Paper 151-28

# Scaling SAS® Data Access to Oracle® RDBMS

Howard Plemmons, SAS Institute Inc., Cary, NC
Andrew Holdsworth, Oracle Corp., Redwood Shores, CA

## ABSTRACT

How much data will you be required to store and process, now, six months from now? How can you effectively manage and balance user requirements for data analytics vs. transaction requirements from your OLTP system? These questions as well as many others are becoming more demands than requests. SAS and Oracle have solutions that will help you provide some answers. Loading, extracting and feeding high end SAS analytics with data stored in Oracle provides answers to more than data volume concerns.

## INTRODUCTION

According to the dictionary Scalable was coined in 1580 AD and is defined as, “capable of being scaled”. It could be speculated that the people of the time would define scaled as the ability to climb a wall. Scalable in their eyes would be defined as their ability to do so. Several hundred years later we are still looking at the wall; however, now instead of bricks and mortar it is made of data. Our ability to scale the wall is directly proportional to how quickly and efficiently we can process data. To aid in that process the SAS/ACCESS® family of products attacks this wall by building interfaces that understand the “ladders” provided by the DBMS vendors. This paper will focus on scalable features delivered with SAS System 9 and Oracle 9i software releases.

## SCALEABLE SAS VERSION 9

With SAS System 9 the SAS/ACCESS to Oracle product was enhanced so that parallel reads could be executed against the Oracle server. This parallel read concept coupled with read thread enabled statistical applications provides the bases for SAS scalable data access.

## SAS SYSTEM OVERVIEW

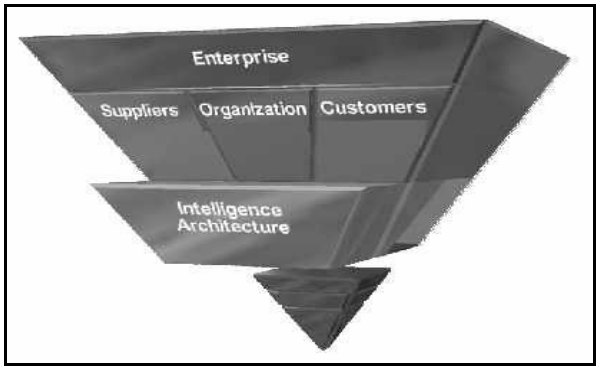


Figure 1

At the core of SAS System 9 is the Intelligence Architecture, the foundation of the SAS Intelligence model as shown above in Figure 1. The Intelligence Architecture contains the technology to perform ETL, business intelligence and analytical intelligence, leveraging the manageability, interoperability, usability, and scalability of the SAS system. Over 60 products are included in the Intelligence Architecture and are fully integrated to offer robust and portable functions of the SAS system.

Below the Intelligence Architecture is the infrastructure layer, denoted by the small pyramid in Figure 1. This pyramid contains ERP, DBMS, Operating System and data sources. SAS can complement this existing infrastructure and leverage the IT investment to optimize total cost of ownership. For example, if the infrastructure consists of Oracle 9i running on HPUX 11i 64 bit, SAS provides the technology to access the data using SAS/ACCESS Interface to Oracle product. In addition, SAS fully integrates and cleanses the Oracle data, preparing the data to be analyzed with SAS software.

As described above the SAS/ACCESS Interface to Oracle product, it is one of many components of the Intelligence Architecture. In order to apply and gain any intelligence, accessing the data is the first critical step. Once the SAS system is plugged into your infrastructure, the power of SAS can be optimized. Leveraging 26 years of SAS knowledge and experience, the Intelligence Architecture also delivers a comprehensive suite of analytical and business intelligence software. Bundled into the SAS solutions, the technology provides you with insight and the power to know your customers, organization and suppliers. SAS gives you the intelligence and “The Power to Know.”

### SOME HISTORY OF THE SAS/ACCESS PRODUCT:

SAS has had an interface to Oracle since early SAS V6, which would have supported Oracle V6. Early SAS releases were made up of PROC ACCESS, PROC DBLOAD and PROC SQL. These early releases of SAS provided customers with the tools to access Oracle; however, they did not contain as tight of integration to Oracle as the current releases of SAS/ACCESS.

