

Faster Data Extraction with Your Own Clustered Index

Zhuan (John) Xu

Big Creek Software, L.L.C, Polk City, Iowa

Abstract

Large data warehouses and decision support systems are becoming more and more popular. They make fast data access more important than ever. With clustered index, data is sorted in clustered index order and thus provide very fast data access. Although SAS does not provide built in clustered index, user may create his own clustered index in a small SAS data set and use data set options (FIRSTOBS= and OBS=) to apply such index to extract data. This method provides a way to extract data faster than SAS index.

A pair of SAS macro programs are presented to greatly simplify the process. They hide details from user yet flexible enough for user to perform other data processing.

Introduction

Data warehouses and decision support systems are becoming more popular than ever. They almost always contain large amount of data. Although the CPU of the computers are much faster and its storage capacity is much larger today, the access time and the transfer rate of the storage device, such as a hard disk, see only relatively small improvement. A data warehouse are often used in many applications. Some use all observations in a large data set. Some require only a small subset. For example, user may need some detail data to conduct further analysis. In this case, indexed data set provides fast access. With clustered index, data is sorted in clustered index order and thus provide very fast data access.

Daniel Dower introduced in his SUGI20 paper¹, titled 'Access Data Faster Than a SAS Index', the concept of zoned-index point data set. It is essentially a clustered index. He demonstrated that such index is faster than a SAS index.

This paper introduces a pair of SAS macro programs for creating and using clustered index. One will sort and produce the index data set. The other will apply the index data set, together with SAS data set options (FIRSTOBS= and OBS=) to subset the data set. This pair of SAS Macro programs makes utilizing clustered index a simple task.

SAS Data Extraction Process

Data extraction can be performed in many different ways with large SAS data sets. There are many SAS papers on this topic. For example, Anthony M. Dymond compared four different data extraction methods; Merge, SQL, key, and format. The result was presented in his SUGI 23 paper². The paper has a reference which listed many works in this area.

I will not get into details of their work. Instead, I will do a quick review of data extraction with SAS data step.

Suppose there is a SAS data set SAMPLE with the data listed below:

OBS	Name	Task	Hours
1	Anthony	Analysis	6
2	Jack	Programming	8
3	Jack	Conference	12
4	Joe	Analysis	15
5	Joe	Programming	14
6	Mark	Conference	12
7	Martin	Vocation	8

To extract data with Name='Jack' from it, you can code a simple SAS program:

```
data SUBSET;
  set SAMPLE;
  if NAME='JACK';
run;
```

When SAS processes the data step, it actually go over each observation and check the value of the variable NAME. That is, it will read in all observations but keep only those with NAME='Jack'. For small data set, it is not a problem. But for a large SAS data set, it is not very efficient. You may index the NAME to speed it up. However, I will present another way to do it.

Recall that there are two data set options FIRSTOBS= and OBS=. You may have used OBS= to debug you SAS program. For example, the

```
data SUBSET;
  set SAMPLE (OBS=3);
  if NAME='JACK';
run;
```

With the OBS=3 option, SAS will only use the first 3 observations and thus a good way to debug your program. Now look at the following revised code:

```
data SUBSET;
  set SAMPLE (FIRSTOBS=1 OBS=3);
run;
```

It will produce the same result as the original program. The advantage is that it no longer go over the whole data set and thus run much faster for a large data set.

if you need to pick two person, you may have the program in original style as:

```
data SUBSET;
  set SAMPLE;
  if NAME in ('JACK' 'Mark');
run;
```

Or in new style as:

```
data SUBSET;
  set SAMPLE (FIRSTOBS=2 OBS=3)
    SAMPLE (FIRSTOBS=6 OBS=6);
run;
```

Note that SAMPLE data set occur twice in the above program.

Clustered Index

Now the question is, how we know the range for FIRSTOBS= and OBS=? The answer is that we don't know. However, we can keep track of it by creating a separate SAS data set as an 'index' file. For the SAMPLE data set, we may create following file:

OBS	Name	OBSF	OBSL
1	Anthony	1	1
2	Jack	2	3
3	Joe	4	5
4	Mark	6	6
5	Martin	7	7

Then for NAME='Jack', we can easily get FIRSTOBS from OBSF (1) and OBS from OBSL (3). Obviously, to reduce the size of the index file, the data should be sorted by the selection variable. In the sense, the above index data set is a clustered index for variable NAME.

Daniel Dower presented sample SAS programs in his SUGI20 paper¹ for using such method. For practical application, it is desirable to hide the details as the coding each extraction program can be time consuming. The two macro programs listed in the Appendix were coded to accomplish this. They are also generalized to handle multiple data sets and allow additional selection on non index variables. Additional SAS statement may also inserted whenever necessary

To apply such index method, you will first need to prepare the data set by sorting and then creating index data set. After that, you can extract data faster than SAS index!

