Paper 96-28

PROC SQL vs. Merge. The Miller Lite Question of 2002 and Beyond.

Kevin J. Smith; Oxford Health Plans Trumbull, CT
Muhammad Z. Khan; Oxford Health Plans Trumbull, CT
Yadong Zhang; Oxford Health Plans Trumbull, CT

ABSTRACT

Whenever SAS® programmers get together to discuss SAS tips and techniques, there always seems to be a conflict regarding linking two tables together. I have noticed that programmers have very strong allegiances to either PROC SQL or Merge. I often think if this was a sports commercial, instead of arguing "Taste's Great" and "Less Filling" we would be arguing "In A and B" or "Left Join".

For a beginning SAS programmer, seeking help from co-workers can be very challenging as one will preach "tastes great" and the other "less filling" and I feel the novice programmer will actually seek the Miller Lite beer as they don't fully understand either technique and when it is appropriate to use one over the other. In this analysis, we will have an active discussion on not only the coding aspect of the two methods; we will discuss hashing and system issues such as CPU, Input/Output, Disk Space, and memory system resources and how it affects productivity.

INTRODUCTION

When we talk about PROC SQL and Merge, the primary focus of this paper will involve combining multiple SAS datasets into one new table. The emphasis of this paper is not designed to teach you Merge or PROC SQL, but the reader should have a basic understanding of each skill and this paper will help emphasize the two techniques and when to appropriately apply each of them. Included in the paper, we will be showing examples of PROC SQL and Merge in action. Now, I would be remiss if I didn't mention there are other ways of combining SAS datasets. One example is a Multiple SET Statement. However, using multiple SET Statements are most effective using small data sets and the matching variables need to be indexed. We will however, discuss the effects of hashing. Hashing is also an effective way of combining SAS datasets, and we will discuss the performance mechanism of each system.

To effectively determine which mechanism is better, we will be concentrating on the following areas:

- CPU Time. This is the amount of time the Centralized Processing Unit takes to complete a task.
- Memory. The total memory used.

The focus of this paper will be on the CPU time and memory for the different methods. Although disk space is an important feature, for our purposes we will "pretend" there is an adequate amount of space on your operating system. Remember, we are assuming that all other users are "good corporate citizens" and no-one rogue programmer is running amok and crashing the system. Although we can't control the Disk Space, it should be noted that we did understand this could be a vital topic and were aware of its implication. Another element is the input/output time, this is the amount of time it takes for the SAS system to read the data and produce a new dataset for you. The last consideration, which is very difficult to measure, will be the SAS Programmer's time. Once again, although it is an important factor, we will not take its implications into account.

To decide which technique is better, we developed benchmarks. These benchmarks tested certain programs in separate SAS sessions. We tested these programs at different intervals during different times of the day to create a normalized factor, which we can then apply. The idea was to run a synopsis, which won't let one "outlier" affect the total outcome.

HYPOTHESIS

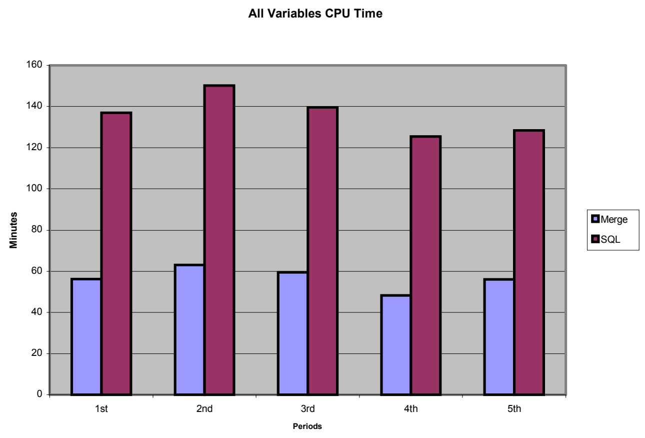
For five weeks straight, we ran the same program every single day at different times during the day to measure the program effectiveness.
We ran the program three times a day at different times, in different orders to get a sense of the different outcomes. The below graphs represent the weekly average.
The data was set up where PROC SQL and the Merge can basically do the same feature. Therefore, we eliminated any discrepancies between the two.
During the middle of the test, the tables were refreshed with "new" data.
We measured the effectiveness of hash joins.

