Paper 234-2011

Subsetting Large Data Sets Created Via the Complex “WHERE” Clause

Mikhail Gruzdev and Allen Blackburn
United States Bureau of Census, Washington, D.C.

ABSTRACT

SAS® offers several methods to enable fast data extraction from large data sets. The discussion, here, is of techniques offered within base SAS as extraction options for the minimization of overall processing times. Our paper points rather positively to the efficient usage of the complex “WHERE” clause coupled with its complex “OR” logic options. Our paper then goes through several iterations of advanced processing techniques including “SORT” and “INDEX” then favorably compounds total processing efficiency by use of the “POINT=” option with utilization of the SAS observation number. By combining these techniques, access speed is increased, at minimum, a factor of five or six times. A rudimentary table explains overall efficiency.

INTRODUCTION

The “WHERE” clause is a powerful and useful tool for rapid creation of data set subsets, but further usage of SORT options and INDEX and POINT options allow for even more increased efficiencies in the overall subset process. Vast terabyte amounts of data exist in repositories across many organizations which require storage, control, and virtually instantaneous access. But what access techniques are best? This paper arrives at usage of several techniques allowing increased speed and performance, roughly judged by actual access times.

ILLUSTRATION

A simple “WHERE” clause and complex “WHERE” clause illustration:

Simple “WHERE” clause – WHERE (ID = ‘AAA’ and TEMPER >= 110);

Complex “WHERE” clause – WHERE (ID = ‘AAA’ and TEMPER >= 110) **OR** (ID = ‘BBB’ and TEMPER <= 85)

Our test SAS data set is created with 10,000,000 records and 12 variables. Two additional data sets will be outputted for use later. One data set is sorted without index and another data set is sorted with index.

```
libname in ‘c:\test’;

data m_nosort_noindex(drop=i j);
length product $ 10 state $ 2 flame $ 3 ctry $4 x 4 y 4 z 4 w 4 f 4  g 4 date 8 ;
do i=1 to 10000000;
product='OIL_TYPE_'; state = 'C'; flame = 'YE'; ctry = 125;
    do j=1 to 5;
    product=trim(left(product))||byte(int(65+26*ranuni(0)));
    ctry=trim(left(ctry))||byte(int(65+16*ranuni(0)));
    state=trim(left(state))||byte(int(65+26*ranuni(0)));
    flame=trim(left(flame))||byte(int(65+2*ranuni(0)));
    end;
    x=2*ranuni(0); y=10*rannor(0); z=2*ranuni(0); w=10*rannor(0);f=2*ranuni(0);
    g=10*rannor(0);
    date=today()+int(y);
if flame = 'YEA' then flame = 'YES';
else flame = 'NO';
output; end;
run;

data in.m_nosort_noindex;
set m_nosort_noindex;
rec_num = _n_;  /* Observation Number can be used later with POINT option */
run;
```

SOME TESTS

Subsetting with no SORT, and no INDEX, and complex “WHERE” clause.

```
data subset_nosort_noindex;
set in.m_nosort_noindex;
where product ='OIL_TYPE_L’ and flame = 'NO' and date = 18464 and state = 'CE' and
ctry='125I'
OR product ='OIL_TYPE_S’ and flame = 'YES' and date = 18464 and state = 'CE' and ctry
='125I';
run;
```

```
NOTE: There were 83 observations read from the data set IN.M_NOSORT_NOINDEX.
      WHERE ((product='OIL_TYPE_L') and (flame='NO') and (date=18464) and (state='CE')
      and(ctry='125I')) or ((product='OIL_TYPE_S') and (flame='YES') and (date=18464)
      and(state='CE') and (ctry='125I'));
NOTE: The data set WORK.SUBSET_NOSORT_NOINDEX has 83 observations and 12 variables.
NOTE: DATA statement used (Total process time):
      real time           1:19.46
      cpu time            7.17 seconds
```

