Paper 309-2009

Top Ten SAS® DBMS Performance Boosters for 2009

Howard Plemmons, SAS Institute Inc., Cary, NC

ABSTRACT

Gleaned from internal development efforts and SAS® technical support, this paper tracks the top ten performance options, code snips, and processes to increase your access speed to your data. Focus will be around both DATA step and solution performance enhancements against a variety of database management systems (DBMSs).

INTRODUCTION

It has been debated that performance and portability are orthogonal. I feel that the point where these two cross can be shown on a graph with software and/or usage techniques that are portable in nature. For example, some of the options discussed are global to SAS and can be executed on many different platforms. Some of the DBMS options discussed are portable in concept even though DBMS specific options or techniques are necessary. I offer ten performance boosters, unordered, which are comprised of code, processes, and techniques.

READBUFF

READBUFF is a SAS/ACCESS® LIBNAME engine option that tells the DBMS how many rows of data are to be read, buffered, and sent to the DBMS client, per read. It is an option that has been recommended by SAS technical support for many years. Its performance boost is gained by reduced network traffic when reading large amounts of data (for example, if you have a table of 10 billion rows and intend to consume a great deal of the data during an analysis phase of your program). Without READBUFF, you would consume this data one row at a time, which would produce additional network traffic every time a row is read from the DBMS. Your multiplier would be rows x network overhead for the read request sent to the DBMS. When you use READBUFF, your cost is rows/readbuff x network overhead. The performance gain can be significant. Hint – READBUFF consumes memory to hold the buffered rows (memory size = readbuff x row_size). Therefore, when doing a READBUFF calculation, make sure you have sufficient memory to hold that many rows; otherwise, performance may degrade by sending data to your hard drive.

BULKLOAD

BULKLOAD is invoked using a set of SAS/ACCESS LIBNAME engine and data set options. These options are used to trigger DBMS bulk loading utilities. These utilities permit you to send data from SAS to the DBMS at a very fast rate. The performance gains that can be realized using bulk loading can be significant. It is important to note that resources used by the DBMS should be considered when using BULKLOAD. For instance, if there are a limited number of BULKLOAD sessions, then your use of this should be reserved for appropriately sized tables. Some DBMS bulk loaders may also degrade performance on small tables (i.e., it is more cost efficient to load small tables using the normal insert process). Reference sections in the SAS/ACCESS documentation detail bulk loading options and techniques for your database.

IMPLICIT PASSTHRU UPDATES

Implicit Passthru modifications in PROC SQL are designed to improve pushdown to the DBMS. The two code examples below illustrate the process improvements underway in PROC SQL. A comparison is made to the PROC SQL processes currently in production. The basic improvement is how SAS SQL, when parsed and textualized correctly, could be executed in your DBMS. Without these performance improvements, the data would be extracted and processed in SAS. These changes should be available soon as a SAS® 9.1.3 and SAS® 9.2 fix.

The following is sample SAS code that identifies the SQL improvements:

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;
libname x teradata …;

/*--- create test tables ---*/
data x.tabs01;
num=1; num2=1; output;
num=3; num2=2; output;
```

```
run;

data x.tabs03;
dept=1; division=10; output;
run;

data x.tabs04;
empnum=1; state='NC'; output;
run;

/*--- sql with "issues" ---*/
proc sql;
    create table work.out as
    select * from (select c.dept,
           (select num as newnum from x.tabs01 where newnum=1)
                 as col2 from x.tabs03 c) qu1,
                 x.tabs04 d where qu1.col2 = d.empnum and qu1.col2 > 0;
quit;

options sastrace=off;
```

Without Implicit Passthru changes, the code above would not be parsed and textualized correctly and could not be passed to the DBMS for processing. The data would have to be extracted and processed in SAS.

```
ACCESS ENGINE: SQL statement was not passed to the DBMS, SAS will do the processing.
```

With Implicit Passthru changes, the following would be parsed and textualized correctly. The SQL can then be sent to the DBMS for processing. Note that the entire statement would be processed inside the DBMS. In this case, it would result in a significant performance gain.

```
/*--- snip from the debug output ---*/
select qu1."DEPT", qu1.COL2, d."EMPNUM", d."STATE" from
(select c."DEPT"(
select TABS01."NUM" as NEWNUM from TABS01 where
TABS01."NUM" = 1) as COL2
from TABS03 c)qu1, TABS04 d where (qu1.COL2 =
d."EMPNUM") ad (qu1.COL2>0)

ACCESS ENGINE: SQL statement was passed to the DBMS for fetching data.
```

PUT

There is a class of components called SAS formats. These SAS formats cannot be parsed, textualized, and passed to the DBMS since databases do not intrinsically have equivalent functionality. For example, if you generate SAS SQL using SQL generators or with PROC SQL and the resulting SQL code contains formats, then it would not go to the DBMS for processing. This could lead to excess data being pulled back to SAS for processing, which would degrade performance. Several steps have been taken to transform the SAS PUT into SQL that can be processed by the DBMS. The example below shows the transformation process:

```
/*--- create a UDF - SAS User-Defined Formats --*/
proc format;
 value fastcar 1-45='slow' 46-75='lawful' 76-90='commuter' 91-125='ticket';
run;

libname x db2 …;
proc sql;
select name, address, phone from x.baddriver
    where put(speed,fastcar)='ticket';

/*--- the SQL passed to the DBMS would resemble ---*/
<select name, address, phone from baddriver where 91 <= speed <= 125>
```

UTILITIES

BULKLOAD is one utility that exists in many DBMSs that SAS/ACCESS can support. There are other utilities that are supported by SAS/ACCESS. Using these utilities can provide significant performance increases for loading, re-loading, and adding to existing data. For example, there are Teradata utilities that are surfaced to SAS procedures, applications, and solutions. They are invoked by using the options and processes shown below:

Multiload

```
/*--- enable multiload use on subsequent SAS output requests ---*/
libname x teradata …;
data x.test1 (multiload=yes);
do I = 1 to 1000;
   a = I * 15;
   b = a/5;
   c = 'test';
   output;
end;
run;

/*--- add some more data to the table ---*/
proc append data=a base=x.test1 (multiload=yes);
run;
```

Fastexport

```
/*--- enable fastexport then use it ---*/
libname x teradata … fastexport=yes;

data work.a;
  set x.test1;
run;

/*--- the following note appears in the SAS log ---*/
NOTE:  Teradata connection:  TPT FastExport has read n row(s).
```

The use of Teradata utilities and associated options is described in the SAS/ACCESS to Teradata documentation. You should reference this user's guide when working with Teradata utilities.

DATA STEP MODIFY

DATA step MODIFY may be an alternative process, in some cases, where it makes sense to extract and let SAS do the processing. The code below identifies a SAS customer process that did not work well with PROC SQL running against the DBMS. SAS technical support provided the DATA step MODIFY code to help them overcome the SAS performance issue they were having.

The following is a PROC SQL code example:

```
libname dblib db2 …;
proc sql;
update dblib.employee a
set name=(select name from upd b where a.empid=b.empid),
    dept=(select dept from upd b where a.empid=b.empid),
    salary=(select salary from upd b where a.empid=b.empid)
    where empid in (select empid from upd);
quit;
```

The code example above will produce multiple selects and updates of the data to satisfy the query, resulting in multiple passes of the data. Given the size of the data, this could degrade performance.

The following DATA step MODIFY code example will make one pass of the data. The performance implications, depending on data size, can be significant.

```
data dblib.employee;
```

```
modify dblib.employee upd;
by empid;
select (_iorc_ );
    when(%sysrc(_sok))
           replace;
    when(%sysrc(_dsenmr)) do;
           _error_ =0;
        /* other types of processing here - if you want to add the record make sure
           reread_exposure=yes is set on the libname statement */
        /* if a non-match is a problem then you may want to put out a note */
        put 'No match was found for'empid=;
    end;
    otherwise do;
        put 'Unexpected error';
        put _all_;
    end;
run;
```

This particular customer and the SAS technical support interaction is a reminder that tuning with basic SAS components can increase performance as well.

FUNCTIONS

Passing functions to the DBMS for processing is a very important concept. To enable the pass down of a SAS function requires that it must map to a DBMS function. By using this mapping process, more of your SQL can be parsed, textualized, and sent to the DBMS for processing. There are two types of function mappings to consider. First, an internal SQL dictionary in each SAS/ACCESS engine is consulted to verify and validate that a SAS function can be mapped to a DBMS function. The second is the capability to modify the SQL dictionary, at run time, to add additional SAS functions and DBMS mappings to the internal SQL dictionary. The process example below provides the conceptual details:

SAS IP SQL using functions

```
libname x oracle …;
proc sql;
    select name, address, phone from x.clients
           where upcase(country)='USA';
quit;
```

Internal SAS/ACCESS Dictionary contents

```
.
.
upcase upper
.
.
```

The internal dictionary in the ACCESS engine maps the SAS function to the corresponding DBMS function. Other metadata items allow for parameter switching when the functions are mapped. The output from processing against the SQL dictionary is an SQL statement that can be processed by the DBMS. In the case above, the SAS upcase function is translated into the Oracle upper function.

Additional functions are added to the internal SAS/ACCESS dictionary at run time.

There is an opportunity at run time to add additional function mappings to the SQL dictionary in the ACCESS engine. These functions are supported for the duration of the LIBNAME statement. The code example below shows the dynamic option mapping process:

```
/*--- get a copy of the SQL functions dictionary from the engine ---*/
libname x oracle … sql_functions_copy=work.a;

/*--- modify the data to all your new function mappings (SAS -> DMBS) ---*/

/*--- add your functions to the dictionary for runtime processing ---*/
```

```
libname y oracle … sql_functions="external_append=work.a";