*SAS V8* saw performance enhancements to the procedure interfaces driven to a great extent by the debut of the SAS/ACCESS to Oracle libname engine. Some of the features and controls make available to the user in SAS V8 are:

A fully functional libname engine in this release of the SAS/ACCESS product provides the user with extract, insert, update, and delete and DDL functionality when interacting with Oracle.

Direct support of DBMS functions. This involves mapping SAS functions to equivalent Oracle DBMS functions and passing the mapped function to the DBMS server for processing. This has a proven positive performance measurement, as seen with where clause processing.

Introduction of a direct interface to Oracle’s SQL Loader. This process was imbedded in the engine so SAS applications that set the appropriate options could take advantage of the performance offered by Oracle’s SQL Loader.

A set of options that allow the SAS/Oracle customer more control over their environment. For example the SAS options INSERTBUFF, UPDATEBUFF and READBUFF allow you to control Oracle buffer sizes from SAS.

Enhancements to our SAS SQL logic to pass additional join components to the DBMS. This performance enhancement helped reduce I/O and disk usage by pushing join components when generating the Oracle SQL where clause.

*SAS SYSTEM 9* saw performance enhancements to the Libname engine with the support of threaded I/O in both the SAS/ACCESS to Oracle engine and SAS applications. Using this concept the SAS product accesses the data in parallel and returns blocks of the data to multi-threaded SAS applications. Diagram 1 below shows this process:

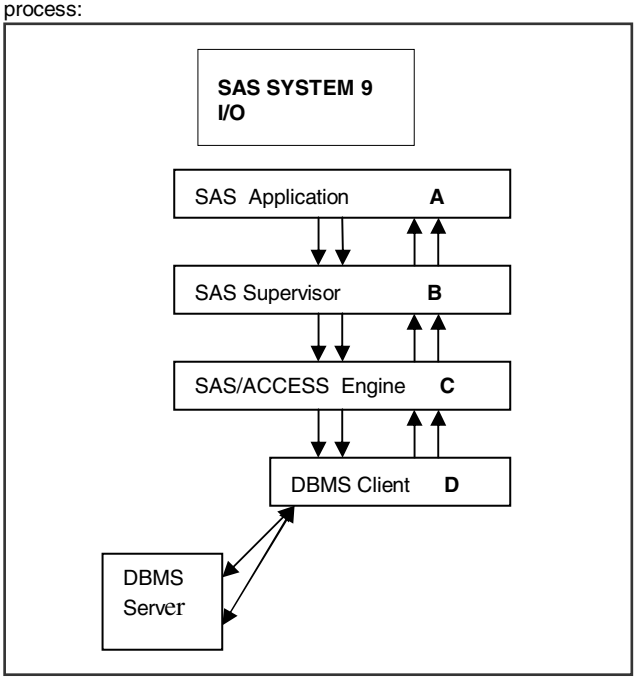


Diagram 1 – SAS SYSTEM 9 Threaded Read Model

With the data streaming back and processed on multiple threads within the SAS application has netted significant SAS performance boosts as compared with the SAS V8 I/O model. This technology will be generally available with the SAS System 9 due to be released in 2003.

The significance of Diagram 1 can be explained in terms of how SAS processes information in each step. Details on this process are:

**A** – A SAS application that has been thread enabled triggers the threaded I/O model available with SAS System 9. A request is made to the SAS Supervisor for threaded services **B**.

**B** – The SAS Supervisor is the controlling force behind the SAS I/O model in SAS Version 8 and in SAS System 9. The threaded I/O model is an extension of the SAS Version 8 I/O model. Both threaded and the SAS V8 I/O model (single threaded) co-exists in SAS SYSTEM 9. Once the supervisor recognizes the new I/O calls it triggers the SAS/ACCESS engine to service the request **C**.

