consuming sort.

USES FOR INDEXES

Indexes are suggested for three types of processing:

- WHERE Statements in PROC Steps
- Match Merging
- BY Statement Processing in DATA/PROC Steps

SAS automatically uses an index to efficiently select cases if you use a WHERE clause which includes an indexed variable in a PROC Step. In the Sales Data Base example, you could generate a report for a particular salesperson, product or invoice without having to read through the entire database sequentially.

Match Merging selects the desired cases which are specified in one dataset that are also in the larger database, using indexes. It is this application that has yielded the greatest productivity gains for us. A typical application involves a few steps:

1. Find all residents with 1+ Rx for a particular drug
2. Find all Rx's for those residents identified in Step 1 for longitudinal analysis
3. Find all these resident's health assessments in an indexed patient assessment file

All three steps involve Match Merging using indexes, in these cases one-to-many merges. Indexing saves time in each step and turned a file development task which previously took several hours into one which now takes minutes. The challenge was to learn the syntax to do the merge, which was not automatic and not intuitive.

SAS also automatically uses indexes in BY processing in DATA Steps and PROC Steps. BY processing requires the data to be sorted on input. Using an index delivers the data to the DATA step or procedure in the desired order. BY processing is used when first. or last. coding is needed or when sub-analysis by group is needed and too many classes exist for available memory to use the CLASS statement that is available for several procedures.

BY processing using the entire database will only be efficient if it avoids costly sorts because several different sorts will be desired by different applications. The overhead needed to retrieve each record directly can be quite substantial. We would not recommend using this application for match merging because it is, in our experience, more time consuming than not using indexes.

CREATING AN INDEX

Indexes can be created using PROC DATASETS or in the DATA Step which creates a file. Each method is demonstrated below.

PROC DATASETS

PROC DATASETS modifies the characteristics of members of an existing SAS library. To add an index, specify the library, the member to modify and the indexes to create. For example:

```
proc datasets library=rx_data
   modify master;
     index create res_id;
     index create compound;
     index create ndc_numb;
     index create yearmo=(year month);
   contents data=master;
run;
quit;
```

The code above modifies library rx_data, member master, and creates simple indexes for patient (res_id), drug compound, ndc number, and a compound index (yearmo) using the variables year and month. The CONTENTS statement provides a revised contents for the master dataset, which will now include a list of the dataset's indexes. PROC DATASETS is an interactive procedure, so make sure to use a QUIT statement to terminate its operation.

CREATING AN INDEX IN THE DATA STEP

Use the (index=) dataset option to create an index in the DATA Step. The syntax is index=([index name]=(var1 var2 …)). In the case of a composite index you list the variables included in the composite for index name. You may leave out the =(var1 var2) for simple indexes. The code below

```
data rx_data.master
       (index=(res_id
               compound
               ndc_numb
               yearmo=(year month)));
    set test;
run;
proc contents;
run;
```

creates the same four indexes as in the PROC DATASETS example. A simple index is created for res_id, drug compound, and NDC number. A compound index called yearmo is created for year and month). The output of the PROC CONTENTS will report:

```
-----Alphabetic List of Indexes and Attributes-----

  #   Index           Var1      Var2
 ----------------------------------------------------------
  1   COMPOUND
  2   NDC_NUMB
  3   RES_ID
  4   YEARMO          YEAR      MONTH
```

Note the difference between the simple and compound indexes. The compound index has entries under Var1 and Var 2 headings to report what variables compose the index.

SAS APPLICATION CODE USING INDEXES

In some cases SAS uses indexed datasets transparently, so no additional coding is necessary. In other cases the code is a bit counterintuitive and hard to remember. We will review that syntax, but suggest these techniques will be adopted more readily by developing general use macros. In yet other cases, SAS may determine that using the index will be counterproductive and chose to ignore the index in processing.

We consider the coding for SAS procedures, match merging, and BY processing below.

SAS PROCEDURES- WHERE CLAUSE

SAS automatically uses an index to efficiently select cases if you use a WHERE clause which includes an index variable. In the Sales Data Base example, a report for a particular salesperson could be coded:

```
proc means data=salesmstr n mean sum std;
   where salesper='SmithL';
   var booked shipped paid;
run;
```

SAS will find the cases for this salesperson using direct access to avoid having to read the entire database.

MATCH MERGING

Our decision to write this paper stemmed from the difficulty we had figuring out how to do a match merge with indexes. Don't use MERGE/BY processing. It will work but be inefficient. The correct techniques are documented, if you know where to look. Better yet, create simple to use macros that use the correct techniques.

