PharmaSUG 2014 – Paper BB20

Efficient SQL for Pharma… and Other Industries

Chris Olinger, d-Wise, Morrisville, NC

ABSTRACT

The SQL Procedure has been used for many years by many different types of SAS® programmers. This paper delves into how PROC SQL processes internally, and how the user can exploit this information to improve performance with larger data sets. We will go over common tricks such as use of _method, as well as talk about views, updates, and the implications of sorting and IO “hints” that you can specify to PROC SQL to perform better. This talk is intended for persons already familiar with PROC SQL, SAS configuration options, and that have used PROC SQL in a clinical setting.

INTRODUCTION

Most SAS programmers have at one time used PROC SQL to manipulate data (and we assume that you have in this paper). What most clinical programmers and IT shops may not know is that PROC SQL performance is strongly tied to SAS configuration options and the tricks you apply in your code. Over the years, our company has done many assessments where performance of the SAS system has been evaluated – and almost always, clinical programmers are upset at the performance of their systems. With the advent of the Virtual Machines (VMWare/HyperV), IT departments are crowding more and more SAS sessions onto over-utilized hardware stacks and disk storage systems, and they are looking for clinical programmers to apply code optimizations that will enable the already stressed hardware to last longer.

The reasons for performance problems with respect to PROC SQL (and the DATA Step) usually lie in only one area – IO. It is rare in clinical programming that a SQL job is actually compute bound, so SQL performance enhancements are usually tailored to minimizing IO reads and writes. With that said, this paper looks at a list of IO do’s and don’ts that you should be aware of when trying to optimize your SAS and SQL code. These tips run from the very obvious to the more subtle. If you are concerned about SQL performance then you should make sure that each of these tips below has been implemented and understood – either by you in your code or by your IT and/or configuration specialist(s).

We cover the following in this paper:

- Sequential reads and writes vs. indexes
- Knowing where your data is located
- Making sure that specific SAS IO options have been configured
- Investigating _METHOD and MAGIC and the implications for the different SQL optimizer methods
- DATA Set options that can be used to minimize internal sorts
- DATA Set options that can be used to force the use of an Index
- Using SQL Views to minimize data passes
- Use PROC SORT instead of ORDER BY to restrict utility space usage
- Fixing lazy UPDATE’s
- Trusting the SQL optimizer?
- Use the most recent version of SAS and newer technologies

Each of these items, whether simple or not obvious, can have a positive effects on the run time performances of your SAS code.

SEQUENTIAL READS AND WRITES VS. INDEXES

To understand SQL you need to understand how SAS processes data sets. SAS is different from databases in that each SAS table is one or more separate contiguous files. These files are usually read in a sequential fashion, reading blocks of data one after the other from the file system. There are various SAS options to control how data is read, and there is a great deal of leeway in configuring the system in order to optimize how these blocks both are written and read. Indexes are separate files that are tied to the data set file. They work by storing key location offsets, and allow SQL and other procedures to skip to the appropriate blocks of data to read, bypassing a full sequential reads.

When SQL employs a full table scan it can be very expensive – especially in scenarios where the IO layer has not been optimized. You can usually tell if you have an IO problem just by just looking at the SAS log. Look at the “real time” vs. the “cpu time.” In the cases where the system is not IO bound you will see the real time coming close to matching the CPU time. If the system is IO bound then real time will outstrip CPU time. This discrepancy is due to the CPU pausing while the IO subsystem delivers the requested data. This metric is the first thing you should look at when evaluating performance. Note, there are other conditions where the real time will exceed the CPU time (or the CPU time will exceed the real time), but the vast majority of timing discrepancies are IO related.

```
51   data subjects;
52   set drive.dm;
53   where usubjid='1013-036-060-008';
54   run;

NOTE: There were 1 observations read from the data set DRIVE.DM.
      WHERE usubjid='1013-036-060-008';
NOTE: The data set WORK.SUBJECTS has 1 observations and 21 variables.
NOTE: DATA statement used (Total process time):
      real time           0.61 seconds
      cpu time            0.00 seconds
55
56   proc sql;
57   create table pass as
58   select lb.*
59     from drive.lb lb,
60          subjects s
61    where subjects.usubjid=lb.usubjid
62   ;
NOTE: Table WORK.PASS created, with 234 rows and 43 columns.
63   quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           2:40.52
      cpu time            0.28 seconds
```

In the above code, notice the discrepancies in the timings. The discrepancy is due to an IO mismatch.

When an index is involved, SQL and other procedures use the information in the index to move directly to the blocks of relevant data without scanning the full file (assuming high cardinality):

```
103  proc sql;
104  create table pass as
105  select lb.*
106    from drive.lb lb,
107         subjects s
108   where subjects.usubjid=lb.usubjid
109  ;
INFO: Index USUBJID of SQL table DRIVE.LB (alias = LB) selected for SQL WHERE clause
      (join) optimization.
NOTE: Table WORK.PASS created, with 234 rows and 43 columns.
110  quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           4.51 seconds
      cpu time            0.01 seconds
```

There are times when an index does not help, but the lesson here is that if you can minimize block level reads then you will increase the performance of your SQL code. This lesson is actually amplified when you are joining large tables that do not fit entirely into memory. In this case, utility files are created in the SAS work area. These utility files are written to and read from to handle inner sorts and joins. In this case, the plan chosen by the SQL optimizer is very important. If you are trying to minimize reads and writes then you sometimes need to help SQL choose the best plan.