**C** – The SAS/ACCESS engine processes the threaded request and attaches to the DBMS to resolve the request. In doing so the access engine will retrieve blocks of data in parallel and pass it back to the SAS Application via the engine supervisor as blocks of data. Using this I/O model data is moved more efficiently from the DBMS to SAS for processing.

**D** – The DBMS client is software used to communicate with the DBMS server. In this case it is the Oracle client software. The SAS/ACCESS engine uses its interface to Oracle to open multiple connections to the DBMS. In doing so the engine is able to obtain data from the DBMS on multiple threads.

The efficiencies of this process are discussed in greater detail in subsequent sections.

## USING ORACLE FEATURES WITH SAS

The SAS/ACCESS Interface to Oracle product provides you with different interfaces into Oracle. These range from direct SQL interaction to SAS SQL generation. The interaction with the Oracle server is enhanced with a variety of options that give you control of everything from buffer sizes to formatting. As mentioned above this functionality provides for full read, write and update controls.

The sample code below shows SAS SYSTEM 9 at work and highlights one of the SAS multi-threaded analytic procedures using the SAS/ACCESS Interface to Oracle Libname Engine. The sample code will be used to highlight the following:

- Generating and loading sample data with SAS and Oracle
- Processing the sample data without threading (single CPU, resource bound systems)
- Processing the sample data using multi-threaded SAS application and multi-threaded access to the Oracle server
- Processing the sample data with limited threads on the SAS application side and multi-threaded access to the Oracle server
- Results of the SAS processes

## GENERATING AND LOADING SAMPLE DATA

```
/*--- this is a SAS macro that will be used
later in the process ---*/
%macro makeRegData(nObs,nVars,oraSize);
   %let ORASIZE=&oraSize;

   /*-- create data for the regression test --*/
   data work.xxx;
      array x{&nVars} x1-x&nVars;
      drop i;

      do n=1 to &nObs;
         do i=1 to &nVars;
             x{i}=ranuni(12345);
         end;
         y=x{1}+3*x{&Nvars}+rannor(12345);
         output;
      end;
  run;
%mend;

/*--- create the data to load into Oracle -
execute the macro above ---*/

%makeRegData(500000,49,20);


/*--- issue the SAS libname statement     ---*/
/*--- bulkcopy=yes will invoke Oracle's   ---*/
/*--- SQL Loader to load data into Oracle ---*/
libname x oracle user=scott pass=tiger
bulkcopy=yes;

/*--- using SAS bulk load the SAS data into
Oracle        ---*/
/*--- this invokes the Oracle SQL Loader under
the covers ---*/
data x.tdata_init;
       set work.xxx;
run;
```

```
/*- with SAS create the Oracle partition table*/
/*- direct access to Oracle using Oracle SQL -*/
proc sql;
       connect to oracle( user=scott
pass=tiger);
exec( create table tdata
       partition by range(n) (partition obs1
values less than (126000) tablespace tabspace1,
        partition obs2 values less than (251000)
tablespace tabspace2,
        partition obs3 values less than (376000)
tablespace tabspace3,
        partition obs4 values less than (500001)
tablespace tabspace4)
         nologging as select * from tdata_init)
by oracle;
quit;

/*- regression procedure execution macro note */
/*- the data= statement is using the libname  */
/*- definition above to access the Oracle     */
/*- table tdata.                              */
%macro runReg(nVars);
  proc reg data=x.tdata;
      model y=x1-x&nVars;
      performance details;
  quit;
%mend;
```

**THREADED PROCESS**

The ability of the application to thread both data acquisition and data processing provides the greatest potential for performance gains. The following SAS procedures have been thread enabled for SAS System 9:

proc sort
proc summary
proc dmine
proc means
proc reg
proc glm
proc dmreg
proc loess
proc robustreg
proc dmdb

Proc reg used in the sample code has implemented the threading process for both input and execution. Refer to procedure specific SAS documentation for additional information on threading and thread control methods.