In our applications we need to search a large database to find records which match a short list of unique drugs or residents. We do inner merges where only records in both files are kept. We'll call the large file the master file and the smaller file the select file. We think doing a match merge should be as simple as the macro call below:

```
%one2many(master=lib2.DISP0101,
          select=min_res(keep=res_id mds_rsid),
          keyvar=res_id,
          mrgout=disp0101
          filter=);
```

Do a one-to-many match merge by simply specifying the master, select, and output files, the key variable to use, and an optional line to filter cases further by non-indexed values (the macro is included on the conference CD). Unfortunately, it is not that easy yet, so we'll go through the syntax you need to know for the one-to-one, one-to-many, and many-to-many match merge cases below.

One-to-One Merging

You can create an output data set containing the records in the select data set that are also contained in the master dataset using the SET/SET syntax shown below.

```
data Output-SAS-data;
  set Select-SAS-data;
  set Master-SAS-data key=key_name/unique;
      select(_iorc_);
        when (%sysrc(_sok)) do;
          output;
        end;
        when (%sysrc(_Dsenom)) do;
          _error_=0;
        end;
        otherwise;
      end;
run;
```

One-to-Many Merging

The one-to-many merge is similar to the one-to-one, but it uses a DO UNTIL to loop through all the records with the match key. The syntax is:

```
data Output-SAS-data;
     set Select-SAS-data;
     do until(_iorc_=%sysrc(_Dsenom));
       set Master-SAS-data key= key_name;
         select(_iorc_);
           when (%sysrc(_sok)) do;
             output;
           end;
           when (%sysrc(_Dsenom)) do;
             _error_=0;
           end;
           otherwise;
         end;
     end;
   run;
```

Like the one-to-one merge, it has some syntax that isn't likely to be familiar to beginners such as _iorc_, %sysrc, _sok, we will discuss that later.

Many-To-Many Merging

Many-to-many merging requires SQL, just as it does when doing such a match merge without indexes. The syntax is:

```
proc sql;
   create table output as
   select b.*
   from slct_tab as a, master as b
   where a.mds_rsid=b.mds_rsid;
run;
```

Table output is created merging all cases in table slct_tab to matching cases in table master. PROC SQL uses the indexes transparently as the WHERE clause does in other SAS procedure calls.

COMMON MNEMONICS

This section explains some of the esoteric code above. The %SYSRC macro function is used to test whether a given return code is a specific known condition. The user passes a 'mnemonic' condition name to the %SYSRC macro. The returned code can then be used in program branching. Example syntax is:

```
_iorc_=%sysrc(_Dsenom)
```

The _iorc_ variable is the return code test whether the _Dsenom condition was the result of an access. Some mnemonics are for KEY= accesses, others are for BY accesses and some are for both. Common mnemonics are:

_SOK	Specifies that the desired observation was located. Mnemonic can be used with KEY= or BY access. In the match merging code above a match is found and written to output.
_DSENOM	Specifies that the master data set does not contain the observation. Used with KEY= access. Can be used to raise error condition or branch to write an unmatched id file.
_DSENMR	Specifies that the transaction data set observation does not exist in the master data set. Used with BY statement access. Uses are the same as for _DSENOM above.
_DSEMTR	Specifies that multiple transaction data set observations with a given BY value do not exist in the master data set. Used with BY statement access. Uses are the same as for _DSENOM above.

The %SYSRC code is just used to ascertain whether a match was found or not and then branch accordingly in the program logic.

OTHER DATA STEP USES

Under the right circumstances it can also be efficient to use BY processing with different indexes to avoid doing time consuming sorts. For example, suppose you have a large sales database and have to produce two reports on a monthly basis that are detailed-nested summaries. For example, a sales report by business entity, brand, region, product code and one by ingredient, warehouse, product. If there are too many categories to accumulate summaries in memory you might try to sort the needed variables from the master database, which is a time consuming step. If the master database was indexed by these composite keys, then data could be accessed sequentially for 'BY' processing without a sort. Beware that the extra overhead in resolving the index and doing the random read could make this approach undesirable. Our advice is to test first.

The code illustrates two 'BY' procedures without an intervening sort because the master data is indexed both ways:

```
proc means data=master;
  by entity brand region product;
  var sales;
run;
proc means data=master;
  by ingredient warehouse product ;
  var sales;
run;
```

UPDATING IN PLACE

For the record, indexed files can be updated in place with updates to the indexes using the MODIFY command. Master files don't have to be completely rewritten to add or update records. That subject is broad and beyond the scope of what we planned here. The reader is referred to a good article on the SAS.COM web site by Moorman and Warner cited in our references for more information.