KNOW WHERE YOUR DATA IS LOCATED

We start with the simplest thing you can do to improve the performance of your SQL – move the SAS execution as close to the data as possible. We have done assessments in the past where programmers were complaining about the performance of PROC SQL, only to discover that the drive that they were reading data from was located in another region. When asked about this, the response was “IT set up the drive mapping and we are not supposed to copy the data to our local machines.” In this case the user was running from home across a cable modem, and the throughput was not nearly enough to perform adequately. The code was using a WHERE statement to subset on a subject id, but what she did not realize is that the WHERE statement executes locally, and that in fact, every time she ran the code, she was re-copying the entire table to her local SAS session for processing. She may not have thought she was bringing the data to her machine, but in fact was doing it over and over again. In this case, you are more than likely better off running SAS on a machine that is running next to the data and creating a remote desktop session.

A similar phenomenon occurs when you inadvertently mix tables from two disparate locations in the same query. For instance, if one table is local to your SAS session and another table is large and stored in a database then SAS may not be able to optimize (due to code that cannot be passed to the database) and you will get terrible results. For instance:

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
libname oracl oracle user=chris pass=password path=clindb;
libname local "z:\camd-data\camd\1013";

proc sql;
create table labs as
select d.ARM, l.*
  from local.dm d, oracl.lb l
 where d.usubjid=l.usubjid and input(lbdtc,is8601da.)='13Oct2004'd
;
quit;

ORACLE_1: Prepared:
SELECT * FROM LB

ORACLE_2: Prepared:
SELECT  "STUDYID", "LBBLFL", "LBCAT", "LBDRVFL", "LBDTC", "LBDY", "LBELTM", "LBENDTC",
"LBFAST", "LBGRPID", "LBLOINC", "LBMETHOD", "LBNAM",
"LBNRIND", "LBORNRHI", "LBORNRLO", "LBORRES", "LBORRESU", "LBREASND", "LBREFID",
"LBRFTDTC", "LBSCAT", "LBSEQ", "LBSPCCND", "LBSPEC",
"LBSPID", "LBSTAT", "LBSTNRC", "LBSTNRHI", "LBSTNRLO", "LBSTRESC", "LBSTRESN",
"LBSTRESU", "LBTEST", "LBTESTCD", "LBTOX", "LBTOXGR",
"LBTPT", "LBTPTNUM", "LBTPTREF", "SUBJID", "USUBJID", "VISITNUM" FROM LB

