Understanding Indexed Datasets and Using Direct Access Queries

John Charles Gober - Intellisoft CPI

Introduction

The MERGE statement is one of the first statements a SAS® programmer learns. However merges can take a considerable amount of processing time, especially when only a few records of a 10 million record dataset are needed. This paper will discuss how and where the MERGE statement can be replaced with a keyed lookup routine often reducing I/O and CPU usage by over 90 percent.

Scenario

The Bogus Bulb Company, a purveyor of questionable quality garden bulbs, has a clientele of over ten million gardeners across the U.S. Periodically the Company's marketing department creates special promotions targeting a select number of customers which will usually consist of no more than 10 percent of their total clientele. These target customers, known only by their customer numbers, are selected based on various criteria including frequency of purchases, kind and number of bulbs purchased, climate zone, etc. Once selected, the customer numbers are then given to the sales department in the form of a SAS dataset in order to create the customer mailing list off of the company's demographics dataset.

Currently the demographics dataset is sorted by the variables ZIPCODE, STATE, CITY, contains the variable CUSTNO (which is unique), ADDRESS1 and ADDRESS2 (which are not unique), is updated once a week, and is not indexed. The target dataset is sorted by CUSTNO which is unique. Addresses of the targeted customers are obtained by resorting the ten million plus records in the demographics dataset and then merging them with the target dataset. A process that takes a considerable amount of space and resources.

Creating the Index

According to SAS, indexes are not recommended for '*WHERE-expression processing if you expect to retrieve more than one-third of the observations in the dataset*'. Even though in this case we will not be doing any *WHERE-expression* processing, the same principle applies for direct access processing. It has already been stipulated earlier that no more than ten percent of the clientele will need to be accessed so the data is well within the range of SAS recommendations. In addition the demographic dataset will only be updated once a week, therefore the extra processing time to maintain the indexes will be minimized.

Since only one variable, CUSTNO, needs to be indexed and has unique values then only a simple unique index needs to be created using the DATASETS procedure. However for the purpose of this paper we will also need to create a second non unique index to demonstrate keyed sequential access. Figure 1 contains the statements to create both the unique and non unique indexes. Also, once created, the index will automatically be maintained as long as the dataset is not deleted. These indexes could have also been created in the data step.

```
PROC DATASETS LIBRARY=BOGUS;
   MODIFY GEOFILE,
   INDEX CREATE CUSTNO/UNIQUE;
   INDEX CREATE ADDRESS1;
RUN;
```

figure 1

```
DATA BOGUS.GEOFILE
   (INDEX=(CUSTNO/UNIQUE
           ADDRESS));
   ...PROGRAM CODE;
RUN;
```

figure 2

The Merge Process

The most common method of selecting observations from one dataset based on observations from a second dataset is by using the *MERGE-expression*. This requires that both datasets either be sorted or indexed by the lookup variable. This method also requires reading in every record from both datasets even if one of them has only ten or twenty records compared to millions in the second dataset. Code for the MERGE-expression processing is located in

Figure 2. Time trials for various size target files are at the end of this paper. Notice that there is no significant time difference when merging a small dataset with a large one than when merging two large datasets.

```
PROC SORT DATA=BOGUS.TARGET
          OUT=WORK.TARGET;
   BY CUSTNO;
RUN;
PROC SORT DATA=BOGUS.GEOFILE
          OUT=WORK.GEOFILE;
   BY CUSTNO;
RUN;
DATA MAILLIST;
   MERGE WORK.TARGET(IN=TARGET)

WORK.GEOFILE(IN=GEOFILE);
   BY CUSTNO;
IF TARGET AND GEOFILE;
RUN;
```

figure 3

The Keyed Access Method

The code sample in Figure 3 is an example of how Bogus Bulb would use keyed access to obtain mailing addresses from their demographics file based on the targeted customer file. The automatic

```
DATA WORK.ADDRESS
   SET WORK.TARGET;
   SET WORK.GEOFILE
      KEY=CUSTNO / UNIQUE;
   IF (_IORC_ NE %SYSRC(_DSENOM))
    THEN OUTPUT;
     ELSE DELETE;
RUN;
```