**PROCESSING THE SAMPLE DEATA – NO THREADS**

```
/*- turn off threading at the application and
SAS/ACCESS Interface to Oracle layer to simulate
the process of previous SAS versions ---*/
options NOTHREADS;
options DBSLICEPARM=NONE;


/*--- execute the regression macro ---*/
%runReg(49);
```

**PROCESSING THE SAMPLE DATA – FULL THREADS**

```
/*--- Using SAS options control the CPU
resources and Oracle connections for this
process ---*/
options CPUCOUNT=4;
options DBSLICEPARM=(ALL,4);
/*--- execute the regression macro ---*/
%runReg(49);
```

**PROCESSING THE SAMPLE DATA – LIMITED SAS THREADS WITH MULTIPLE ENGINE THREADS**

```
options DBSLICEPARM=(ALL,4);
data _NULL_;
set x.tdata;
run;
```

**RESULTS OF THE SAS PROCESS**

Running on a 4 way configuration on a SUN E10000 with 4G RAM and Solaris 2.8 using the sample code above netted the following performance metrics. As performance metrics go, your results may differ. These are offered as an illustration of the value adds in parallel processing from both the SAS data acquisition and internal processing:

Given the non threaded execution as the control group the performance percentages were:
Control group (no threads) data access and analytics – 1
Unrestricted threading (full threading) data access and analytics – delivered a result **34 percent faster** than the control group
Control group (no threads) data access only - 1
Unrestricted threading (full threading) data access - using the libname engine delivered a result set, in this case null set **27 percent faster** than the non-threaded execution.

SAS threaded applications can be parameter controlled as shown in the sample code. Optimization concerns when dealing with threaded implementations:

- System resources and system load – multi-threading or processing in parallel will consume CPU and disk resources; however, the strain on the system should result in significant performance (time to answer) improvements. For example, using SAS options and a threaded SAS procedure with the SAS/ACCESS to Oracle engine will use threads for:

  Connecting to the Oracle server for data extraction in parallel

  Processing the data extracted from the DBMS on parallel threads in the SAS procedure.

  Therefore, care must be taken to monitor and tune the threaded process. Both SAS and system options that controls threads, thread limits and thread usage must be considered.
- Data Organization and optimizations – DBMS tuning, data organization, indexed access and statistics are key components in any data access performance project. We have seen query performance significantly increased when the data and DBMS have been sufficiently tuned for data access (hours to minutes, minutes to seconds).
- Tuning with SAS options – as shown in the sample code the number of threads selected from the SAS/ACCESS options matched the number of Oracle table partitions which will drive one thread/connection to each partition. The SAS analytical procedure will run a thread process on each CPU available which is four. In the unrestricted case I/O and threading match 1-1. SAS options and libname options will allow you to optimally tune this process. As mentioned earlier the number of threads generated per CPU needs to be monitored and tuned.

## DATABASE SERVER CONCEPTS

The ability of the SAS user to effectively use the two products in a desirable fashion at times can be daunting. To help empower these users some key concepts of Oracle 9i are described below. These include:

- Optimization of the Star Schema
- Use of Partitioning
- Use of Parallelism
- Use of Direct Path Options
- Use of Specialist Indexes
- Use of Oracle Analytical Functions
- Use of Materialized Views
- Correct Use of the Optimizer

SAS continues to support new DBMS features and releases and has support for the new optimizations in Oracle 9i. The support for various releases of Oracle can be viewed on the SAS web site:

http://www.sas.com/service/techsup/access/searchPage.hsql

You can view SAS/ACCESS engine performance enhancements and features for SAS System 9 on the following web site:

http://v9doc.sas.com/sasdoc

The documentation on the web site above describes both the threaded support for SAS System 9 and new features and support added to the SAS/ACCESS Interface to Oracle. Note if you are a first time user to this site you will have to provide some information to gain access.

### TUNING STAR QUERIES

The star query and associated star schema is a popular implementation in many data warehouses today. The Oracle9i optimizations for the star query are simple to implement and tune. These notes provide a quick and simple way to optimize your star queries.

