Subsetting Observations from Large SAS® Data Sets
Christopher J. Bost, MDRC, New York, NY

ABSTRACT
This paper reviews four techniques to subset observations from large SAS® data sets: MERGE, PROC SQL, user-defined format, and DATA step hash object. Pros and cons of each technique are noted. Test results on a SAS data set with 100 million observations are summarized.

INTRODUCTION
Multiple techniques can be used to subset observations from large SAS data sets. Techniques vary in complexity, efficiency, and run times. Understanding the pros and cons of each can help the programmer determine the best technique for a given situation. This paper focuses on subsetting sample members from a much larger data set. The number of conditions (i.e., sample member IDs) precludes use of a WHERE statement.

SAMPLE DATA SETS
Three SAS data sets are used in this paper: SMALL, LARGE, and SUBSET.

Data set SMALL contains one variable and three observations. There is one observation for each ID (i.e., no duplicates). Each observation represents a sample member. Subset all observations with these IDs from data set LARGE.

<table>
<thead>
<tr>
<th>Data set SMALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Data set LARGE contains two variables and ten observations. There is more than one observation for some IDs (i.e., duplicates). Each observation represents a unique event. Occurrences of each ID are numbered sequentially by the variable N. Observations with ID values that occur in data set SMALL are highlighted.

<table>
<thead>
<tr>
<th>Data set LARGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Data set SUBSET contains two variables and six observations. It contains all variables in data set LARGE and all observations in data set LARGE with an ID value that occurs in data set SMALL.

<table>
<thead>
<tr>
<th>Data set SUBSET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
SUBSETTING OBSERVATIONS WITH MERGE

Use a match-merge to combine data sets SMALL and LARGE, and keep only observations where the ID occurs in both. For example:

```sas
proc sort data=small;
  by id;
run;

proc sort data=large;
  by id;
run;

data subset;
  merge small(in=inS)
    large(in=inL);
  by id;
  if inS and inL;
run;
```

Sort data sets SMALL and LARGE by ID with PROC SORT.

Merge the sorted data sets by ID. Use the IN= data set option to create variables that store information about the origin of each observation. SAS creates temporary variables INS and INL that have the value 1 when the current value of the BY variable contributed to the output data set at least once and that have the value 0 when it did not.

Use an IF-THEN statement to output only those observations with an ID in both data sets SMALL and LARGE to data set SUBSET.

**PROS AND CONS**

Pros: easy to use; well-known technique

Cons: Data set LARGE must be in sort order, sorted with PROC SORT, or indexed

SUBSETTING OBSERVATIONS WITH PROC SQL

Use PROC SQL to perform an inner join of rows (observations) in tables (data sets) LARGE and SMALL where the ID occurs in SMALL. For example:

```sql
proc sql;
  create table subset as
  select large.*
  from large, small
  where large.id = small.id;
quit;
```

Use CREATE TABLE to create data set SUBSET.

Use SELECT LARGE.* to retrieve all variables from data set LARGE.

Specify FROM LARGE, SMALL to read observations from data sets LARGE and SMALL.

Specify WHERE LARGE.ID = SMALL.ID to include observations with an ID in data set LARGE that matches an ID in data set SMALL.

**PROS AND CONS**

Pros: easy to use; well-known technique; data set LARGE does not have to be in sort order, sorted with PROC SORT, or indexed (although an index might improve performance)

Cons: SAS may run PROC SORT in the background
SUBSETTING OBSERVATIONS WITH A FORMAT
Use PROC FORMAT to create a temporary format that assigns all ID values in data set SMALL the same label. Then use a WHERE statement to subset observations from data set LARGE where the formatted value of ID equals that label. The format serves, in effect, as a "lookup table." For example:

```sas
proc format;
  value $key '2','4','5' = 'Y'
    other = 'N' ;
run;

data subset;
  set large;
  where put(id,$key.) = 'Y' ;
run;
```

Use PROC FORMAT to create a temporary character format named $KEY. The values '2', '4', and '5' will be formatted as 'Y'. All other values will be formatted as 'N'.

Create data set SUBSET with the DATA statement.

Read observations from data set LARGE with the SET statement.

Load only those observations where the value of ID, formatted with $KEY, via the PUT function, equals 'Y' with the WHERE statement. Observations with ID values '2', '4', and '5' are written to data set SUBSET.