Figure 4

SAS variable **_IORC_** is a variable that contains a number representing the status of the latest I/O operation. In this case it will contain the result of the ***SET WORK.GEOFILE...*** operation. The macro in Figure 3 refered to as **%SYSRC** is a SAS supplied macro which can usually be found in the **SASAUTOS** subdirectory. Its purpose is to equate many of the most common SCL return codes from numbers to the more easily remembered variable names. For example the variable **_DSENOM** equates to the return code 1230015 which means no matching observation on the master dataset was found. If the variable _IORC_ is also 1230015 it means the read was unsuccessful and the record will not be output (note the NE operator).

It is good practice to place code similar to that in Figure 4 in an appropriate place in your dataset to clear out the PDV. This will insure the PDV (program data vector) has been cleared for the next read operation. Reason being that SAS retains values found from the result of an indexed read until replaced by another *successful* read or by encountering a *run* statement. In the case of multiple index reads in one pass through the dataset values may be retained from a previous read.

The Keyed Sequential Access Method

Bogus Bulb's demographics dataset BOGUS. GEOFILE has also been indexed by the variable ADDRESS1. Throughout the years customers have moved, mailing lists have been bought and sold, duplicate addresses now appear on the GEOFILE. From time to time it is necessary to perform inquiries on specific addresses. By using the non unique index on ADDRESS1 in conjuction with the variable ZIPCODE a search can be executed to find all customers having identical addresses (Figure 5). Note that ZIPCODE is not a key variable in this scenario. Items of note in this code are as follows: 1) use BOGUS.LOOKUP as the transaction dataset being sure to rename any variable such as ZIP to avoid conflict in the PDV. 2) Execute the *do until* loop until the system return code _IORC_ equals the value represented by _DSENOM which is 1230015

```
DATA BOGUS.DUPADDRS;
   SET BOGUS.LOOKUP
   (RENAME=(ZIPCODE=NEWZIP));①
  DO UNTIL
        (_IORC_=%SYSRC(_DSENOM));②
     SET BOGUS.GEOFILE
          KEY=ADDRESS1;③
     SELECT (_IORC_);④
       WHEN (%SYSRC(_SOK))⑤
      DO;
            IF ZIPCODE = NEWZIP
            THEN OUTPUT;
         END;
         WHEN (%SYSRC(_DSENOM))⑥
      DO;
            _ERROR_ = 0;
         END;
      END;
   END;
RUN;
```

Figure 5

meaning key not found. 3) Read a record from BOGUS.GEOFILE which contains the non-unique key of ADDRESS1 which is being supplied from the file BOGUS.LOOKUP, based on the value of _IORC do the following: 5) if _IORC is 0 (the value of _SOK) then a match was found therefore continue to verify the zipcode and output if true, 6) if _IORC is 1230015 (the vakue of _DSENOM) then a match was not found so do not output any record but change the _ERROR_ variable back to 0 (_ERROR was set to 1 due to an unsuccessfull read). End of do loop so check the DO UNTIL condition and execute the loop again if necessary.

Note that because of the non-unique key in BOGUS.GEOFILE the pointer will automatically be positioned at the next matching record (if any) and only be repositioned to a new key until no more old keys appear in the index.