MERGE VS. PROC SQL USING ALL VARIABLES

```
options fullstimer;

proc sql;
create table clmsql2 as
select  a.*,b.*
from medmart.qclaims as a, medmart.members as b
where a.memcode=b.memcode and a.dosdte between
'01jan2002'd and '31jan2002'd ;
quit;

data mergestep2;
merge medmart.qclaims ( in=a  where = ( '01JAN2002'd <=
dosdte <= '31JAN2002'd))
medmart.members (in=b  )  ;
by memcode;
if a and b ;
run;
```

In this example, we pulled one month's worth of claims (January 2002) from the claims table and pulled the entire member demographics information from the member's table. It is important to note, that both tables are sorted and indexed on the memcode field. Memcode is the member identification number.

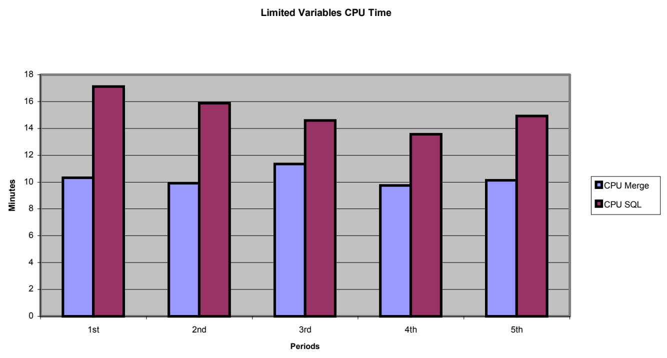
The data yielded some noticeable results. In system CPU time, SQL always took at least twice as long to complete the same task. The Merge step took approximately 56 minutes to 1 hour and three minutes to complete. SQL on the other hand, took between 2 hours and 5 minutes to 2 hours and 31 minutes to complete the same task. The system memory aspect was really intriguing. The memory was over 517,013K in SQL. This number remained consistent throughout the study, except when the data was refreshed. At that point the memory increased slightly. On the Merge step, the memory remained at 206K throughout the study. This is a big advantage for Merge.



MERGE VS. PROC SQL USING LIMITED VARIABLES

```
options fullstimer;

proc sql;
create table clmsql as
select  a.prvcode, a.reqamt, a.allowamt, a.payamt, a.memcode,
b.memfname, b.memlname
from medmart.qclaims as a, medmart.members as b
where a.memcode=b.memcode and a.dosdte between
'01jan2002'd and '31jan2002'd ;
quit;

data mergestep;
merge medmart.qclaims ( in=a keep = prvcode reqamt allowamt
payamt memcode dosdte where = ( '01JAN2002'd <= dosdte <=
'31JAN2002'd))
medmart.members (in=b keep = memcode memfname
memlname )  ;
by memcode;
if a and b ;
run;
```

In this example, we used the same program as before, except we used "keep" statements and limited the fields to only a handful of variables. Once again, Merge returned more desirable results, however, this time, the results were closer. On the Merge statement, the system CPU time ranged from 9.76 minutes to 11.04 minutes, while in PROC SQL, the system CPU time was 13.56 minutes to 17.11 minutes. Once again, the system memory was vastly in favor of Merge. However, it is worth noting the difference in system time when you pull only limited variables, vs. all variables.



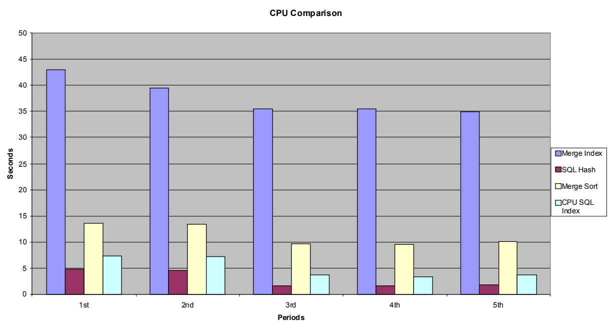
SQL HASH VS INDEXING

```
options fullstimer msglevel=i;
** Preparation **;

data sugi;
 set r.sugi;
run;

data a;
set sugi (keep=memcode);
if uniform(1)<.001
run;

proc sql _method;
create table mem as
select distinct memcode
from a
order by 1;
quit;

** Hash Join **;
 proc sql _method ;
         create table smeth as
         select s.*
         from sugi s , mem m
         where s.memcode=m.memcode;

NOTE: SQL execution methods chosen are:

      sqxcrta
          sqxjhsh
              sqxsrc( WORK.SUGI(alias = S) )
              sqxsrc( WORK.MEM(alias = M) )
NOTE: Table WORK.SMETH created, with 23962 rows and 19
columns.

** Index Join **;
proc datasets nolist;
 modify sugi;
 index create memcode;
quit;




proc sql _method ;
```