Creating Clustered Index Data Set

SAS macro program **clstrin** is used to create the index data set. It has the following parameters:

- dt_in input data set.
- dt_index output index data set.
- vari variable to be indexed
- dt_out output data set. If dt_out is blank, the output use same name as input file.

As a example, the following code will prepare the SAMPLE data set:

```
libname test 'c:\sasdata';
%clstrin(dt_in =test.SAMPLE,
        dt_index=test.SAMPLE_i,
        vari =NAME,
        out_dt =test.SAMPLE_O);
run;
```

Note that the all data sets are in the directory c:\sasdata; the index data set is SAMPLE_i, which is created for variable NAME; The output is in SAMPLE_O.

Using Clustered Index

With index data set ready, you can apply it to extract data with SAS macro program **clstrout**. This program has the following parameters:

- dt_ins input data sets.
- dt_index input index data sets in the order of dt_in.
- dt_out output data set.
- vari indexed variable to be subsetted.
- select SAS statement for selection on vari.
- keeplist input list.
- st1-st9 additional SAS statement. such as keep, drop list or any valid SAS statement.

The following is a sample program for extracting data:

```
libname test 'c:\sasdata';
%clstrout(dt_ins =test.a_o1 test.a_o2,
        dt_index=test.a_i1 test.a_i2,
        dt_out =test.subset,
        vari =name,
        select =where name='Jack',
        keeplist=name task hours,
        st1 =if hours>10);
```

Note that it uses multiple input data sets and their corresponding index data sets. It also listed variables to be kept and has additional selection specified in st1.

The parameter select can be any valid SAS statement performed on the index variable NAME, such as:

- if name='Jack'
- if name ne 'Jack'
- where name in ('Jack', 'Joe')
- if NAME=: 'J'

Inside the macro program, the index data set is first subsetted. Then a data step is coded to combine the continuous range. For example, with selection of 'Jack' and 'Joe', the ranges are 2-3 and 4-5. They are combined into 2-5. This reduces the number of pairs of FIRSTOBS and OBS significantly. These ranges then are moved into SAS macro variables and used as parameters in the data set options FIRSTOBS= and OBS=.

Performance comparison and Discussion

Applying the macro presented, You can extract data from large data set very fast. The actual speed up is almost inversely proportional to the percentage of the data extracted. It is much faster than processing data without index. My personal experience with health insurance claim history data with about 10 million observation on HP UNIX is very positive. Without the clustered index, it would take about 40 minutes. With such index, it typically took only 1 minute.

Daniel Dower's test¹ shows such index is actually faster than SAS native index by 8 - 25%.

Note that the data set can be used as usual without the index data set. This means that you can still create a native index on other variable(s). Another thing worth mention is that the SAS data set used as index file is much smaller than the native SAS index file.

On the other hand, since the data set options FIRSTOBS= and OBS= are used in the process, the SAS data must stored in uncompressed format. Also, when the index data set is created with macro `clstrin`, it sort data set and then run a data step. As sorting uses a lot disk space and time, it may not be effective for frequently updated data base due to the one time overhead.

Conclusion

Using clustered index can be very efficient in extracting data from large SAS data set. The two SAS macro programs presented in the paper greatly simplified its application. It is recommended to evaluate the cost and benefit before using such index. The best candidates are static, frequently accessed large SAS sets.

References

- ¹ Daniel Dower, (1995), Accessing Data Faster Than a SAS Index, *Proceedings of the 20th Annual SAS Users Group International Conference*, Orlando, Florida
- ² Anthony M. Dymond, (1998), Data Extraction Methods in Medium and Large Databases, *Proceedings of the 23rd Annual SAS Users Group International Conference*, Nashville, Tennessee.

Acknowledgments

SAS and SAS/BASE are registered trademark of SAS Institute Inc. in the USA and other countries. Other brand and product names are registered trademark of their respective companies.

