Paper 306-2011

Working with a DBMS using SAS® Enterprise Guide®

Howard Plemmons, SAS Institute Inc., Cary, NC

ABSTRACT

The ability to identify process issues when using a database management system (DBMS) can be difficult as you move farther away from direct SQL to the DBMS. As a SAS Enterprise Guide user, how do you ensure that you are executing optimal code for your DBMS? With many new features including in-database procedures, SQL pushdown techniques, and optimizations in SAS/ACCESS®, that task can be overwhelming. This paper provides a step-by-step approach to identification and use of DBMS interaction techniques designed to help you improve your SAS Enterprise Guide experience.

INTRODUCTION

This paper is divided into two sections:

- An overview of SAS Enterprise Guide and components in the current SAS Enterprise Guide 4.3 release along with a peek at some new capabilities in the upcoming SAS Enterprise Guide 5.1 release. This overview helps focus SAS Enterprise Guide components that are used to interact with the DBMS.
- An overview of SAS usage scenarios, DBMS interaction, validation/debugging techniques, and best practices that can be applied when you interact with a DBMS. Rather than trying to document all the DBMSs supported by SAS we will focus on Oracle as a base. Many of the concepts that apply to accessing Oracle will apply to other DBMSs as well.

SAS ENTERPRISE GUIDE

What Is SAS Enterprise Guide?

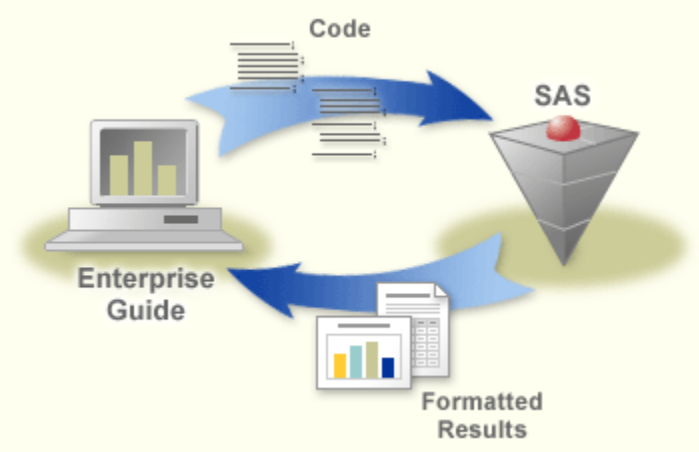

Figure 1: The Big Picture

SAS Enterprise Guide is an easy-to-use Windows client application that provides these features:

- access to much of the functionality of SAS
- an intuitive, visual, customizable interface

- transparent access to data stored in SAS or other 3rd party data stores
- ready-to-use tasks for analysis and reporting
- easy ways to export data and results to other applications
- scripting and automation
- a program editor with syntax completion and built-in function help

The SAS Enterprise Guide 4.3 release has been called the “enhanced enhanced editor”. A description of these enhanced editor functions can best be described by screenshots of functionality in action. Note that this information is important as we move to performance analysis in the second half of the paper.

From these bullet points and diagram you can derive the overall concepts of SAS Enterprise Guide. To understand program/database interaction we will look into components of SAS Enterprise Guide that can be used to facilitate interaction with a DBMS.

SYNTAX SUGGESTION

SAS provides a very robust language that has been tuned with DATA step components, procedures, parameters, and options over the years. Knowledge of these components can be critical to success both in process and performance. You will first view the functionality in the diagrams below which are followed by descriptions of the functionality:

SAS SUGGESTION AND AUTOCOMPLETE

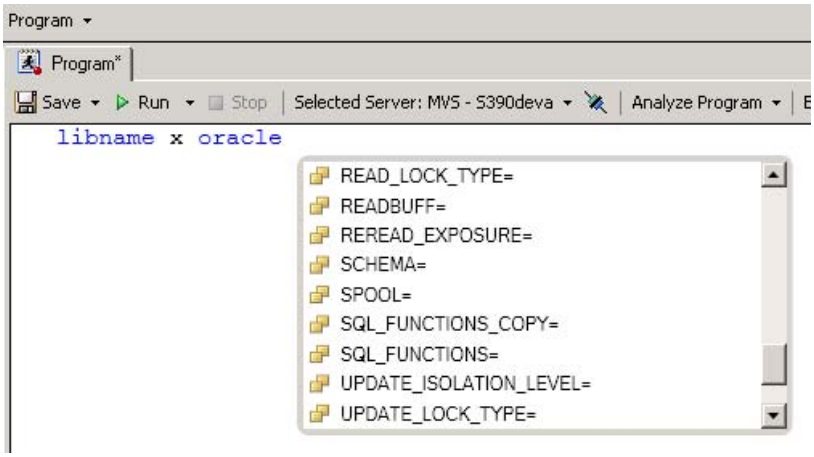

Figure 2: Oracle LIBNAME Options – Suggestion and Autocomplete

The ability to see suggestions for completing statements and option names during the creation process can be very valuable when you create your programs. There might be some options that do not appear in the suggestion boxes when you deal with databases. These would include Bulk Loading options that are listed in the SAS/ACCESS DBMS documentation.

LIBRARIES AND DATA MEMBERS

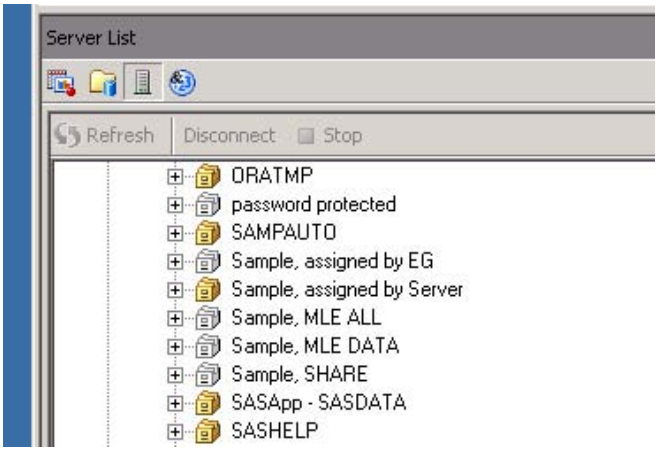

Figure 3: Libraries Available for Processing in Your SAS Job

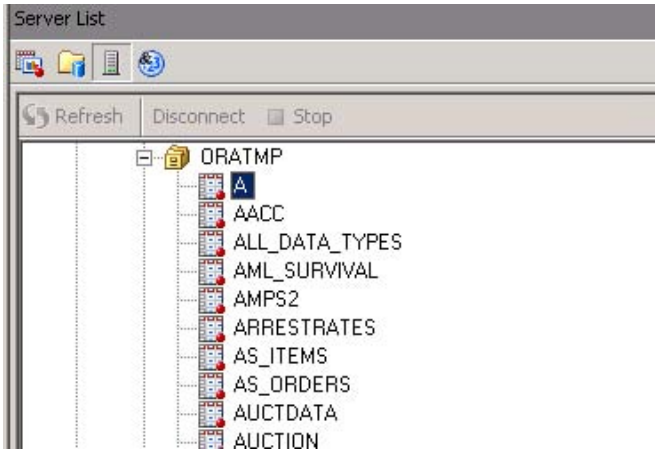

Figure 4: Members (DBMS tables) Available for Processing in Your SAS Job

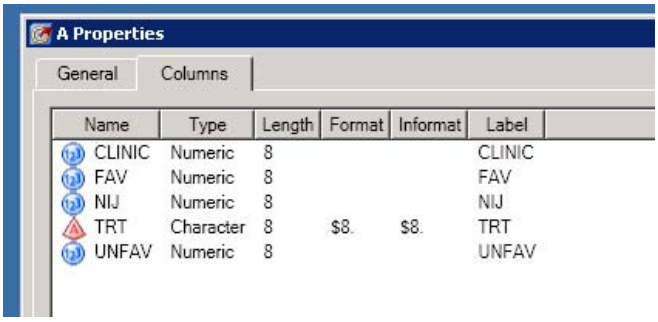

Figure 5: Contents of the DBMS Table A from the Library ORATMP

These figures depict a sequence of events as you drill down through the resources, for example, libraries, available to your SAS Enterprise Guide session. Note that these requests for metadata against the DBMS do not extract table data. They are acquired through an SQL prepare/describe process or by accessing DBMS system tables.

