PharmaSUG 2012 - Paper TF08

Merge vs. Join vs. Hash Objects: A Comparison using "Big" Medical Device Data

James Johnson, Medtronic, Inc., Minneapolis, MN

ABSTRACT

Due to the use of memory-based operations, hash objects have the potential to reduce the time it takes to combine and sort data files. This presentation compares the performance of three methods in SAS® used to retrieve a subset of data from a SQL Server database table with over 370 million records. The methods compared are 1) merging data in a DATA step and sorting with PROC SORT; 2) PROC SQL inner join with an ORDER BY statement; and 3) hash object programming.

INTRODUCTION

For years, combining data from two data files matched according to key fields was done in SAS by first using the SORT procedure to order records in each file by the keys and then bringing the data together in a DATA step with a MERGE statement. If there was a need to post-sort the final data file by another set of key fields, then another PROC SORT was used. With the introduction of the SQL procedure, all three tasks could be accomplished in SAS with a single procedure, since joining data in PROC SQL does not require any pre-sorting and the ORDERED BY statement can be used to sort the final data file. More recently, the SAS hash object became available in Version 9, giving users another method for combining and sorting data. When combining two data files, hash object programming, like PROC SQL, does not require any pre-sorting of the data files. However, unlike PROC SQL, post-sorting of the final data set requires a separate hash object.

While combining and sorting of data might be considered trivial applications of hash object programming (1), these have become popular uses of hash objects because of the potential to accomplish these tasks faster than other methods available in SAS. Contrasted to a DATA merge or PROC SQL, which repeatedly access data stored on disks, hash objects read the contents of a data file into memory once. Hash object operations are then run entirely in memory, which are usually faster than disk-based operations.

This paper compares the performance of the three methods in the processing of tables from a large medical device database.

IMPLANTABLE CARDIOVERTER-DEFIBRILLATORS

Implantable cardioverter-defibrillators (ICDs) are battery powered devices that are implanted in the chest and connected to the heart by wires (“leads”) (Figure 1). They are designed to detect potentially life-threatening irregular heartbeats known as ventricular arrhythmias and at such time to generate an electrical impulse to jolt the heart back

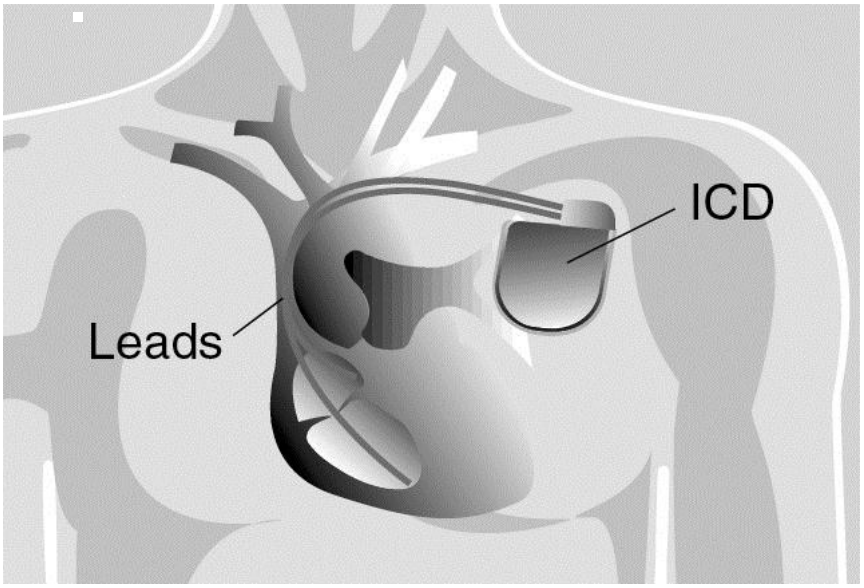

Figure 1. ICD implant

into normal sinus rhythm. These devices store data on programmed parameters and daily measurements of heart rate, patient activity, detected arrhythmias, and delivered therapies. Stored data is reviewed by physicians when patients attend follow-up visits. Since 2005 ICD manufacturers have implemented remote monitoring systems that allow device data to be transferred electronically over phone networks into a secure data server for later review through a secure internet web-site. Remote transmissions can be sent according to pre-determined schedules programmed into the device by clinic staff or on-demand when need arises (Figure 2).