ORACLE_3: Executed:
SELECT statement  ORACLE_2
```

In this case, SAS can only copy the full table from Oracle and process the query locally on the SAS side because Oracle does not understand the input() function. The first statement is a debug setting that will print the implicit pass through statements that are being sent to Oracle. You should always include this to see what SAS is doing under the covers.

MAKE SURE THAT SPECIFIC SAS IO OPTIONS HAVE BEEN CONFIGURED

Let's face it - not every clinical IT department in the world actually understands SAS. And this means that there is a decent chance that some of the basic SAS configuration options have not been changed from the defaults. As mentioned earlier, SAS data sets are typically read sequentially and this means that the proper IO settings should be configured in order to speed up data set access. SAS typically recommends that the following options be set, depending on the size of the data sets you will be accessing:

-**work** – specifies the folder location that SAS work tables are written to. This area should be on a separate disk channel from the permanent SAS data area used for inputs and outputs. The use of SSD for the SAS work area can have dramatic performance implications for your system. If SSD is not available then high performance disks running RAID 10 should be used. SAS work is temporary storage and should be as fast as possible.

-**utilloc** – specifies the area that SAS utility files are written to. This area should be on a separate disk channel from the permanent SAS data area used for inputs and outputs. This location is similar to the work location in requirements for disk write speeds. This location is where all temporary utility files for SQL processing will be written to.

-**memsize** – controls the total amount of memory that a SAS job can use. Think of this option as the high water mark for memory consumption that is used by SAS. It should be set to some value that is smaller than the total amount of memory on the system. The default of 0 says to use all of the available memory on the machine. At a minimum, memsize should be set to 512M and should be at least double the sortsize parameter.

-**sortsize** – controls the amount of memory available for the SORT procedure to use. The higher this value is, the more likely that sorts (both PROC SORT and SQL ORDER BY statements) can be done in memory without using the disk as a backing store. At a minimum, sortsize should be set between 256M and 512M.

-**bufsize** – controls the permanent buffer page size of data sets created by SAS. This option allows SAS datasets to be written with larger page sizes thus reducing the amount of physical reads that must occur when a data set is read. Setting this value to 64K is recommended for general mixed data set sizes where there is a decent amount of larger files to be read. Note, this parameter is applied when a file is written.

-**ubufsize** – controls the permanent buffer page size of utility files created by SAS. This option allows SAS utility files to be written with larger page sizes thus reducing the amount of physical reads that must occur when a utility file is read. Setting this value to 64K is recommended.

-**ibufsize** – controls the permanent buffer page size of index files created by SAS. This option allows SAS index files to be written with larger page sizes thus reducing the amount of physical reads that must occur when an index file is read. Setting this value to 32K is recommended.

-**bufno** – controls the number of data set buffers to keep resident in memory at any given time. Increasing this value affects memory consumption, and the amount of disk reads, writes and updates that will occur.

-**ibufno** – controls the number of utility file buffers to keep resident in memory at any given time. Increasing this value affects memory consumption, and the amount of disk reads, writes and updates that will occur.

-**ubufno** – controls the number of buffers to keep resident in memory at any given time. Increasing this value affects memory consumption, and the amount of disk reads, writes and updates that will occur.

The following are recommend settings from SAS (which can be optimized for individual installs. Some options are only valid for 9.2 or 9.3 and above):

MEMSIZE	512M	
SORTSIZE	256M	
BUFSIZE	64K	(depends on average sizes of data sets that will be processed)
UBUFSIZE	64K	(min)
IBUFSIZE	32767	
BUFNO	10	
UBUFNO	10	
IBUFNO	10	
ALIGNSASIOFILES		(9.3+ only)

In practice, we have seen increases in performance of roughly 15%-30% just by setting the options above.

INVESTIGATE _METHOD AND MAGIC

The SQL procedure has a number of built in options that let you see what it is doing under the covers. Chief among these are _METHOD and, in Versions 9.3+, MAGIC. _METHOD prints the selected join method and other details to the SAS log, and MAGIC provides a mechanism to inform the optimizer with hints as to which method to choose. There are a number of white papers available that talk about _METHOD and MAGIC (just Google "SAS SQL _METHOD MAGIC") so we will not repeat the fundamentals here. What we will talk about is how the main join criteria can impact performance of a SQL step. Consider the following query:

```
proc sql _method;
create table pass as
select s.ARM, lb.*
  from drive.lb lb,
       subjects s
 where subjects.usubjid=lb.usubjid and lb.usubjid='1013-036-060-008'
;
quit;
```

```
NOTE: PROC SQL planner chooses merge join.
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjm
              sqxsort
                  sqxsrc( WORK.SUBJECTS(alias = S) )
              sqxsort
                  sqxsrc( DRIVE.LB(alias = LB) )

NOTE: Table WORK.PASS created, with 234 rows and 44 columns.
```

Note the log output that is created. _METHOD is telling you exactly what it is going to execute as part of the query. In this case, the SQL statement is going to create a table (sqxcrta) by sourcing rows from two different tables (sqxsrc), each of which will be sorted internally (sqxsort), and then joined using a merge join (sqxjm). The universe of available methods is described here:

- Sqxcrta – Create a table from a select
- Sqxslct – Select statement
- Sqxjsl – Cartesian step loop join
- Sqxjm – Merge join
- Sqxjndx – Index join
- Sqxjhsh – Hash join
- Sqxsort – Sort
- Sqxsrc – Source or list rows from a table
- Sqxfil – Filter rows from a table
- Sqxsumg – Summary statistic using group by
- Sqxsumn – Summary statistics without a group by

For our purposes, we care most about any method that causes SQL to process or reprocess a file. Remember, we want to minimize the amount of IO that SQL is performing. In the example above, we notice that we have two internal sorts before the two files are merged. Each sort will read and write keys to the disk in the utility file area. To deal with this we can employ a few different strategies:

- Encourage SQL to use a different method
- Tell SQL that the data is already sorted
- Add indexes to the tables

You can give hints to the SQL optimizer using the MAGIC option (most functional in 9.3+). In an ideal situation, a Hash Join would be used. Hash joins are supposed to be chosen when the tables to be joined are smaller and can fit in memory effectively. In 9.4 Hash Joins are a much more common outcome from the optimizer due to improvements made by SAS R&D. In earlier releases, however, Hash Join sightings were akin to Yeti sightings. So, the version of SAS you are running is important!

Adding MAGIC=103 to the invocation shows us reverting to a Hash Join:

```
117  proc sql _method magic=103;
118  create table pass as
119  select s.ARM, lb.*
120    from drive.lb lb,
121         subjects s
122   where subjects.usubjid=lb.usubjid and lb.usubjid='1013-036-060-008'
123  ;
NOTE: PROC SQL planner chooses sequential loop join.
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjhsh
              sqxsrc( WORK.SUBJECTS(alias = S) )
              sqxsrc( DRIVE.LB(alias = LB) )
NOTE: Table WORK.PASS created, with 234 rows and 44 columns.
```

Another route is to inform SQL that the data is already sorted, thus bypassing the internal sorts. We will talk more about this technique later, but here is a sample that pre-sorts the data and shows a merge join without any inner sorts:

```
129  proc sort data=subjects; by usubjid; run;
130  proc sort data=drive.lb out=lb; by usubjid; run;
132  proc sql _method;
133  create table pass as
134  select s.ARM, lb.*
135    from lb lb,
136         subjects s
137   where subjects.usubjid=lb.usubjid and lb.usubjid='1013-036-060-008'
138  ;
NOTE: PROC SQL planner chooses merge join.
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjm
              sqxsrc( WORK.SUBJECTS(alias = S) )
              sqxsrc( WORK.LB(alias = LB) )
NOTE: Table WORK.PASS created, with 234 rows and 44 columns.
```

Lastly, you can add indexes to the various tables and use MAGIC or IDXNAME to force the use of the index. We will cover this later.

USING DATA SET OPTIONS TO MINIMIZE INTERNAL SORTS

In the previous section we described _METHOD and MAGIC. It was noted that Merge Joins will tend to do internal sorts in utility space in order to prepare subsets of the data for merging. This extra IO can kill the performance of SQL for large tables. Enhancements to 9.4 with respect to Hash Joins is a very welcome update, but for very large tables, it is almost guaranteed that the optimizer will default to a Merge Join if an index is not available or an index is not chosen. To deal with this you can pre-sort data and maintain this order with implied SORTEDBY assertions or add an index.

When data is sorted by PROC SORT or by using an ORDER BY statement in SQL then resultant data set is tagged with what is known as a hard sort assertion. This assertion is a flag on the physical file that says that the data is sorted by the specified criteria. SQL will not choose to an inner sort if the criteria shows that the data is pre-sorted. However, if it is the case that a data set has lost this assertion then you may need to tell SQL to treat the data as if it were sorted anyways. Consider the following:

```
proc sort data=drive.dm out=subjects; by usubjid; run;
proc sort data=drive.lb out=lb; by usubjid; run;

data lb2;
set lb;
processedDate=date();
format processedDate date9.;
run;

proc sql _method;
create table pass as
select s.ARM, lb.*
  from lb2 lb,
       subjects s
 where subjects.usubjid=lb.usubjid and lb.usubjid='1013-036-060-008'
;
quit;
```

In this case, the sort assertion was lost when we made a copy of the Lab data set. This translates into an inner sort for the Merge Join that is actually not necessary – as we know that the data is still sorted. As such, we wind up with the following optimizer plan:

```
NOTE: PROC SQL planner chooses merge join.
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjm
              sqxsrc( WORK.SUBJECTS(alias = S) )
              sqxsort
                  sqxsrc( WORK.LB2(alias = LB) )
```

We can remove the inner sort just by telling SQL that the data is in fact still sorted:

```
proc sql _method magic=102;
create table pass as
select s.ARM, lb.*
  from lb2(sortedby=usubjid) lb,
       subjects s
 where subjects.usubjid=lb.usubjid and lb.usubjid='1013-036-060-008'
;
quit;
```

```
NOTE: PROC SQL planner chooses merge join.
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjm
              sqxsrc( WORK.SUBJECTS(alias = S) )
              sqxsrc( WORK.LB2(alias = LB) )
```

In practice, this type of scenario occurs all of the time. If your data is really not sorted by the criteria then SQL will catch it and throw an error. You can also use this trick to force the usage of an index on a table that the SQL optimizer has ignored – just use the IDXNAME option to force the index to be used.

USING DATA SET OPTIONS TO FORCE THE USE OF AN INDEX

Indexes can improve the performance of SQL queries dramatically, or they can cause slowdowns. An index is akin to a map that points data references to the proper page and disk location. Instead of full table scans, an index enables SQL to jump directly to the indexed location on disk. Indexes tend to work well when the cardinality of the variable(s) being indexed is high. If the cardinality of the data is low then you are in effect indexing most of the rows and the value of the index goes down.

Note, an index maps different points in the data. If the indexed columns distinct values are scattered across the data set then the use of an index has the potential to cause many data pages to be swapped into memory when reading. Sorting the data by the variables that are to be indexed can help with this phenomenon. Paul Kent likens an index to a postman. If the postman must deliver mail to disparate parts of a city, jumping from one location to another, then the speed of delivery is significantly slowed. If the postman can deliver to one mailbox after another on the same street then the mail is delivered much more efficiently.

The use of an index is up to the SQL optimizer. It will look at the index and its attributes and make an estimation if its use will speed the join or not. You can check whether the index is used by using the following option:

```
options msglevel=i;
```

Note, if you are using a multi-column index then any WHERE statement that you use must use the same columns that were defined, in the correct order. Multi-column indexes in SAS are not supported as well as they should be. Regardless, SQL may choose not to use the index, and it seems to happen too often. You can force the use of an index on a join using the same technique as implying a sort order. Use IDXNAME in the data set options to force the use of an index when SQL is not cooperating:

```
options msglevel=i;
proc sql _method;
create table pass as
select s.ARM, lb.*
  from lb(idxname=usubjid idxwhere=usubjid),
       subjects s
 where subjects.usubjid=lb.usubjid
;
quit;
```

```
INFO: Index USUBJID of SQL table WORK.LB selected for SQL WHERE clause (join) optimization.
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjndx
              sqxsrc( WORK.SUBJECTS(alias = S) )
              sqxsrc( WORK.LB )
NOTE: Table WORK.PASS created, with 161526 rows and 44 columns.
```

You can also use IDXWHERE=YES to force the use of an index on a WHERE statement when SQL won't cooperate.

USE SQL VIEWS TO MINIMIZE DATA PASSES

If you find yourself writing multiple SQL steps that reference data created by previous steps then you may want to consider using SQL views. Using views in place of physical SQL allows the optimizer to weed out unused columns and to optimize utility file creation and management. As well, intermediate tables and files that may have been created will be managed and cleaned up after the SQL step is complete. In the case of large file manipulation, using cascading SQL views can significantly decrease IO time for your SAS job. Consider the following (contrived) example which uses the _TREE option to print the table plan to the SAS log. This example creates a copy of an existing table and then references the new table in the next SQL step:

```
proc sql _tree;
create table lb2 as
select lb.*,
date() as processedDate format=date9.
from drive.lb
;
create table pass as
select s.ARM, lb.processedDate
  from lb2 lb,
       drive.subjects s
 where subjects.usubjid=lb.usubjid and lb.usubjid='1013-036-060-008'
;
quit;
```

Below is the list of variables that are referenced by the two steps. Depending on the size of the data sets, the two steps can produce a significant amount of IO:

Table 1:

```
Tree as planned.
                               /-SYM-V-(lb.STUDYID:1 flag=0001)
                     /-OBJ----|
                    |         |--SYM-V-(lb.LBBLFL:2 flag=0001)
                    |         |--SYM-V-(lb.LBCAT:3 flag=0001)
                    |         |--SYM-V-(lb.LBDRVFL:4 flag=0001)
                    |         |--SYM-V-(lb.LBDTC:5 flag=0001)
                    |         |--SYM-V-(lb.LBDY:6 flag=0001)
                    |         |--SYM-V-(lb.LBELTM:7 flag=0001)
                    |         |--SYM-V-(lb.LBENDTC:8 flag=0001)
                    |         |--SYM-V-(lb.LBFAST:9 flag=0001)
                    |         |--SYM-V-(lb.LBGRPID:10 flag=0001)
                    |         |--SYM-V-(lb.LBLOINC:11 flag=0001)
                    |         |--SYM-V-(lb.LBMETHOD:12 flag=0001)
                    |         |--SYM-V-(lb.LBNAM:13 flag=0001)
                    |         |--SYM-V-(lb.LBNRIND:14 flag=0001)
                    |         |--SYM-V-(lb.LBORNRHI:15 flag=0001)
                    |         |--SYM-V-(lb.LBORNRLO:16 flag=0001)
                    |         |--SYM-V-(lb.LBORRES:17 flag=0001)
                    |         |--SYM-V-(lb.LBORRESU:18 flag=0001)
                    |         |--SYM-V-(lb.LBREASND:19 flag=0001)
                    |         |--SYM-V-(lb.LBREFID:20 flag=0001)
                    |         |--SYM-V-(lb.LBRFTDTC:21 flag=0001)
                    |         |--SYM-V-(lb.LBSCAT:22 flag=0001)
                    |         |--SYM-V-(lb.LBSEQ:23 flag=0001)
                    |         |--SYM-V-(lb.LBSPCCND:24 flag=0001)
                    |         |--SYM-V-(lb.LBSPEC:25 flag=0001)
                    |         |--SYM-V-(lb.LBSPID:26 flag=0001)
                    |         |--SYM-V-(lb.LBSTAT:27 flag=0001)
                    |         |--SYM-V-(lb.LBSTNRC:28 flag=0001)
                    |         |--SYM-V-(lb.LBSTNRHI:29 flag=0001)
                    |         |--SYM-V-(lb.LBSTNRLO:30 flag=0001)
                    |         |--SYM-V-(lb.LBSTRESC:31 flag=0001)
                    |         |--SYM-V-(lb.LBSTRESN:32 flag=0001)
                    |         |--SYM-V-(lb.LBSTRESU:33 flag=0001)
                    |         |--SYM-V-(lb.LBTEST:34 flag=0001)
                    |         |--SYM-V-(lb.LBTESTCD:35 flag=0001)
                    |         |--SYM-V-(lb.LBTOX:36 flag=0001)
                    |         |--SYM-V-(lb.LBTOXGR:37 flag=0001)
                    |         |--SYM-V-(lb.LBTPT:38 flag=0001)
                    |         |--SYM-V-(lb.LBTPTNUM:39 flag=0001)
                    |         |--SYM-V-(lb.LBTPTREF:40 flag=0001)
```