SAS ENTERPRISE GUIDE OPTIONS

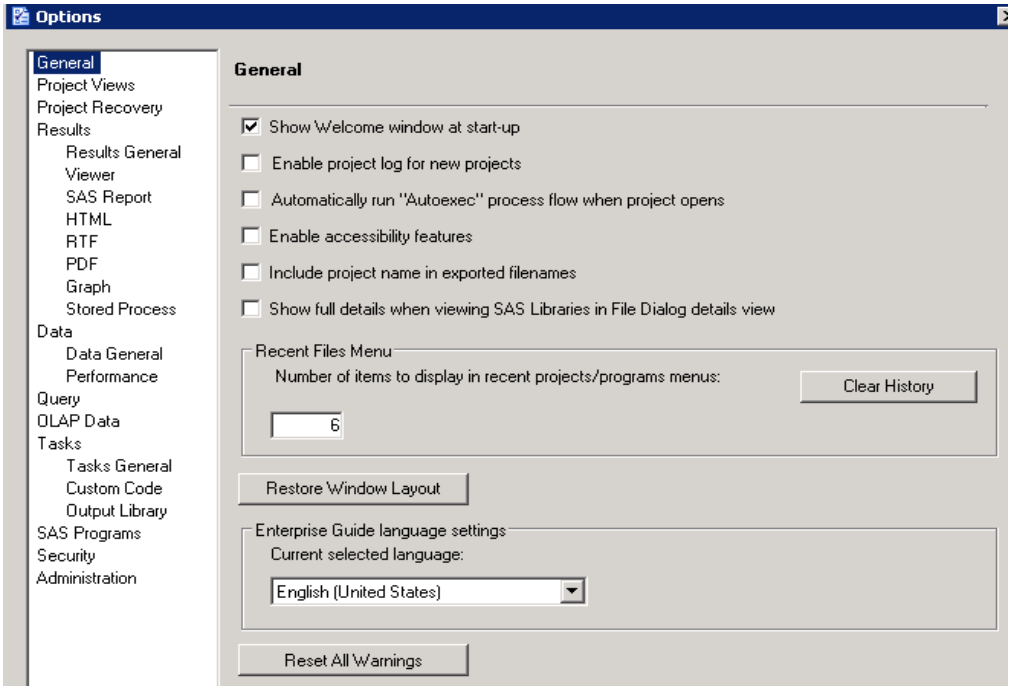

Figure 6: SAS Enterprise Guide Editing and Formatting Options

Controlling and customizing the editing options is a few clicks away (Tools -> Options). The Editor options provide both control and flexibility over editor behavior; however, you probably should try the options settings in default mode first before modifying them.

ENTERPRISE GUIDE 5.1—SNEAK PREVIEW

As the development of SAS Enterprise Guide moves forward, functionality will continue to evolve in order to simplify interactions with SAS. To that end, a couple of new features being worked on for SAS Enterprise Guide 5.1 include:

- Integrated syntax help – introduced with SAS Enterprise Guide 4.3 and enhanced in SAS Enterprise Guide 5.1