INPUT CONTROL DATA SET
It is not practical to specify large numbers of ID values when creating a format. The FORMAT procedure, however, can create formats from an input control data set — a SAS data set that stores the information needed to create a format. For example:

```
Input control data set FMT

<table>
<thead>
<tr>
<th>Obs</th>
<th>fmtname</th>
<th>type</th>
<th>label</th>
<th>start</th>
<th>HLO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>key</td>
<td>C</td>
<td>Y</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>key</td>
<td>C</td>
<td>Y</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>key</td>
<td>C</td>
<td>Y</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>key</td>
<td>C</td>
<td>N</td>
<td>5</td>
<td>O</td>
</tr>
</tbody>
</table>
```

Data set FMT has the five variables required for an input control data set: FMTNAME (the name of the format); TYPE (the type of format ['C' for character or 'N' for numeric]); LABEL (the formatted value); START (the starting value to be formatted); and HLO (a special variable needed for formats that use High, Low, and/or Other).

Data set FMT has four observations, one for each value to be formatted. All observations have the value 'key' for FMTNAME and the value 'C' for TYPE. This input control data set will create one character format named $KEY. The first observation formats values that start with '2' as 'Y'. (The ending value is not specified; it defaults to the starting value.) The second and third observations format values that start with '4' and '5' as 'Y' as well. The fourth observation has the value 'O' for HLO, indicating that any Other values not specified should be formatted as 'N'. (The value '5' for START is ignored when HLO equals 'O'.)

BUILDING AN INPUT CONTROL DATA SET FROM DATA SET SMALL
Build an input control data set that assigns the label 'Y' to all IDs in data set SMALL. For example:

```sas
proc sort data=small nodupkey out=ids;
  by id;
run;
```
data fmt(rename=(id=start));
  retain fmtname 'key'
    type 'C'
    label 'Y' ;
set ids end=eof;
output;
if eof then do;
  HLO   = 'O' ; /* format all Other */
  label = 'N' ; /* values as 'N' */
  output;
end;
run;

First, sort data set SMALL with PROC SORT. Use the NODUPKEY option to delete any observations with duplicate ID values. (An input control data set should not have any duplicate values, just like a VALUE statement in PROC FORMAT.) Use the OUT= option to store the sorted observations in data set IDS. These observations contain the unique ID values to subset from data set LARGE.

Next, start to create data set FMT, the input control data set, with the DATA statement. Use the RETAIN statement to create the required variables FMTNAME, TYPE, and LABEL. The value ‘key’ is assigned to FMTNAME; the value value ‘C’ is assigned to TYPE; and the value ‘Y’ is assigned to LABEL. Values are retained across all observations.

Read observations from data set IDS with the SET statement. Create an end-of-file variable named EOF that equals 1 when SAS reaches the last observation and that equals 0 for all other observations. Output each observation that is read to data set FMT.

Use an IF-THEN statement to tell SAS to output an additional observation to data set FMT after reading all observations from data set IDS (i.e., after it reaches the end of the file, or EOF). The variable HLO (High/Low/Other) is created and assigned the value ‘O’ for ‘Other’. The value ‘N’ is assigned to label all other values not specified. Output the observation to data set FMT.

At the end of the DATA step, rename ID to START (i.e., one of the variables required in an input control data set) as specified on the data set option.

These statements create input control data set FMT printed on page 3.

CREATING THE FORMAT FROM THE INPUT CONTROL DATA SET

Use PROC FORMAT to create the format from the input control data set. For example:

    proc format cntlin=fmt fmtlib;
    run;

Use PROC FORMAT to create formats. Use the CNTLIN= option to specify the input control data set FMT. Use the FMTLIB option to print the format (i.e., for illustrative purposes only). Format $KEY has the following specification:

<table>
<thead>
<tr>
<th>FORMAT NAME: $KEY</th>
<th>LENGTH: 1</th>
<th>NUMBER OF VALUES: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN LENGTH: 1</td>
<td>MAX LENGTH: 40</td>
<td>DEFAULT LENGTH: 1</td>
</tr>
</tbody>
</table>

The format $KEY labels all ID values in data set SMALL as ‘Y’, and all other ID values as ‘N’.

| START | END | LABEL (VER. V7|V8 05JUL2006:09:44:05) |
|-------|-----|------------------|
| 2     | 2   | Y                |
| 4     | 4   | Y                |
| 5     | 5   | Y                |
| **OTHER** | **OTHER** | N               |
SUBSETTING OBSERVATIONS WITH THE FORMAT