```
          |         |--SYM-V-(lb.SUBJID:41 flag=0001)
          |         |--SYM-V-(lb.USUBJID:42 flag=0001)
          |         |--SYM-V-(lb.VISITNUM:43 flag=0001)
          |          \-SYM-A-(processedDate:1 flag=0031)
 /-FIL----|
|         |                    /-SYM-V-(lb.STUDYID:1 flag=0001)
|         |          /-OBJ----|
|         |         |         |--SYM-V-(lb.LBBLFL:2 flag=0001)
|         |         |         |--SYM-V-(lb.LBCAT:3 flag=0001)
|         |         |         |--SYM-V-(lb.LBDRVFL:4 flag=0001)
|         |         |         |--SYM-V-(lb.LBDTC:5 flag=0001)
|         |         |         |--SYM-V-(lb.LBDY:6 flag=0001)
|         |         |         |--SYM-V-(lb.LBELTM:7 flag=0001)
|         |         |         |--SYM-V-(lb.LBENDTC:8 flag=0001)
|         |         |         |--SYM-V-(lb.LBFAST:9 flag=0001)
|         |         |         |--SYM-V-(lb.LBGRPID:10 flag=0001)
|         |         |         |--SYM-V-(lb.LBLOINC:11 flag=0001)
|         |         |         |--SYM-V-(lb.LBMETHOD:12 flag=0001)
|         |         |         |--SYM-V-(lb.LBNAM:13 flag=0001)
|         |         |         |--SYM-V-(lb.LBNRIND:14 flag=0001)
|         |         |         |--SYM-V-(lb.LBORNRHI:15 flag=0001)
|         |         |         |--SYM-V-(lb.LBORNRLO:16 flag=0001)
|         |         |         |--SYM-V-(lb.LBORRES:17 flag=0001)
|         |         |         |--SYM-V-(lb.LBORRESU:18 flag=0001)
|         |         |         |--SYM-V-(lb.LBREASND:19 flag=0001)
|         |         |         |--SYM-V-(lb.LBREFID:20 flag=0001)
|         |         |         |--SYM-V-(lb.LBRFTDTC:21 flag=0001)
|         |         |         |--SYM-V-(lb.LBSCAT:22 flag=0001)
|         |         |         |--SYM-V-(lb.LBSEQ:23 flag=0001)
|         |         |         |--SYM-V-(lb.LBSPCCND:24 flag=0001)
|         |         |         |--SYM-V-(lb.LBSPEC:25 flag=0001)
|         |         |         |--SYM-V-(lb.LBSPID:26 flag=0001)
|         |         |         |--SYM-V-(lb.LBSTAT:27 flag=0001)
|         |         |         |--SYM-V-(lb.LBSTNRC:28 flag=0001)
|         |         |         |--SYM-V-(lb.LBSTNRHI:29 flag=0001)
|         |         |         |--SYM-V-(lb.LBSTNRLO:30 flag=0001)
|         |         |         |--SYM-V-(lb.LBSTRESC:31 flag=0001)
|         |         |         |--SYM-V-(lb.LBSTRESN:32 flag=0001)
|         |         |         |--SYM-V-(lb.LBSTRESU:33 flag=0001)
|         |         |         |--SYM-V-(lb.LBTEST:34 flag=0001)
|         |         |         |--SYM-V-(lb.LBTESTCD:35 flag=0001)
|         |         |         |--SYM-V-(lb.LBTOX:36 flag=0001)
|         |         |         |--SYM-V-(lb.LBTOXGR:37 flag=0001)
|         |         |         |--SYM-V-(lb.LBTPT:38 flag=0001)
|         |         |         |--SYM-V-(lb.LBTPTNUM:39 flag=0001)
|         |         |         |--SYM-V-(lb.LBTPTREF:40 flag=0001)
|         |         |         |--SYM-V-(lb.SUBJID:41 flag=0001)
|         |         |         |--SYM-V-(lb.USUBJID:42 flag=0001)
|         |         |          \-SYM-V-(lb.VISITNUM:43 flag=0001)
|         |--SRC----|
|         |          \-TABL[WORK].lb opt=''
|         |--empty-
|         |--empty-
|         |--empty-
|         |                    /-SYM-A-(processedDate:1 flag=0031)
|         |          /-ASGN---|
|         |         |          \-LITN(19829) DATE.
```

```
          |          \-ERLY---|
 --SSEL---|
```

Table 2:

```
Tree as planned.
                              /-SYM-V-(s.ARM:4 flag=0001)
                    /-OBJ----|
                   |          \-SYM-V-(lb.processedDate:44 flag=0001)
          /-JOIN---|
         |         |                              /-SYM-V-(s.ARM:4 flag=0001)
         |         |                    /-OBJ----|
         |         |                   |          \-SYM-V-(s.USUBJID:21 flag=0001)
         |         |          /-SRC----|
         |         |         |         |--TABL[WORK].subjects opt=''
         |         |         |         |          /-NAME--(USUBJID:21)
         |         |         |          \-CEQ----|
         |         |         |                    \-LITC('1013-036-060-008')
         |         |--FROM---|
         |         |         |                    /-SYM-V-(lb.processedDate:44 flag=0001)
         |         |         |          /-OBJ----|
         |         |         |         |          \-SYM-V-(lb.USUBJID:42 flag=0001)
         |         |          \-SRC----|
         |         |                   |--TABL[WORK].lb2 opt=''
         |         |                   |          /-NAME--(USUBJID:42)
         |         |                    \-CEQ----|
         |         |                              \-LITC('1013-036-060-008')
         |         |--empty-
         |         |          /-SYM-V-(s.USUBJID:21)
         |          \-CEQ----|
         |                    \-SYM-V-(lb.USUBJID:42)
 --SSEL---|
```

If we switch the first SQL step to a view instead of a table then the SQL optimizer will figure out and optimize variable and data set references:

```
proc sql _tree;

create VIEW lb2 as
select lb.*,
date() as processedDate format=date9.
from lb
;

create table pass as
select s.ARM, lb.processedDate
  from lb2 lb,
       subjects s
 where subjects.usubjid=lb.usubjid and lb.usubjid='1013-036-060-008'
;
quit;
```

This produces the following _tree debug output:

```
Tree as planned.
                              /-SYM-V-(s.ARM:4 flag=0001)
                    /-OBJ----|
                   |          \-SYM-A-(processedDate:1 flag=0031)
```

```
          /-JOIN---|
         |         |                              /-SYM-V-(s.ARM:4 flag=0001)
         |         |                    /-OBJ----|
         |         |                   |          \-SYM-V-(s.USUBJID:21 flag=0001)
         |         |          /-SRC----|
         |         |         |         |--TABL[WORK].subjects opt=''
         |         |         |         |          /-NAME--(USUBJID:21)
         |         |         |          \-CEQ----|
         |         |         |                    \-LITC('1013-036-060-008')
         |         |--FROM---|
         |         |         |                    /-SYM-V-(lb.USUBJID:42 flag=0001)
         |         |         |          /-OBJ----|
         |         |         |         |          \-SYM-A-(processedDate:1 flag=0031)
         |         |          \-FIL----|
         |         |                   |                    /-SYM-V-(lb.USUBJID:42 flag=0001)
         |         |                   |          /-OBJ----|
         |         |                   |--SRC----|
         |         |                   |         |--TABL[WORK].lb opt=''
         |         |                   |         |          /-NAME--(USUBJID:42)
         |         |                   |          \-CEQ----|
         |         |                   |                    \-LITC('1013-036-060-008')
         |         |                   |--empty-
         |         |                   |--empty-
         |         |                   |--empty-
         |         |                   |                    /-SYM-A-(processedDate:1 flag=0031)
         |         |                   |          /-ASGN---|
         |         |                   |         |          \-LITN(19829) DATE.
         |         |                    \-ERLY---|
         |         |--empty-
         |         |          /-SYM-V-(s.USUBJID:21)
         |          \-CEQ----|
         |                    \-SYM-V-(lb.USUBJID:42)
--SSEL---|
```

As you can see, only the necessary resultant columns have been referenced. The rest have been dropped from the processing vector. In practice, it is a good idea to leverage SQL (and Data Step) views, although there are times when you will see no gain in performance.

USE PROC SORT INSTEAD OF ORDERBY TO RESTRICT DISK SPACE USAGE

It is very convenient to use the ORDER BY statement in PROC SQL to sort created result sets. It is a single statement with a relatively terse syntax. However, if you are sorting very large (multi-GB) tables and you find yourself with constrained disk space in your utility areas then you should consider using PROC SORT. Both PROC SORT and ORDER BY use the same internal sorting routine. However, for whatever reason, ORDER BY tends to use roughly 4x the size of the data set to sort in the utility area. PROC SORT uses roughly 2x the size of the data to sort. The downside is that with PROC SORT you require a physical table, where ORDER BY will sort data contained in the utility files. However, if you are space constrained then your best bet is PROC SORT.

If space is not a concern then your best bet is to use ORDER BY as it does not require a physical copy of the table to function.

DON’T BE A LAZY UPDATE PROGRAMMER

The SQL UPDATE statement is a very handy way to update a SAS table. It updates in place, is similar to MODIFY, and does not rewrite the table. It can do correlated sub-queries for selecting subsets of rows, and it works especially well when there is an index in play, bypassing full table scans. Use of UPDATE can save on IO. However, if you are not using indexes and full table scans are being performed then you should take care not to simply use UPDATE statements because it is easy. Consider the following:

```
proc sql;
update t1 set name='Chris' where name is null;
update t1 set date=today() where date is null;
update t1 set state='NC'   where state is null;
quit;
```

In this case, we are updating multiple variables when they are NULL. Each statement has its own WHERE criteria, so you cannot merge the assignments on a single SET statement. This code is very simple to write, but inefficient to run as the code performs three full table scans. You should redo this to use a data step using a SET statement (or UPDATE, MERGE, MODIFY, etc…):