Subsetting with SORT, and no INDEX, and complex “WHERE” clause.

```
data subset_sort_noindex;
set in.m_sort_noindex;
where product ='OIL_TYPE_L' and flame = 'NO' and date = 18464 and state = 'CE' and
ctry='125I'
OR product ='OIL_TYPE_S' and flame = 'YES' and date = 18464 and state = 'CE' and ctry
='125I';
run;
```

```
NOTE: There were 83 observations read from the data set IN.M_SORT_NOINDEX.
      WHERE ((product='OIL_TYPE_L') and (flame='NO') and (date=18464) and (state='CE')
      and(ctry='125I')) or ((product='OIL_TYPE_S') and (flame='YES') and (date=18464)
      and(state='CE') and (ctry='125I'));
NOTE: The data set WORK.SUBSET_SORT_NOINDEX has 83 observations and 12 variables.
NOTE: DATA statement used (Total process time):
      real time           1:19.20
      cpu time            9.04 seconds
```

Subsetting with SORT, and INDEX, and complex “WHERE” clause.

```
data subset_sort_index;
set in.m_sort_index;
where product ='OIL_TYPE_L' and flame = 'NO' and date = 18464 and state = 'CE' and
ctry ='125I'
OR product ='OIL_TYPE_S' and flame = 'YES' and date = 18464 and state = 'CE' and ctry
= '125I';
run;
```

```
NOTE: There were 83 observations read from the data set IN.M_SORT_INDEX.
      WHERE ((product='OIL_TYPE_L') and (flame='NO') and (date=18464) and (state='CE')
      and(ctry='125I')) OR ((product='OIL_TYPE_S') and (flame='YES') and (date=18464)
      and(state='CE') and (ctry='125I'));
NOTE: The data set WORK.SUBSET_SORT_INDEX has 83 observations and 12 variables.
NOTE: DATA statement used (Total process time):
      real time           1:28.35
      cpu time            6.61 seconds
```

Subsetting in SAS using SORT, and INDEX and complex “WHERE” clause provides seemingly equal access times:

SUBSET CONDITIONS	Real Times	Searched Records
No SORT, and no INDEX, and complex WHERE	1:19.46	10,000,000
SORT, and no INDEX, and complex WHERE	1:19.20	10,000,000
SORT, and INDEX, and complex WHERE	1:28.35	10,000,000

PROBLEM

And just why are the access times so seemingly close after sorting and indexing? The answer is simply that the usage of the complex “OR” logic forces our program to evaluate “correctness” of selection with SAS Boolean logic, and still executes that logic for every observation. The Boolean “IF” logic test has the intrinsic quality that it is “True” if only one, or both of logic conditions is met And the statement is “False” only if both logic tests are “False”. When a check of a first condition is “False”, then a second logic check must still be made. Also note with interest that “WHERE conditions are applied **before** the data enters the input buffer while IF conditions are applied **after** data enter the Program Data Vector, PDV”[1].

SOLUTION

Build a better mousetrap solution, and use a 3-step subset process whereby the observation number is retained in an intermediate data set and used with a POINT option to increase efficiency of large data set creation. Judicious use of the “WHERE Clause” allows for faster intermediate file creation.

Step #1

Subset 10,000,000 record data set by both common variables using KEEP statements. By applying KEEP or DROP statements to all procedures that reference data sets/view “overall performance savings of 10-20% CPU time are not uncommon”[2]. Our intermediate output data set will have just two variables, product and its record number or observation number. Our input data set has been both sorted and indexed.

```
data temp_subset1;
set in.m_sort_index(keep=product rec_num);
where product in ('OIL_TYPE_L', 'OIL_TYPE_S');
run;

NOTE: There were 767717 observations read from the data set IN.M_SORT_INDEX.
      WHERE product in ('OIL_TYPE_L', 'OIL_TYPE_S');
NOTE: The data set WORK.TEMP_SUBSET1 has 767717 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           9.57 seconds
      cpu time            2.10 seconds
```

Step #2

Subset large data set again by reading from large 10,000,000-record SAS data set and output of complex “WHERE” generated data set from previous step #1 and use the POINT option. SAS will read only records from large SAS data set with equal record number (observation number) into data set temp_subset2 from our step #1. Our output again now contains all twelve original data set variables. Now the search is limited to only a portion of the original data set or 767,717 observations.