Finally, subset observations in data set LARGE where the value of ID formatted with $KEY equals 'Y':

```sas
data subset;
  set large;
  where put(id,$key.) = 'Y' ;
run;
```

PROS AND CONS

Pros: speed; data set LARGE does not have to be in sort order, sorted with PROC SORT, or indexed

Cons: requires multiple steps; not an intuitive or obvious technique

SUBSETTING OBSERVATIONS WITH THE DATA STEP HASH OBJECT

Use the DATA step hash object, an in-memory lookup table, to store all ID values in data set SMALL. Process data set LARGE and output any observations where the ID value is found in the hash table. For example:

```sas
data subset;
  if _N_ = 1 then do;
    declare hash h(dataset:'small');
    h.defineKey('id');
    h.defineDone();
  end;
set large;
  if h.find() = 0 then output;
run;
```

Create data set SUBSET with the DATA statement.

In the first iteration of the DATA step (i.e., _N_ = 1), use the DECLARE statement to create a hash object named H. Use the DATASET: argument tag to load the contents of data set SMALL into the hash object.

Use the DEFINEKEY method to define ID as the key variable for the hash object.

Use the DEFINEDONE method to indicate that all definitions are complete.

Close the DO loop with an END; statement.

Read observations from data set LARGE with the SET statement.

Use the FIND method to determine whether the key (i.e., the current value of ID from data set LARGE) is stored in the hash object. A return code of 0 indicates that it was found. Output all observations where the ID is found in the hash object.

PROS AND CONS

Pros: speed; efficiency; data set LARGE does not have to be in sort order, sorted with PROC SORT, or indexed

Cons: hash table must fit in memory; keys must be unique

TEST RESULTS

The above techniques were evaluated using SAS 9.1.3 Service Pack 3 on an IBM xSeries 445 server with four Intel Xeon processors running at 3.0 GHz and 16 GB of RAM, under Windows Server 2003, Enterprise Edition. Files were on a NetApp FAS3050c storage system with data files and SASWORK files on separate filer heads.

- Data set SMALL contains 10,000 observations and 1 variable (85K)
- Data set LARGE contains 100,000,000 observations and 101 variables (22 GB with COMPRESS=BINARY)

Techniques, respective run times (i.e., times for the entire SAS program to complete), and additional notes are summarized in the following table:
<table>
<thead>
<tr>
<th>Technique</th>
<th>Run Time</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERGE</td>
<td>2 hr, 36 min</td>
<td>SORT: 2 hr, 17 min</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MERGE: 0 hr, 20 min</td>
</tr>
<tr>
<td>PROC SQL</td>
<td>0 hr, 16 min</td>
<td></td>
</tr>
<tr>
<td>Format</td>
<td>0 hr, 17 min</td>
<td>FORMAT: 0.1 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATA step: 17 min</td>
</tr>
<tr>
<td>Hash object</td>
<td>0 hr, 17 min</td>
<td></td>
</tr>
</tbody>
</table>

The PROC SQL technique, the format technique, and the hash object technique had comparable run times. The MERGE technique was dramatically slower (i.e., due to the PROC SORT step).

**EFFICIENCY**

SAS reports real time (i.e., the “clock” time that it takes for a step to run) and CPU time (the amount of time that the processor[s] are being used). The difference between real time and CPU time is the amount of time that the processors are idle, waiting for data from the I/O subsystem.

The MERGE technique (i.e., including the SORT) took 2 hours, 36 minutes, approximately 51 minutes of which were CPU time. The PROC SQL technique took 16 minutes, approximately 7 minutes of which were CPU time. The format technique took 17 minutes, approximately 7 minutes of which were CPU time. The hash object technique took 17 minutes, approximately 8 minutes of which were CPU time. The hash object technique had the highest ratio of CPU time/real time.

**CONCLUSIONS**

Different techniques to subset observations from large SAS data sets may entail significantly different run times. In our tests, the SQL technique, format technique, and hash technique worked faster than the MERGE technique. Performance may vary based on the operating environment and files used. Test multiple techniques where time permits to see which works best.

**REFERENCES**


**ACKNOWLEDGMENTS**

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.

The author appreciates the help of Janice Bloom and Jason Secosky in reviewing this paper.

**CONTACT INFORMATION**

Christopher J. Bost  
Director, Research Technology Unit  
MDRC  
16 East 34th Street, 19th Floor  
New York, NY 10016  
(212) 340-8613 telephone  
(212) 684-0832 fax  
christopher.bost@mdrc.org  
www.mdrc.org