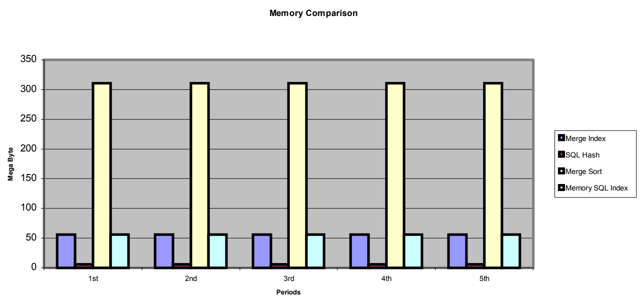
```
create table s2 as
select s.*
from sugi s , mem m
where s.memcode=m.memcode;
quit;
Note :
sqxcrta
     sqxjndx
       sqxsrc( WORK.MEM(alias = M) )
       sqxsrc( WORK.SUGI(alias = S) )
```

DATASTEP SORT MERGE VS INDEX MERGE

```
** Data Step **;
** Index only **;
data dstep;
 merge sugi mem(in=m);
 by memcode;
 if m;
run;
**  Sort **;
proc sort data=sugi force;
 by memcode;
run;
data d2;
 merge sugi mem(in=m);
 by memcode;
 if m;
run;
```



In this example, we compared the hash join and indexed join of PROC SQL and the sort merge and indexed merge of data step. To then effectively compare the CPU time, we included in our study the time for SAS to sort the table or create the index. As you can see, hashing took less CPU Time than the Merge step.



	1st	2nd	3rd	4th	5th
CPU Merge + Index	42.89	39.51	35.5	35.5	34.84
CPU Merge + Sort	13.6	13.49	9.73	9.59	10.14
CPU SQL Hash	4.86	4.66	1.77	1.76	1.8
CPU SQL + Index	7.43	7.26	3.83	3.32	3.76
Memory Index	56.051	56.051	56.051	56.051	56.051
Memory Merge sort	310.848	310.848	310.848	310.848	310.848
Memory SQL Hash	0.491	0.491	0.491	0.491	0.491
Memory SQL Index	56.036	56.036	56.036	56.036	56.036

* CPU is measured in seconds and memory is displayed in MB.

CONCLUSION

Although this paper compared PROC SQL vs. Merge and the effect of Hashing it was not intended and will not recommend any one procedure over the other. The user needs to understand that their platform may not yield the same results as our test. However, it should be noted that our results compared to SAS benchmarks were remarkably consistent (1). The objective of this paper is to familiarize the reader with all techniques and allow the user the knowledge to create an effective and efficient program. To us, an efficient program is more than the output. An efficient program allows for the data to be pulled concisely, without straining the resources available, and allows for debugging to take place. We recommend the user test their data and understand that there are always shortcuts or better techniques available. Finally, the user should be aware of external factors effecting the data combinations. As always, we welcome any comments or feedback the programmers can provide.

ACKNOWLEDGMENTS

The authors wish to thank all their fellow co-workers in the Health Care Economics team at Oxford Health Plans. We are honored to work with such a bright, energetic group of people who make coming to work both fun and interesting.

REFERENCES

1. “Advanced SAS Programming Techniques and Efficiencies” Course Notes. 1999 SAS Institute Inc. Cary NC. Page 61.

CONTACT INFORMATION

Kevin J. Smith
Oxford Health Plans
48 Monroe Turnpike
Trumbull, CT 06611
(203) 459-6145
kevsmith@oxfordhealth.com
www.oxfordhealth.com