Accessing Multiple Files (historic data)

Ocassionally, it might be necessary to access several files in order to find sought after information. This is most likely to be true when dealing with historic, yearly, or archived data. One method of dealing with

```
DATA BOGUS.RESULTS;
   SET BOGUS.LOOKUP;
   SET BOGUS.YEAR96
       KEY=CUSTNO /UNIQUE;
   IF (_IORC_NE %SYSRC(_DSENOM))
      THEN OUTPUT;
   ELSE
   DO;
      SET BOGUS.YEAR95
          KEY=CUSTNO /UNIQUE;
      IF (_IORC_NE %SYSRC(_DSENOM))
          THEN OUTPUT;
      ELSE
      DO;
         ...more indexed reads
            ... more indexed reads
     END;
   END;
RUN;
```

Figure 6

this kind of data is to *layer* the code in a series of embedded do loops as in figure 6. The array statements used to clear the PDV have been left out of this example to conserve space. Also, with a little more code and thought, this routine can easily be modified to accommodate the use of non unique keys.

Unpredictable Results

Again it can not be emphasized enough to carefully think out your code. In many cases the clearing out of the PDV after a lookup is unecessary. But in other cases unwanted results may creep into your results. Espeacially when retaining variable and working within do loops, etc.

To Direct Access or Not

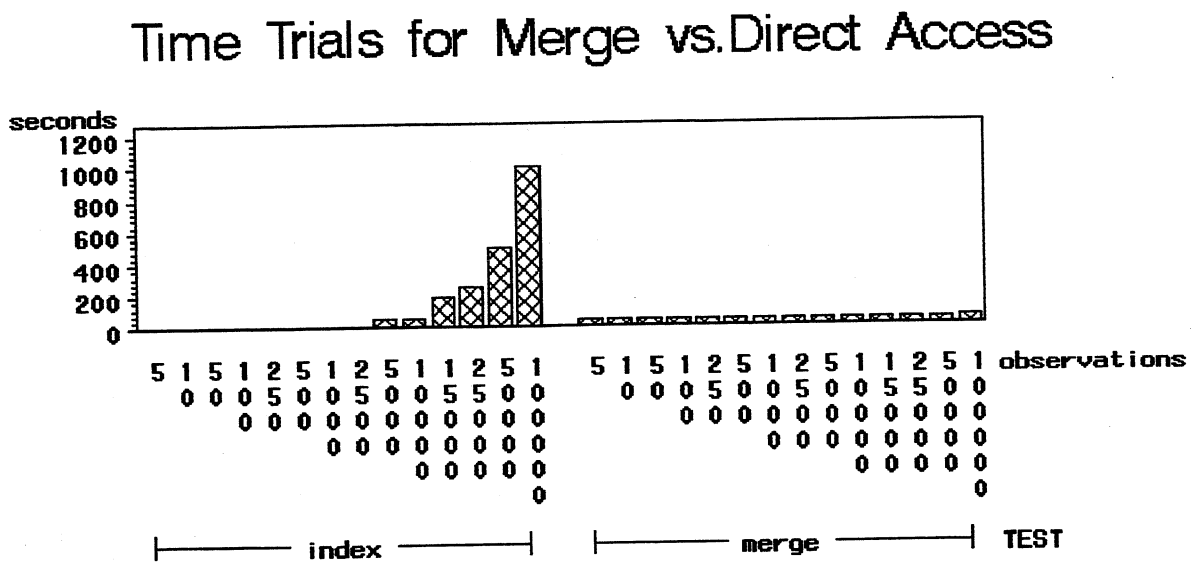
In order to verify the *one-third* rule for direct access, we created a dataset containing one million records with unique customer numbers. This represented our master file. We then randomized the customer numbers and created lookup datasets with 5, 10, 50, 100, 500, 1000, 2500, 5000, 10000, and 20000 observations. These lookup datasets were then each merged with the master to determine times for the common sequential access method. Then the lookup datasets were again used to access the master dataset but using direct access. These trials were undertaken both on a dedicated PC and on a Unix enviroment with many users going hard against against it. Both gave similar results. The four charts at the end of the paper represents the results. As you will noticed it is evident that direct access can either save you an incredible amount of processing time if correctly applied . . . or loose you your job.