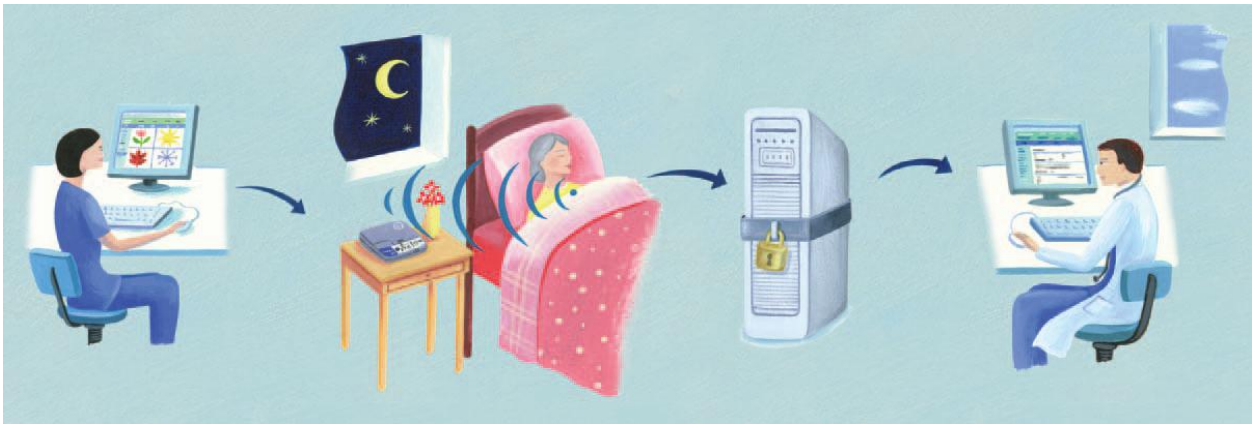

Figure 2. Automatic follow-up of ICD patients: 1) clinic staff programs schedule of device checks; 2) data are transmitted wirelessly as the patient sleeps; 3) data are received into a secure server; clinician reviews device data on secure web-site

The wealth of transmitted device data provides manufacturers with an opportunity to better understand how these devices are used and how they perform. Physicians utilizing remote monitoring systems for patient follow-up are asked to allow transmitted data be used for research purposes. Centers that agree to participate sign a data use agreement. Transmitted data is moved into a data warehouse that adheres to regulations stipulated by the Health Insurance Portability and Accountability Act (HIPAA). With this information, improved algorithms can be designed to more effectively detect and treat potentially fatal arrhythmias.

MEDTRONIC DE-IDENTIFIED DATA MART

The Medtronic CareLink® Network is used for remote monitoring of cardiac devices manufactured by Medtronic, Inc. Data transmitted into the network data warehouse is moved into a data mart that has been created for research purposes (Figure 3).

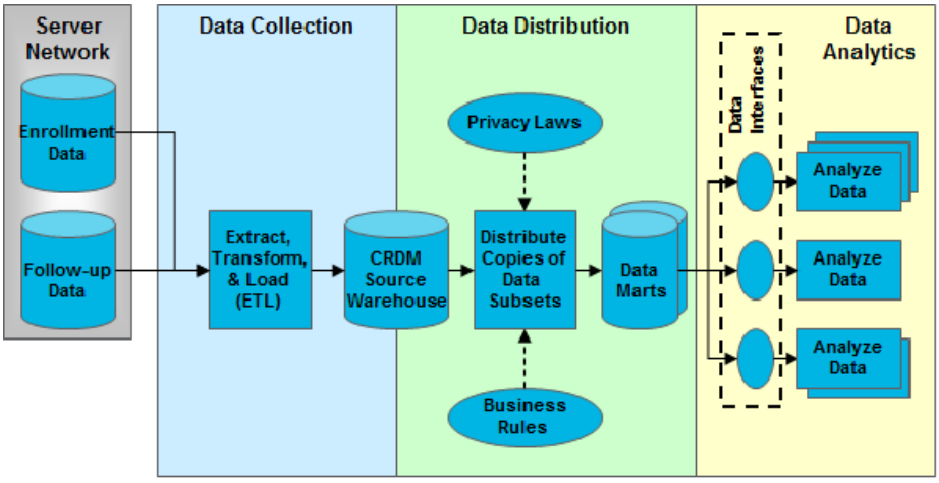

Figure 3. Flow of data from secure network server into de-identified data marts.