```
data t1;
set t1;
if missing(name)  then name='Chris';
if missing(date)  then date=today();
if missing(state) then state='NC';
run;
```

If the variables that you are referencing are all indexed then it may be that it is more efficient to spot update each of the variables directly – it really depends on the cardinality of the data, and whether or not the index is actually used (SQL will determine if a full table scan is more advantageous than using the index). You have to try it both ways to see which will work best for you.

DON'T TRUST THE SQL OPTIMIZER ALL OF THE TIME

I have run into this next phenomenon a couple of times in real world situations. In this case, the order of left joins actually matters to the optimizer even though the underlying sort order of the tables being joined has not changed. As an example, consider the following:

```
/* generated, already sorted by i, j, k */
data x;
do i = 1 to 100;
  do j=1 to 50;
    do k=1 to 10;
      lbl="X Y Z";
      output;
    end;
  end;
end;
run;

/* generated, already sorted by i, j */
data y;
do i = 1 to 100;
  do j=1 to 50;
    text="Hello: "||strip(put(i,best.))||","||strip(put(j,best.));
    output;
  end;
end;
run;

/* look at sorts */
proc sql _method;
create table foo as
select x.*, y1.text as T1, y2.text as T2
  from x
  left join y as y1 on x.i=y1.i
  left join y as y2 on x.i=y2.i and x.j=y2.j
;
quit;
```

```
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjm
              sqxsort
                  sqxsrc( WORK.Y(alias = Y2) )
              sqxsort
                  sqxjm
                      sqxsort
                          sqxsrc( WORK.Y(alias = Y1) )
                      sqxsort
                          sqxsrc( WORK.X )
NOTE: SAS threaded sort was used.
NOTE: Table WORK.FOO created, with 2500000 rows and 6 columns.
```

```
/* no sorts using asserted sortedby */
proc sql _method;
create table foo as
select x.*, y1.text as T1, y2.text as T2
  from x(sortedby=i j k)
  left join y(sortedby=i j) as y2 on x.i=y2.i and x.j=y2.j
  left join y(sortedby=i j) as y1 on x.i=y1.i
;
quit;
```

```
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjm
              sqxsrc( WORK.Y(alias = Y1) )
              sqxjm
                  sqxsrc( WORK.Y(alias = Y2) )
                  sqxsrc( WORK.X )
NOTE: Table WORK.FOO created, with 2500000 rows and 6 columns.
```

```
/* ack?! sort was added back in just by changing left join order! */
options SORTEQUALS;
proc sql _method;
create table foo as
select x.*, y1.text as T1, y2.text as T2
  from x(sortedby=i j k)
  left join y(sortedby=i j) as y1 on x.i=y1.i
  left join y(sortedby=i j) as y2 on x.i=y2.i and x.j=y2.j
;
quit;
```

```
NOTE: SQL execution methods chosen are:
      sqxcrta
          sqxjm
              sqxsrc( WORK.Y(alias = Y2) )
              sqxsort
                  sqxjm
                      sqxsrc( WORK.Y(alias = Y1) )
                      sqxsrc( WORK.X )
NOTE: SAS threaded sort was used.
NOTE: Table WORK.FOO created, with 2500000 rows and 6 columns.
```

The lesson here is to always use _method to see what your code is actually doing. Do not just assume that the optimizer is doing the right thing. You may have inner sorts that you don't know about, and all you did was clean up your code some.

USE THE MOST RECENT VERSION OF SAS

SQL is constantly being improved. The most recent versions of SAS have optimizations that enable Hash Joins to be used more often without special tricks, optimizations regarding pass through to databases, new procedures that fix old problems (such as lack of true DB data types), and other languages that manipulate data. Notable in 9.4 are a production DS2 language, PROC FEDSQL (Federated SQL), and more thorough IN-DB processing routines.

Also, it must be said that there are certain operations that really should be performed with the data step. Case in point – determining the first or last record of a by group. For SQL, this is a messy operation, with extra variables needed to mark the start and end of the by group (remember, with SQL order of records is not guaranteed). SQL can use max() and min() to determine the minimum and maximum values. But, in the end, there is no substitute for data step operations first dot and last dot.

Remember, SQL is just another tool. SAS is one of the most flexible data processing tools out there. You should be experienced in multiple processing techniques.

CONCLUSION

There are many techniques that will help you optimize your SQL queries. SAS data sets are stored in distinct files on a file system, requiring that you understand how to optimize file IO. Each technique presented in this paper is intended to either to optimize or reduce IO between SAS and the file system.

REFERENCES AND ACKNOWLEDGMENTS

Any and all papers by Kirk Lafler, Robert Ray, and Paul Kent regarding PROC SQL ☺

CONTACT INFORMATION

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

Name:	Chris Olinger
Enterprise:	d-Wise
Address:	1500 Perimeter Park Dr, Suite 150
City, State ZIP:	Morrisville, NC 27560
Work Phone:	919-600-6235
E-mail:	colinger@d-wise.com
Web:	http://www.d-wise.com
Twitter:	@saswise