Conclusion

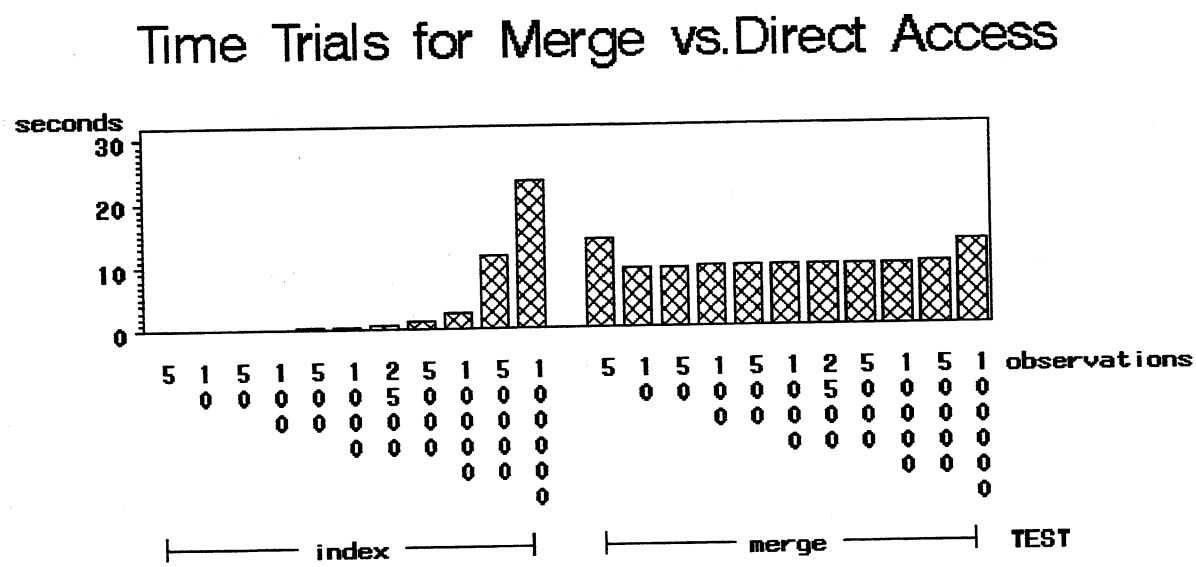
Based on time trials, direct access is a viable option for matching data for a limited number of records. However it may not be the best option. Some of the ey factors in deciding are: The number of records to be access, how static is your dataset (with the advances in SAS the time diferences between creating an index and doing a sort are almost nil); the expertise of your programmers; and the frequency of execution of the program.

Author and Contact Information

John Charles Gober
Intellisoft CPI
1006 West 9th Avenue
King of Prussia, PA 19406-1206
j.gober@worldnet.att.net



PC Environment using SAS 6.12



Unix Environment using SAS 6.12

Comparison of Sequential Access vs. Direct Access

OBS	PLATFORM	Number of Observations	Direct Access	Sequential Access
1	PC	5	0.77	43.77
2	PC	10	0.44	42.13
3	PC	50	1.37	43.67
4	PC	100	1.69	45.92
5	PC	250	1.50	44.10
6	PC	500	10.43	42.61
7	PC	1000	11.41	43.78
8	PC	2500	11.41	41.03
9	PC	5000	57.34	46.46
10	PC	10000	59.56	45.09
11	PC	15000	190.66	43.21
12	PC	25000	250.67	46.03
13	PC	50000	495.03	46.96
14	PC	100000	1003.42	52.35
15	Unix	5	0.24	13.95
16	Unix	10	0.15	9.44
17	Unix	50	0.17	9.52
18	Unix	100	0.20	9.66
19	Unix	500	0.32	9.61
20	Unix	1000	0.47	9.68
21	Unix	2500	0.79	9.67
22	Unix	5000	1.34	9.69
23	Unix	10000	2.54	9.78
24	Unix	50000	11.68	10.05
25	Unix	100000	23.33	13.34