The data mart is a SQL Server relational database and is de-identified in the sense that all patient identifiable information, including device serial numbers, has been removed.

As of March 16, 2012 data from over 390,000 devices were included in the data mart. The largest table, with daily information on device performance (thus multiple rows per device), contains over 371,000,000 records.

DATA RETRIEVAL USING SAS

In this example, records are selected from the largest table in the data mart for a subset of devices. A SAS data file contains the devices of interest, and it is first sorted by a key field.

```
proc sort data=dev1;
  by devicekey;
run;

NOTE: Table WORK.DEV1 created, with 71118 rows and 3 columns.
```

The next sections show the steps for selecting records in the data mart table for devices in DEV1 using 3 methods: merging data in a DATA step and sorting with PROC SORT, PROC SQL inner join with an ORDER BY statement, and using hash object programming. The programs were executed using SAS Version 9.2.

DATA STEP WITH PROC SORT

Records from the data mart table are read using SAS/Access for ODBC. When referencing a database table with a BY statement using SAS/Access products, SAS sorts the data before it is used in the program. Thus DEV1 and the data mart table can be merged in a DATA step without a preceding PROC SORT of the data mart table.

```
libname dm odbc dsn='data_mart';
data daily1;
  merge dev1(in=in1)
        dm.dailydata(in=in2);
  by devicekey;
  if (in1 and in2);
run;

NOTE: There were 71118 observations read from the data set WORK.DEV1.
NOTE: There were 371853941 observations read from the data set DM.dailydata.
NOTE: The data set WORK.DAILY1 has 79704457 observations and 26 variables.
NOTE: DATA statement used (Total process time):
      real time      1:20:46.79
      cpu time       42:26.93
```

The final data file DAILY1 contains multiple rows per device. In addition, some patients might have been implanted with a new device after a previous device reached its end-of-life. The following code shows the sorting of the final data file.

```
proc sort data=daily1
  by patientkey implantdate devicekey dailydate;
run;

NOTE: There were 79704457 observations read from the data set WORK.DAILY1.
NOTE: SAS threaded sort was used.
NOTE: The data set WORK.DAILY1 has 79704457 observations and 26 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time      1:15:26.26
      cpu time       11:04.67
```

PROC SQL

With PROC SQL, pre-sorting of either data file is not necessary in order to join them into a single file. In addition, sorting of the final data file can be done using an ORDER BY statement in the procedure. An INNER JOIN is used so that only observations with records in both data files are included in the final output.

```
proc sql;
  create daily2 as
    select a.*,
           b.*
    from dev1 a,
         dm.dailydata b
    where (a.devicekey = b.devicekey)
    order by a.patientkey, a.implantdate, a.devicekey, b.dailydate;
quit;

NOTE: There were 71118 observations read from the data set WORK.DEV1.
NOTE: There were 371853941 observations read from the data set DM.dailydata.
NOTE: The data set WORK.DAILY2 has 79704457 observations and 26 variables.
NOTE: PROCEDURE SQL used (Total process time):
      real time      3:21:27.36
      cpu time       51:06.94
```

HASH OBJECT PROGRAMMING

The hash object syntax used follows that presented by Lafler (2).

Combining of data using hash objects does not require any pre-sorting of the data files. In the steps below, all records from the smaller data file DEV1 are loaded into the hash object before records from the larger SQL table DAILYDATA are processed. Records in DAILYDATA with matching key fields in DEV1 are then added to the final data file DAILY3.

```
data daily3;
  if (0) then
    set dev1                            /*  Include properties of all fields from */
        dm.dailydata;                   /*  both tables into the hash table.      */

  set dm.dailydata;

  if (_n_ eq 1) then
    do;
      declare hash a (dataset:'dev1',   /*  DEV1 is loaded into the hash object.  */
                      hashexp:20);      /*  Use maximum hash table size.          */
      a.definekey('devicekey');         /*  Define key field.                     */
      a.definedata(all:'y');            /*  Include all fields in final output.   */
      a.definedone();
    end;

  if (a.find() eq 0);                   /*  Output record when match found.       */

run;

NOTE: There were 71118 observations read from the data set WORK.DEV1.
NOTE: There were 371853941 observations read from the data set DM.dailydata.
NOTE: The data set WORK.DAILY3 has 79704457 observations and 26 variables.
NOTE: DATA statement used (Total process time):
      real time      1:07:50.37
      cpu time       46:04.74
```

After the final data file is created, it can be sorted using additional hash syntax. In this case, however, the data file was too large to sort with a single hash object. It was necessary to write a macro that sorted approximately two

million records at a time, and then interweaved all of the smaller data files. The macro appears in the Appendix.