A start schema consists of a central fact table that contains a number of dimension keys that reference the primary keys to a number of dimension tables. The fact table usually contains some sort of data value such as sales amount on a transaction in addition to the dimension keys. The dimension tables usually represent basic reporting groups such as time, product, customer, geographical location etc. The dimension tables are usually small in comparison to the fact table yet the fact table data can be described a sparse in that its size is small compared to the Cartesian product of all the dimension tables.

The Oracle9i star query optimization is a three-step process as follows:

Apply query where clause predicates to dimension tables and generate lists of values of keys to query into fact table.
Query Fact table using foreign keys from step 1. This is done by use of single column bitmapped indexes on the low cardinality columns in the fact table.
Having resolved the query within the fact table re-join the result set back out to the dimension tables and applies rollups as required within the Query.

To enable this optimization in the database implementation the DBA should follow the following steps.

- Build single column bitmapped indexes on each dimension key on the fact table. If you wish the query to execute in parallel partition the fact table on one of the dimension keys(usually time) and build local bitmapped indexes with the parallel clause set.
- Build indexes on the dimension tables columns most common for query predicates.
- Build indexes on the dimension tables columns primary keys to assist the re-join to the dimension tables.

Generate statistics on the Schema Set
STAR_TRANSFORMATION_ENABLED=true in the init.ora or session.

To further optimize the query the following tuning steps can be performed.

- Build bitmap join indexes to merge steps 1 and 2 into one step.
- Build better indexes on dimension tables to support step 3 and minimize logical I/O
- Optimize the join back to the dimension tables by setting the init.ora or session parameter STAR_TRANSFROMATION_ENABLED=TEMP_DISABLE. This will prevent the optimizer creating a temp table of the dimension tables rows and allow indexed access back into the dimension tables.

### THE USE OF PARTITIONING

Oracle's partitioning allows a table, index or cluster to be broken down into a smaller number of elements. These are known as partitions. The motivation for partitioning came from system administrators who were facing increasing sized tables. The problems with very large tables were that maintenance operations such as reorganization and index rebuilds became longer than permitted in a downtime window.
By use of partitioning a table the maintenance operations can be done on smaller pieces and the Database administrator can adopt a divide and rule policy to achieve higher availability.

In data warehousing partitioning is commonly adopted for exactly the same reasons in that makes database administration simpler and faster. In many data warehouses portioning is crucial to the success of the project. Examples of partitioning to speed administration of a data warehouse include:

Use of rolling windows for online data. New partitions are exchanged into a table and old partitions exchanged out of table by use of a dictionary operation rather than by massive dml statements (insert and delete). Allowing index rebuilds/ table re-orgs to be performed on a single partition at a time. This frees up system resources and minimizes outage times.
What is more interesting however for a data warehouse user is the use of partitioning to speed up query performance. Because by partitioning a table into partitions data is located according to its data values. All information about the table-partitioning scheme is kept in the data dictionary and for this reason when a SQL statement is optimized it is able to optimize the query to reduce the number of rows accessed and hence reduce the system response time.

Oracle has three core methods of partitioning. These are as Follows:

Range Partitioning. Range partitioning is done by placing data into a partition according to its data value being between a high and low value. This is often used for date fields where a period such as month can be stored in single partition. Queries that contain where clause predicates that reference the partitioning column will only reference the partitions that have valid values. This technique is often called partition pruning.
Hash Partitioning. Hash partition divides the rows of a table between a number of specified hash partitions. The number of hash partitions should be a power of 2 and the column or key to hash on should be either a column that is frequently joined on or key that can define a discreet set of rows. The advantage of hash partitioning on

a join key is that hash join optimizations can be done to reduce the join cost.

List Partitioning. List partitioning allows the portioning scheme to create partitions for specific data values rather than a specified range. This is commonly used in grouping columns and geographical fields where the number of distinct values is small but fairly selective for query processing.

These methods can be combined to a composite method of partitioning. This is called sub partitioning. Examples of this would include an orders table partitioned by range on the order date and sub partitioned by hash on the customer key.