The author may be contacted at:

Zhuan (John) Xu
Big Creek Software L.L.C.
400 53rd PL
W. Des Moines, Iowa 50266
(515) 984-6243
johnxu@bigcreek.com

Appendix A: SAS Macro Program for Creating Clustered Index

```
%macro clstrin(dt_in=,dt_index=,dt_out=,vari=);
/*****
/* Macro program for creating custom clustered index dataset */
/* (dt_index) on variable (vari). dt_out is same as dt_in except */
/* it is sorted by (vari). */
/* Program: clstrout */
/* Function: Extract data using custom clustered index */
/* By: Zhuan (John) Xu */
/* Big Creek Software, Polk City, Iowa */
/*****/
```

```

/*          johnxu@bigcreek.com    http://www.bigcreek.com    */
/* USAGE:                                         */
/*      dt_in    --- input data sets              */
/*      dt_index --- output index data set        */
/*      dt_out   --- output data set. if dt_out is blank, the */
/*                  output use same name as input file.      */
/*      vari     --- variable to be indexed        */
/* NOTE: use macro clstrout to extract dataset    */
/*****
%if &dt_out= %then
  %do;
    proc sort data=&dt_in;
      by &vari;
    data &dt_index;
      set &dt_in (keep=&vari);
      by &vari;
      retain obsf obsl 0;
      if first.&vari then obsf =_n_;
      if last.&vari then do;
        obsl=_n_;
        output;
      end;
      keep &vari obsf obsl;
    run;
  %end;
%else
  %do;
    proc sort data=&dt_in out=&dt_out;
      by &vari;
    data &dt_index;
      set &dt_out (keep=&vari);
      by &vari;
      retain obsf obsl 0;
      if first.&vari then obsf =_n_;
      if last.&vari then do;
        obsl=_n_;
        output;
      end;
      keep &vari obsf obsl;
    run;
  %end;
%mend clstrin;

```

Appendix B: SAS Macro Program for Using Clustered Index

```

%macro clstrout(dt_ins=,dt_index=,dt_out=,vari=,select=,keeplist=,
               st1=,st2=,st3=,st4=,st5=,st6=,st7=,st8=,st9=);
/*****
/* Macro program for subsetting using index dataset (dt_index) */
/* It is necessary that input dataset(s) (dt_in) was created by */
/* clstrin.                                                    */
/*
/* Program: clstrout
/* Function: Extract data using custom clustered index
/* By: Zhuan (John) Xu
/*      Big Creek Software, Polk City, Iowa
/*      johnxu@bigcreek.com    http://www.bigcreek.com
/* USAGE:
/*      dt_ins    --- input data sets
/*      dt_index  --- input index data sets in the order of dt_in*/
/*      dt_out   --- output data set
/*      vari     --- indexed variable to be subsetting
/*      select   --- SAS statement for selection on vari
/*      keeplist --- input list.
/*      st1-st9  --- additional sas statement. such as keep,
/*                  drop etc;
/* NOTE:
/*****
/* create macro vars for input dataset and its index dataset
data _null_;
  words="&dt_ins";
  do i=1 to 100;
    word=scan(words,i,' ');

```

```

    if word ne '' then
        call symput('dt' || compress(i),word);
    else do;
        call symput('dtc',compress(i-1));
        i=101;
    end;
end;
run;

/* create macro vars for the index dataset(s) of the input */
/* dataset(s) */
data _null_;
    words="&dt_index";
    do i=1 to 100;
        word=scan(words,i,' ');
        if word ne '' then
            call symput('dti' || compress(i),word);
        else do;
            call symput('dtic',compress(i-1));
            i=101;
        end;
    end;
run;

/* Verify count */
data _null_;
    if &dtc ne &dtic then
        error "Number of data sets: &dtc does not match number of Index data sets: &dtic";
run;

/* Subset Index */
%do k=1 %to &dtic;
    data _sel;
        set &dti&k;
        &select;
    run;
    /* Create macro vars for the ranges */
    data _null_;
        set _sel end=enddata;
        retain obsf_k obsl_k 0;          /* range of obs kept */
        if _n=1 then do;
            obsf_k=obsf;
            obsl_k=obsl;
        end;
        else if obsf-obsl_k>1 then do; /* descrt, so create macro vars */
            i+1;
            call symput('f' || compress(&k) || compress(i),compress(obsf_k));
            call symput('l' || compress(&k) || compress(i),compress(obsl_k));
            obsf_k=obsf;
            obsl_k=obsl;
        end;
        else
            obsl_k=obsl;                  /* update end of range var: obsl_k */

        if enddata then do;              /* last obs, create macro vars */
            i+1;
            call symput('f' || compress(&k) || compress(i),compress(obsf_k));
            call symput('l' || compress(&k) || compress(i),compress(obsl_k));
            call symput('c' || compress(&k), compress(i));
        end;
    run;
%end;

data &dt_out;
    set
    %do k=1 %to &dtc;
        %do j=1 %to &c&k;
            &dt&k (firstobs=&f&k&j obs=&l&k&j
                %if &keeplist= %then
                )
            %else
                keep=&keeplist)
        %end;
    %end;
run;

```