Note that otherwise when a new dataset is created from an indexed dataset using the SET command that the new file is not indexed, unless specified with INDEX= or modified later with PROC DATASETS.

RESTRICTIONS ON USE

Indexes can't be used everywhere in SAS. Here is a short list of restrictions:

- must be direct access datasets – no tapes
- can't be used on compressed datasets
- Not available for SAS Transport datasets.

PERFORMANCE COMPARISONS

Indexing can produce benefits of shorter processing times but can involve extra costs to such as additional storage space for the index file and additional coding. In our application, the benefits clearly outweighed the costs.

ADDITIONAL DISK SPACE

Adding 4 simple indexes to our databases added 9% to our disk space requirements on both the PC and UNIX platforms. This amounts to over 2 gigabytes, relatively inexpensive with today's hardware prices.

TIME REQUIRED TO CREATE INDEXES

Time required to create a set of indexes isn't substantial. Adding 4 indexes to a database with 1.89 million records took:

30 minutes	PC Platform
2.5 minutes	UNIX platform.

A good candidate application for indexing then, is one that doesn't need to be recreated too frequently and one where the master database is used often to produce many different displays.

Test	Time to Complete (secs) w/o Indexes	with Indexes	Savings (%)	Observations in Master File
Where Processing in PROC step				
PC SAS v6.12	215.6	10.8	95.0%	1,892,000
UNIX SAS v8.2	33.4	23.2	30.5%	1,892,000
Match Merging				
One-to-One				
PC SAS v6.12	344.8	48.6	85.9%	1,482,000
UNIX SAS v8.2	28.1	2.9	89.8%	1,482,000
One-to-Many				
PC SAS v6.12	233.1	13.0	94.4%	1,892,000
UNIX SAS v8.2	32.8	5.1	84.4%	1,892,000

TIME SAVED WHEN USING INDEXES

The figure above reports comparison 'real times' required to complete selected tasks on the PC and UNIX platforms. We report results for WHERE Processing and Match Merging (one-to-one and one-to-many) for some large databases with 1.5-1.9 million records. Our actual database is nearly 30 times larger than this 'test' file. We don't have a real world example of BY processing, but the remaining cases presented are representative of our work with the prescription database.

The WHERE-processing example selected less than .1 % of all records for summary. The savings using indexes were 95% for the PC platform and 31% for the UNIX platform. The total savings are much larger on the PC platform as the UNIX is much faster to complete a given task.

Match Merging is used widely in our early stages of project database development from the larger master database. It includes some of the longest runs we encounter, over 10 hours per run for the entire database. In the test presented here 20 thousand selected residents were match merged against 1.5 million master records in a demographic database. Savings on both the PC and UNIX platforms were over 85%.The non-indexed case counts both the time to sort the reference file and the BY match merge.

The one-to-many case selected all prescriptions for a few drugs from the master file. The non-indexed case was a simple SET DATA step with an IF to specify the list of drugs desired. The PC-based test saved over 94% and the UNIX-based test saved over 84%.

Applications that produce many sets of displays, using a small percentage of all observations each time, are good candidates for indexing. The percent savings are substantial. Production savings are accrued several fold and the one-time cost to create the indexes is spread over many runs.

CONCLUSION

Under the right circumstances SAS indexes can provide dramatic improvements in performance. With the right training or macro infrastructure they can also be easy to create and use. We hope this paper has helped explain when and how to do so.

REFERENCES

Horowitz, Lisa, *Compressing and Indexing Large SAS Data Sets: The Hows, Whens, and Wherefores,* SAS Institute, A presentation to PhilaSUG, November 1995.

Karp, Andrew H., *Indexing and Compressing SAS Data Sets: How, Why and Why Not,* Sierra Information Services Inc, PowerPoint Presentation.

Moorman, D.J. and Warner, D., *Updating Data Using the MODIFY Statement and the KEY Option*, SAS Service and Support Library, 2001.

SAS Institute Inc, *SAS Language Reference Version 6 First Edition,* CARY, NC, SAS Institute Inc, pp. 212-221.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

Alex Vinokurov
Omnicare Clinical Research
630 Allendale Road
King of Prussia, PA 19406
Work Phone: 484-679-3148
Fax: 484-679-2509

Email: Alexander.Vinokurov@omnicarecr.com
Lawrence Helbers
Omnicare Clinical Research
630 Allendale Road
King of Prussia, PA 19406
Work Phone: 484-679-2421
Fax: 484-679-2509
Email: Larry.Helbers@omnicarecr.com