SUMMARY OF RESULTS

Table 1 summarizes the time needed to accomplish the tasks described above. Initially these steps were done using SAS Version 9.2 but were repeated when Version 9.3 became available. Version 9.3 appears to provide a considerable improvement in the performance of both PROC SORT and sorting within PROC SQL.

Method for combining/sorting	V9.2 Real time (minutes)	V9.3 Real time (minutes)
DATA step MERGE/PROC SORT	81 + 75 = 156	82 + 56 = 138
PROC SQL	67 + 134* = 201	62 + 37* = 99
Hash match-merge/sort	68 + 63** = 131	63 + 58** = 121

Table 1. Performance of 3 methods to combine and sort data.
*Difference between time with and without an ORDER BY statement.
**Total time for executing a macro to sort and interweave smaller data files.

For this example, under V9.2, combining and sorting large data files using hash object programming was 16% and 35% faster, respectively, than either a DATA step with PROC SORT or PROC SQL. While both hash programming steps were each more than 10 minutes faster than the DATA step and PROC SORT, the hash sort took less than half the time needed for sorting within PROC SQL. However, major improvements in the sorting steps are seen with Version 9.3. These improvements result in PROC SQL being 18% faster than hash programming.

CONCLUSION

Several authors have described how hash object programming can improve the performance of joins and sorts as compared to other methods in SAS. Blackwell (3) reports on a hash object example that improved the time it took to extract data from a large Oracle table from 30 minutes down to 3 minutes. In the example presented here, involving medical device data with tens of millions of records, hash programming demonstrated 16% and 35% reductions in time compared to other methods using SAS Version 9.2. However, with Version 9.3, PROC SQL is faster than hash programming in accomplishing these tasks. While there are advantages to using hash objects when working with large data files, the results of the comparisons presented here led to the decision to use PROC SQL.

REFERENCES

1. Dorfman, Paul M and Shajenko, Lessia S (2011). "Hash + Point = Key." Proceedings of the 2011 North East SAS Users Group (NESUG) Conference.
2. Lafler, Kirk Paul (2011). "An Introduction to SAS® Hash Programming Techniques." Proceedings of the 2011 Pharmaceutical SAS Users Group (PharmaSUG) Conference.
3. Blackwell, John (2010). "Find() the Power of Hash – How, Why and When to Use the SAS® Hash Object." Proceedings of the 2010 North East SAS Users Group (NESUG) Conference.

CONTACT INFORMATION

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

James Johnson
Medtronic, Inc.
8200 Coral Sea Street – MS MVN41
Mounds View, MN 55112
(763) 526-0255
james.johnson@medtronic.com

APPENDIX

Macro used for sorting larger data file in parts using hash programming.

```
%macro hashsort(incr=);
  proc sql noprint;
    select count(*) into :nrec
    from DAILY3;
  quit;
  %let maxi=%sysfunc(ceil(&nrec/&incr));
  %do i=1 %to &maxi;
    %let np1=%eval((&i-1)*&incr+1);
    %let np2=%sysfunc(min(&i*&incr,&nrec));
    data _null_;
      if (0) then set DAILY3;
      if (_n_ eq 1) then
        do;
          declare hash b(dataset:"DAILY3(firstobs=&np1 obs=&np2)",
                         hashexp:20,
                         multidata:'y',
                         ordered:'a');
          b.definekey('patientkey', 'implantdate', 'devicekey', 'dailydate');
          b.definedata(all:'y');
          b.definedone();
        end;
      b.output(dataset:"temps&i(compress=yes)");
    run;
  %end;
  data DAILY3_SORTED(compress=yes);
    set
      %do i=1 %to &maxi;
        temps&i
       %end;
       ;
    by patientkey implantdate devicekey dailydate;
  run;
%mend hashsort;
%hashsort(incr=2000000);
```