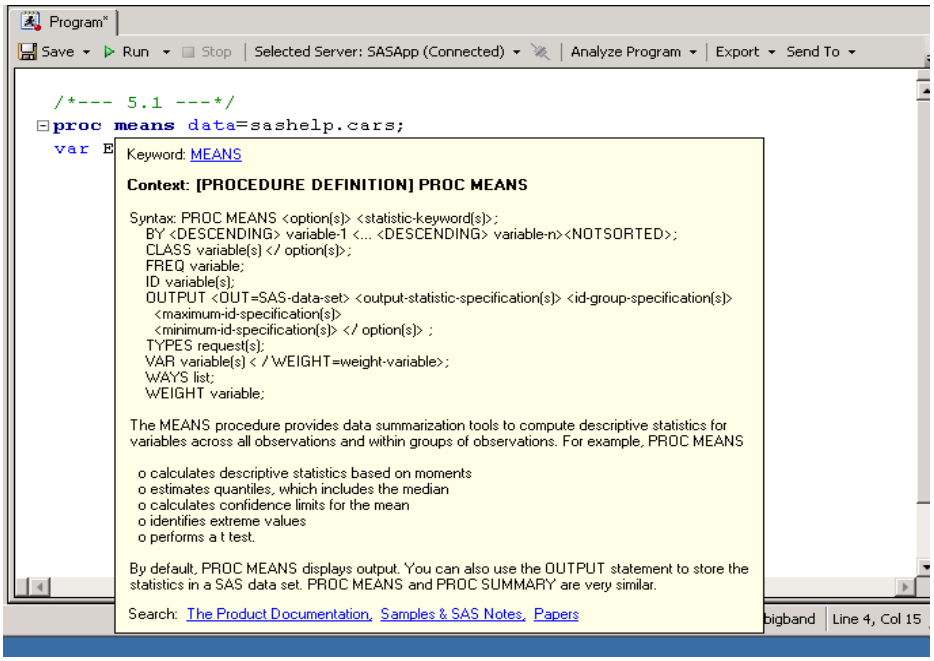

Figure 7: Getting Integrated Syntax Help

This provides you with additional syntax help with the SAS language and SAS procedure options as you craft your SAS code. By holding the mouse on a keyword, in this case, MEANS, you can see additional details. If you need to consult documentation, another mouse click will take you to additional reference materials.

- Variable metadata resolution

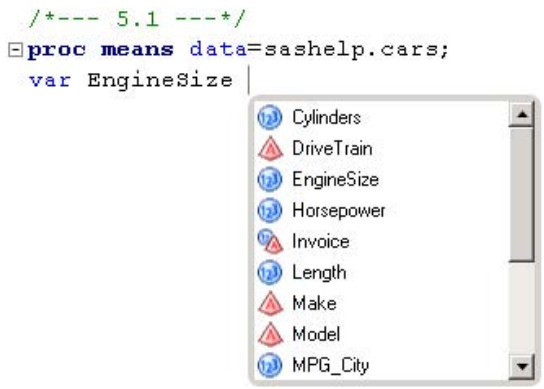

Figure 8: Variable Metadata Resolution

This provides you with additional information about metadata components as you enter your program. In this case you can see all the columns in the table Cars as you type or make you selection from the pull-down list to complete your VAR statement.

SAS ENTERPRISE GUIDE AND DATABASES

SAS Enterprise Guide facilitates communication with databases via the SAS DATA step, SAS Procedures, and SQL entered using PROC SQL. A SAS Enterprise Guide user might have a project that used to run in a matter of seconds that has ballooned into minutes and/or they are creating a project for use by others and need to "tune it up". The diagrams and process hints below will introduce you to concepts used to identify issues when your data comes from a database. Also, a review of some of the in-database procedures will be provided as well.

SAS/ACCESS TO DATA

A LIBNAME statement is the method you would use to hook your DBMS to SAS. The library picture, Figure 3, shows in picture form the libraries that are available to SAS Enterprise Guide. The libraries that point to DBMS data, ORATMP, for example, provide a starting point for SAS code development. Specifically, the library and its contents are your connection to the DBMS.

Note that viewing metadata about a database table does not extract the data. If you are creating libraries in SAS metadata or have them pre-assigned, there are LIBNAME options that need to be considered:

- Connection options – basic connection options might get you to the database; however, SAS Enterprise Guide operations on databases with thousands of tables with many columns should be considered. Some ideas here would be restrictions by DBMS role or schema to reduce quantities of tables and/or columns that need to be processed. This would be a discussion that you need to have with your DBA.
- Specific LIBNAME Options – there are performance-related options that should be considered required on the LIBNAME statement. Consider the table below when setting up your LIBNAME statement. Note some of the defaults in the table below might be specific for a DBMS. You can view the SAS/ACCESS documentation for exact values.