/*--- use the functions in PROC SQL IP code ---*/
```

See the SAS/ACCESS documentation and search for SQL_FUNCTIONS. You will see the definition for the SQL dictionary structure and other features and limitations across DBMS.

IN-DATABASE PROCESSING, BASE PROCEDURES

The process of running SAS procedures in-database, in some cases, is the merging of DBMS SQL into SAS procedure logic. These in-database enabled procedures provide a performance boost by running aggregations in the database. The aggregated result in specific cases is extracting and processing a smaller subset of data in SAS, which can increase performance significantly. The sample code below, running against Teradata, has shown significant performance gain:

```
libname x teradata …;
proc freq data=x.big_table; run;
```

The current PROC FREQ would perform a data extraction and calculate the table statistics as the data is read. The in-database version, running against an enabled database, would issue SQL to perform the initial table aggregation. The resulting aggregated data would be extracted into the PROC FREQ, and procedure processing would continue. The user would see the same behavior, functionally; however, they can also see drastic performance increases with the in-database version.

SQL PROCESS

Adding additional SQL processes in SAS infrastructure adds performance value by pushing more into the database for processing to help control data extraction. Some scenarios below show how SQL processes have impacted specific performance issues:

Create, Delete – direct pushdown to the DBMS

The code snip below shows an Implicit Passthru SQL statement that can be parsed and textualized into DBMS SQL for processing in the DBMS. The key is to specify the dbidirectexec option to enable SQL events that can be passed through to the DBMS. Also note the sastrace option, which will show you the SQL that has been pushed down.

```
libname x teradata …;
options sastrace=',,,d' nostsuffix;
options dbidirectexec;
proc sql;
create table x.newtab as select * from x.oldtab;
delete * from x.newtab where custid > 1010;
quit;
```

Other SQL process improvements are consumed by SAS solutions. For example, the process identified above is now a part of SAS® Data Integration Studio. Other SQL processes have been upgraded in the SAS Data Integration Studio ETL process to improve load performance by passing SQL to the database for processing.

PROCESS - EXAMINE, MEASURE, AND MODIFY

One of the strategies to consider when interacting between SAS and your DBMS is the examine, measure, and modify strategy. There are options and information that you can use and review to enable better performance. The code examples below show how to collect information that can help identify DBMS performance issues. This information can also be used to validate the strategy you are using to improve the interaction between SAS and your DBMS.

Collecting timings – there are options that you can set to collect timing information, noting the sastrace statement below. This would give you processing times inside the DBMS. Using these measurements will help you identify SQL that needs further refinement. The option below turns on the timing feature:

```
options sastrace=',,,ts' nostsuffix;
```

Timing information along with assurances that SQL intended to be processed in the database is making it to the database are a good foundation for process improvement. The SAS/ACCESS documentation identifies and explains SASTRACE options and functionality.

Collecting pushdown information – there are SAS options that you can set to collect pushdown information. Note the sastrace statement below. This would show the SQL that is being passed to the DBMS. Using this measure will help you identify SQL that needs restructure so it will pass down to the DBMS. The option below turns on the SQL tracing feature:

```
options sastrace=',,,d' nostsuffix;
```

Identifying SQL flowing through the ACCESS engine can help identify SQL issues, especially if the SQL were part of a code generation process or PROC SQL Implicit Passthru. You can use the SQL captured in the log as input into DBMS query explain tools to help you gather more information as part of your analysis.

CONCLUSION

The "Top Ten" list presented here is a start in your quest for better performance between SAS and your DBMS. Many of these performance boosters efficiently use SQL or DBMS components to improve interaction with your database. Understanding how to reduce data pulls or effectively use DBMS tools can provide significant performance gains. Your process of examine, measure, and modify can be useful tools in increasing performance or providing a communication means to SAS in reporting problems. We are interested in feedback on your interesting and novel performance initiatives or solutions between SAS and your database.

RECOMMENDED READING

SAS Institute Inc. 2009. *Base SAS® 9.2 Procedures Guide*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/cdl/en/proc/59565/PDF/default/proc.pdf.

SAS Institute Inc. 2009. *SAS® 9.2 Companion for UNIX Environments*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/cdl/en/hostunx/59542/PDF/default/hostunx.pdf.

SAS Institute Inc. 2009. *SAS® 9.2 Companion for Windows*. Cary, NC: SAS Institute Inc. Available at support support.sas.com/documentation/cdl/en/hostwin/59544/PDF/default/hostwin.pdf.

SAS Institute Inc. 2009. *SAS® 9.2 Language Reference: Concepts*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/cdl/en/lrcon/59522/PDF/default/lrcon.pdf.

SAS Institute Inc. 2009. *SAS® 9.2 Language Reference: Dictionary*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/cdl/en/lrdict/59540/PDF/default/lrdict.pdf.

SAS Institute Inc. 2009. *SAS/ACCESS® 9.2 for Relational Databases: Reference*. Cary, NC: SAS Institute Inc. Available at support.sas.com/documentation/cdl/en/acreldb/59618/PDF/default/acreldb.pdf.

CONTACT INFORMATION

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

Howard Plemmons
SAS Institute, Inc.
Cary, NC 27523
Work Phone: 919-531-7779
Fax: 919-677-4444
E-mail: Howard.Plemmons@sas.com