---

[1] Gupta.

[2] Wilcox.

```
data temp_subset2;
set temp_subset1;
set in.m_SORT_index point = rec_num;
output;
run;
```

```
NOTE: The variable rec_num exists on an input data set, but was also specified in an
I/O statement option.  The variable will not be included on any output data set.
NOTE: There were 767717 observations read from the data set WORK.TEMP_SUBSET1.
NOTE: The data set WORK.TEMP_SUBSET2 has 767717 observations and 11 variables.
NOTE: DATA statement used (Total process time):
      real time           7.81 seconds
      cpu time            1.12 seconds
```

Step #3

Now, again create a subset data set from previous step #2 output using the complex "WHERE" clause. And the new double subset is accomplished by viewing only 767717 observations. It is worth noting that data is not in the PDV yet, but still in the data buffer.

```
data double_subset;
set temp_subset2;
where product ='OIL_TYPE_L' and flame = 'NO' and date = 18464 and state = 'CE' and
ctry='125I'
OR product ='OIL_TYPE_S' and flame = 'YES' and date = 18464 and state = 'CE' and ctry
='125I';
run;
```

```
NOTE: There were 83 observations read from the data set WORK.TEMP_SUBSET2.
      WHERE ((product='OIL_TYPE_L') and (flame='NO') and (date=18464) and (state='CE')
      and (ctry='125I')) or ((product='OIL_TYPE_S') and (flame='YES') and (date=18464)
      and(state='CE') and (ctry='125I'));
NOTE: The data set WORK.DOUBLE_SUBSET has 83 observations and 11 variables.
NOTE: DATA statement used (Total process time):
      real time           0.25 seconds
      cpu time            0.22 seconds
```

Results of 3-Step Process:

STEPS	SUBSET CONDITIONS	Real Times	Searched Records O/P
Step 1	SORT, INDEX, simple WHERE clause + KEEP product and record_number	0:09.57	10,000,000
Step 2	SORT, INDEX, + create POINT +KEEP all variables	0:07.81	767,717
Step 3	SORT, INDEX, complex WHERE clause	0:00.25	83
	Subset Total Time	0:17.63	

SUMMARY

Our paper identifies that a combination of techniques rather than single usage of a complex WHERE clause might be the better method to access vast amounts of data. SORT and INDEX are two proper techniques but can be enhanced by intermediate file creations and use of the observation number as pointer to vast data amounts giving us faster access to raw data sets. By creation of small subsets from relatively massive amounts of data allows for faster analysis. And that is a tool for making elusive data more valuable and available to all users.

CONCLUSION

There is a better mousetrap solution for creation of files from somewhat cumbersome and large data sets. Through the usage of the above somewhat circuitous SORT, and INDEX techniques, and most interestingly usage the observation number coupled with the POINT option, tremendous efficiencies can be obtained. Critical placement of

WHERE clause structures remain an extremely valuable SAS tool in the speedy access of vast terabyte files. Raw data will increase in size in geometric proportion, not in arithmetic proportion, and this compounding of volumes of data sets requires users to utilize present tools to gain efficiencies in data handling techniques.

REFERENCES

1. Gupta, Sunil, “WHERE vs. IF Statements: Knowing the Difference in How and When to Apply”, Paper 213-2007, SUGI 2007 Proceedings, http://www2.sas.com/proceedings/forum2007/213-2007.pdf.

2. Wilcox, Andrew, “Efficiency Techniques for Accessing Large Data Files”, Paper 115-25, SUGI 29 Proceedings, http:www2.sas.com/proceedings/sugi25/25/dw/25p115.pdf.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Mikhail Gruzdev
U.S. Census Bureau
Foreign Trade Division
Rm 6K502
Washington, D.C. 20233
Phone: 301-763-2206
E-mail: mikhail.g.gruzdev@census.gov

Allen Blackburn
U.S. Census Bureau
Foreign Trade Division
Rm 6K106
Washington, D.C. 20233
Phone: 301-763-6921
E-mail: allen.j.blackburn@census.gov