Parameter	Default	Syntax	Description
READBUFF	250	*READBUFF=1024;*	Specifies the number of rows of DBMS data to read into the buffer. SAS allows the maximum number that is allowed by the DBMS.
INSERTBUFF	10	*INSERTBUFF=1024;*	INSERTBUFF, specifies the number of rows to insert. SAS allows the maximum that is allowed by the DBMS. Note if you specify a db_commit value less than the insert buff the db_commit value will override the insertbuff value.
UPDATEBUFF	1	*UPDATEBUFF=0;*	Specifies the number of rows that are processed in a single DBMS update or delete operation
SCHEMA	Null	*SCHEMA=SH;*	The SCHEMA option allows you to read and write objects in a specified RDBMS schema. Note by default this is the same as the connection user. Also note that many RDBMS schemas are case sensitive.
DB_COMMIT	1000 – inserts 0 – for updates	*DB_COMMIT = 0;*	DBCOMMIT= affects update, delete, and insert processing. The number of rows that are processed includes rows that are not processed successfully. If you set DBCOMMIT=0, a commit is issued only once (after the procedure or DATA step completes). Note the 0 for updates means that the commit is issued only after all the update statements have been executed.
DBCREATE_TABLE_OPTS	Null	**DBCREATE_TABLE_OPTS=** *'DBMS-SQL-CLAUSES'*	You can use DBCREATE_TABLE_OPTS= to add DBMS-specific clauses to the end of the SQL CREATE TABLE statement. For example if you wanted to ensure that tables were created in a specific table space.

As you drill though the data you see tables available to you for processing. Looking at metadata from databases is usually not very expensive compared to data reading or extraction. See Figure 4 for a list of tables available for use in SAS Enterprise Guide.

Note the contents of the data. If you are unsure where the data is coming from the number of observations gives you a hint. If it is a missing value you are processing against data whose observation count is not known as it is with SAS data sets.

The CONTENTS Procedure

Data Set Name	ORATMP.A	**Observations**	.
Member Type	DATA	**Variables**	5
Engine	ORACLE	**Indexes**	0
Created	.	**Observation Length**	0
Last Modified	.	**Deleted Observations**	0
Protection		**Compressed**	NO
Data Set Type		**Sorted**	NO
Label			
Data Representation	Default		
Encoding	Default		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
1	CLINIC	Num	8			CLINIC
3	FAV	Num	8			FAV
5	NIJ	Num	8			NIJ
2	TRT	Char	8	$8.	$8.	TRT
4	UNFAV	Num	8			UNFAV

Figure 9: PROC CONTENTS output

All of the interactions described above are the results of executing SQL against the database. These statements are prepared and executed by SAS as the communication mechanism to the DBMS. Now we move forward to reading the data as shown in the picture below:

A ▾

Filter and Sort | Export ▾ Send To ▾

	CLINIC	TRT	FAV	UNFAV	NIJ
1	1	vvv	99	99	99
2	12	zzz	90	12	1
3	2	drug	11	25	36
4	1	cntl	10	27	37
5	.	drug	17	4	20
6	2	cntl	22	10	32
7	3	drug	14	5	19
8	3	cntl	7	12	19
9	4	drug	2	14	16

Figure 10: View Data Rows in the DBMS Table

The data shown above is the results of executing a select against the database. What you need to remember is that you might be selecting against a very large table and some databases do not display rows until the result set is complete. You will find debugging information below to help find issues when processing DBMS data.

LOOKING AT SAS CODE

The picture below shows the sas jobs in a process flow that you can create and execute using SAS Enterprise Guide. SAS Enterprise Guide also provides a SAS log where you can view the results of the execution of each step.

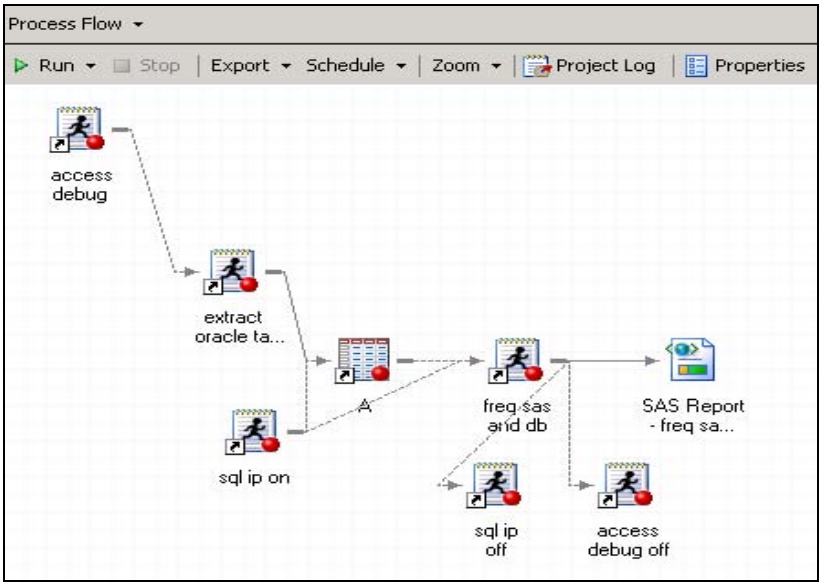

Figure 11: SAS Code in a Process Flow

This SAS Enterprise Guide process consists of several SAS programs. The ones labeled “access debug” and “sql ip” are used to interject debug statements into the job flow. In this process flow you can see how the debug code can be hooked in to provide information about a specific SAS job component. In this case we want to see SQL debug statements and PROC SQL execution statements. The log below shows the debug content.

```
15          /*--- SQL IP going to the DBMS ---*/
16          proc sql;
17           select distinct(clinic) from oratmp.a;

ORACLE_38: Prepared: on connection 0
SELECT * FROM egtest.A


ORACLE_39: Prepared: on connection 0
 select distinct TXT_1."CLINIC" from egtest.A TXT_1

SQL_IP_TRACE: The SELECT statement was passed to the DBMS.

ORACLE_40: Executed: on connection 0
SELECT statement  ORACLE_39

ACCESS ENGINE:  SQL statement was passed to the DBMS for fet
18          quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

Figure 12: SAS Log Showing SQL Tracing Information

This log shows two levels of debugging information that you can use in your process flow. The first set, everything between line 17 and 18 in the log except for SQL_IP_TRACE, comes from the SAS/ACCESS engine. The second set, SQL_IP_TRACE, comes from PROC SQL. With the options set in the SAS flow we are asking for two pieces of information:

1) What is the SQL that is going to be processed by the DBMS
2) Did PROC SQL pass down and execute the SQL on the DBMS

As you can see from the log, an Implicit Pass-Through (IP) statement was prepared and executed in Oracle. Note that the SQL statement executed and the SAS statement are different. This difference is the result of the parse and textualization step that SQL IP must go through to become Oracle SQL. The debug messages indicate success in that the SQL statement was generated by SAS and executed in the DBMS on the first try. If SAS had to remove components that could not be passed down then the SQL_IP_TRACE message would be repeated for each attempt.

Here is your debug reference card. The contents include debug and use options that will help you debug your process flow.

SAS Option	Description and Example
SASTRACE = ',,,d'	Detailed tracing outputs all of the SQL passed to the database including CREATE, DROP, SELECT, INSERT, UPDATE, DELETE, and COMMIT.
	option sastrace=',,,d' sastraceloc=saslog nostsuffix; **data** sales_1; set o.sales; **run**; **produces in the log:** ORACLE_1: Prepared: on connection 0 SELECT * FROM SALES
SASTRACE = ',,d,'	Routine tracing, specifies that all "routine calls" used by tech support for debugging purposes are sent to the log. This option is not particularly useful for the SAS programmer as you can see.
	option sastrace=',,d,' sastraceloc=saslog nostsuffix; **data** sales_1; set o.sales; **run**; **produces in the log:** option sastrace=',,d,' sastraceloc=saslog nostsuffix; ACCESS ENGINE: Entering yoeopen ACCESS ENGINE: Open Mode is XO_INPUT ACCESS ENGINE: Access Mode is XO_SEQ ACCESS ENGINE: Shr flag is XHSHRREC ACCESS ENGINE: Successful SHARING existing connection id 0 ACCESS ENGINE: Entering dbiopen ORACLE: oropen() ACCESS ENGINE: Successful dbiopen, open id 0, connect id 0 ACCESS ENGINE: Exit dbiopen with rc=0X00000000 And a whole lot more….
SASTRACE = ',d,,'	Used for OLE/DB calls, it specifies that all OLE DB API calls for connection information, column bindings, column error information, and row processing are sent to the log. This is useful only for OLE/DB calls.
SASTRACE = ',,,s'	Specifies that a summary of timing information for calls made to the DBMS is sent to the SAS log. This basically tells you where the time is spent during the SQL statement execution. This is not too useful either since we expect that 99.99% of the time is spent fetching rather than preparing or executing SQL statements.
	option sastrace=',,,s' sastraceloc=saslog nostsuffix; **data** sales_1; set o.sales; **run**; **produces in the log:** Summary Statistics for ORACLE are: Total row fetch seconds were: 47.968742 Total SQL execution seconds were: 0.002208 Total SQL prepare seconds were: 0.002550 Total seconds used by the ORACLE ACCESS engine were 52.948376

SAS Option	Description and Example
SQL_IP_TRACE=NOTE	Specifies that a note will be written to the log each time the SQL is modified by SAS and submitted to the DBMS. Each time you see the message in the log you will see additional SQL that is processed through the SAS/ACCESS engine. Each time you receive the message indicates that the SAS Implicit Pass-Through (PROC SQL executing SAS SQL) is changing the SQL so it can be processed by the DBMS. If your SAS SQL contains SAS components that cannot be passed to the DBMS then it will iterate through the Parse/Textualize/Execute process until a successful SQL statement executes on the DBMS. As a rule of thumb, you only want one message from the SQL_IP_TRACE option in the log, which indicates success on the first try to Parse/Textualize and Execute SAS SQL on the DBMS. If you see more than one then you should investigate both the SAS SQL and the DBMS SQL displayed when using SASTRACE. option sql_ip_trace=note;

Finally, to ensure that your SAS log is readable, use the "**nostsuffix**" option, for example:
option sastrace=',,,d' sastraceloc=saslog **nostsuffix**;

SAS Option	Description and Example
Normal	option sastrace=',,,d' sastraceloc=saslog 8 1562694543 no_name 0 DATASTEP ORACLE_3: Executed: 9 1562694543 no_name 0 DATASTEP SELECT "ID", "NAMEID" FROM SALES_TEST WHERE mod(id,2)=0 10 1562694543 no_name 0 DATASTEP 11 1562694543 no_name 0 DATASTEP 12 1562694543 no_name 0 DATASTEP ORACLE_4: Executed: 13 1562694543 no_name 0 DATASTEP SELECT "ID", "NAMEID" FROM SALES_TEST WHERE mod(id,2)=1 14 1562694543 no_name 0 DATASTEP 15 1562694543 no_name 0 DATASTEP
With. NOSTSUFFIX	option sastrace=',,,d' sastraceloc=saslog **nostsuffix**; ORACLE_2: Executed: SELECT "ID", "NAMEID" FROM SALES_TEST WHERE mod(id,2)=0 ORACLE_3: Executed: SELECT "ID", "NAMEID" FROM SALES_TEST WHERE mod(id,2)=1

BEST PRACTICES

Here are some tips that might help in both the performance discovery process and interaction between generated SQL and the DBMS:

1. Avoid and or control heterogeneous joins – joining data from two different sources. This would include:
   - Joining SAS data with DBMS data where the DBMS table size is large. In this case the data from the DBMS will be extracted into SAS and joined with the SAS data.
   - Joining DBMS tables identified by two different LIBNAME statements. Unless the elements of the two LIBNAME are equivalent then both DBMS tables will be extracted into SAS where the join will take place.
2. Always use buffering options. This would include:
   - READBUFF – one of the biggest performance influencers that you can use. This identifies the number of rows that a DBMS will pass to SAS in one fetch operation.
   - INSERTBUFF – identifies the number of records to be buffered on output.

3. Always use bulk loading (BULKLOAD=YES data set option) and associated BL_ options when loading data of size into the DBMS. This would include creating and loading new tables as well as adding rows using PROC APPEND.

4. Compress your SAS data sets.

   Here's how:

   Data Set Option – add **compress=yes** as a data set option (for example, data a(compress=yes))

   System Option – submit **option compress=yes** .

   Here's why: more efficient data transfer, smaller footprint on disk, more blocks into memory will give you performance gains

5. Compress your database tables.

   Some databases offer table level compression. If they do you can pass this request to the DBMS using the DBCREATE_TABLE_OPTS SAS/ACCESS engine option. For Oracle you could specify this option as a data set option when creating new tables in the database from SAS:

   data lib.tab(DBCREATE_TABLE_OPTS=COMPRESS);

   Compressing data at the DBMS level enables many of the features as compressing a SAS data set.

6. Refresh table statistics.

   The DBMS relies on table statistic information to optimize requests for data. If the table statistics are not current then the DBMS SQL optimizer could make execution choices that directly impact performance. Although table statistics are on the DBMS side of the equation you can influence them from SAS. Consider the following code snip:

```
proc sql;
connect to Oracle( USER=XX PASSWORD=XX path=ORA10g);
execute ( execute
            DBMS_STATS.GATHER_TABLE_STATS (
              ownname => 'SH',
              tabname => 'SALES',
              estimate_percent => 40 ) ) by oracle;
              disconnect from oracle;
quit;
```

   In this example we have updated DBMS statistics from SAS. Your DBA might not permit this type of operation; however, you should consider invoking the process of updating statistics if you perform these operations on the DBMS:

   - After loading a significant number of rows into a table.
   - After updating data in a table you might need new statistics on the average row length.
   - After bulk load operations
   - After table structure has changed

7. Consider CONNECTION= as you drive multiple users against the DBMS. For example do you want unique connections or shared connections? This would save database connection by allowing you to share read connections across multiple LIBNAME statements.

IN-DATABASE PROCEDURES

Some of the procedures in your SAS jobs might now run inside the DBMS. Simply, this is a method of data access that pushes aggregation and processing down to the DBMS level. This is accomplished by adding SQL components to the SAS procedure. The performance lift can be significant given that less data is flowing out of the DBMS. Some of the procedures that now run with augmented data access include:

- PROC CORR
- PROC FREQ
- PROC RANK

- PROC REPORT
- PROC SORT
- PROC SUMMARY/MEANS
- PROC TABULATE

Details on the in-database concepts can be found in SAS 9.2/9.3 documentation. The in-database capability of these procedures works with many different DBMS. The execution times between PROC FREQ with and without in-database capability can be significant.

CONCLUSION

SAS Enterprise Guide provides an excellent development framework for entering, organizing, sharing, and executing SAS code. When processing against databases you should be aware of potential issues that might occur. The ability gained through practice and established best practices will improve your interaction with a DBMS from SAS Enterprise Guide.

REFERENCES

Bangi, Audimar, Chris Hemedinger, and Stephen Slocum. 2010. “SAS Programmer’s Paradise: New Goodies in SAS Enterprise Guide 4.3.” *Proceedings of the SAS Global Forum 2010 Conference*. Cary, NC. SAS Institute Inc.

SAS/ACCESS documentation for SAS 9.3

SAS Enterprise Guide 4.3 documentation

RECOMMENDED READING

SAS Enterprise Guide documentation available on the SAS Technical Support Web site:

http://support.sas.com/eguide

SAS/ACCESS documentation available on the SAS Technical Support Web site:

http://support.sas.com/documentation/onlinedoc/access/index.html#access92

CONTACT INFORMATION

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

Howard Plemmons
SAS Campus Drive
SAS Institute Inc.
Cary, NC 27513
E-mail: howard.plemmons@sas.com