**THE USE OF PARALLEISM**

The use of parallelism to reduce query response times fundamental to most data warehouses. Oracle is able to run queries in parallel on tables. It does not require any partitioning to achieve parallelism. Oracle achieves its parallelism by use of multiple query slaves that work on behalf of a user process often referred to as the query coordinator. The degree of parallelism for the query can be specified in multiple ways to control the amount of query slaves used for a query. Oracle is able to scan tables, join tables, sort row sources in parallel to reduce query execution times.

When using SAS it is possible to retrieve from Oracle by selecting from a partitioned table to retrieve data on multiple SAS threads. This is not part of Oracle's parallel query functionality but can be combined with parallel query is the results set is inserted into an intermediate partitioned table.

**THE USE OF DIRECT PATH**

Data warehousing requires the loading and movement of data in large quantities at high speed. Oracle has the ability to move data rapidly by use of the direct path options. The direct path options allow data to be loaded directly into the database files by bypassing the Oracle buffer cache and transaction layer. SAS allows use of the direct path by allowing the invocation of the direct path option within SQL*Loader. However this is not the only place that the direct path option is available. Other way to invoke the direct path include:

Use of "CREATE TABLE AS SELECT" or "INSERT" with the append hint when moving data from within a SQL statement. This a very common technique for creating intermediate results sets.

SQL*Loader

Oracle Call Interface

It should be noted that these techniques are very powerful when combined with both parallelism and partitioning.

**THE USE OF SPECIALIST INDEXES**

Oracle has numerous indexing options. Each indexing type has a specific use and care should be given to the design and implementation of a data warehouses indexes. The types of indexes used in Oracle data warehouses include the following:

B-Tree Indexes. B-Tree indexes are well documented and under stood. They are particularly good when the indexes are very selective. For this reason they are often used to maintain constraints to primary and unique keys as well provide keyed access to tables. The key disadvantage of B-Trees in the data warehouse space is when queries require index merge from two or more indexes it is very I/O and CPU intensive.

Bitmapped Indexes. Bit mapped indexes negate the issues with B-Tree indexes in that they use a compressed bitmap to store the rowids for each row for each distinct value. For columns with few distinct values a bitmap index is an excellent choice as it is both fast and small in terms of disk space. The full power of a bit mapped index is fully exploited when additional columns in the table have their own bitmap index. The query is then able to use OR and AND operations on the actual bitmaps to reduce the row set to be returned.

Functional Index. A functional index is a special index in that it allows the query to work against a value derived from one or more of the columns in the row. A simple example of the would be the simple function UPPER(col) on a column to enable case insensitive query searches without having to scan and convert a table. More complex functions may include the line item value e.g. ( price*qty*(1-discount)*(1+taxrate)).

Join Indexes. Join indexes often used in optimizations to star queries allow the rowids in a second table to be stored with respect to a value in a second table that it is joined to. A simple implementation would be to index a product category in a product table against the product_id in a star schema fact table.

**THE USE OF ORACLE SQL ANALYTICAL FUNCTIONS**

The Oracle analytical functions allow further segmentation and functions that can be applied sub sets of data returned from queries. The analytical functions represent SQL extensions developed in conjunction with the ANSI SQL committee to allow the definition of queries with in built analytical functions. These functions include ranking, moving averages and other analytical functions.

The performance implications of this functionality is that it allows queries to be written that require only one pass of the data rather than multiple passes that would be required in conventional SQL.

**THE USE OF MATERIALIZED VIEWS**

Oracle has the ability to store the results of predefined queries within the database. These are known as materialized views. The Oracle optimizer is able to recognize when materialized queries exist and if the results set would be useful in speeding up as user query. This feature is part of the query rewrite features of Oracle and is transparent to the end user. The most common uses of Materialized views is to store the data in sum de-normalized form. This includes aggregates, pre-joined data and data formatted for reporting. Because a Materialized view is stored as a table it can take advantage of all other Oracle facilities such as Partitioning, Parallelism, Special Indexes etc. In many cases an experienced warehouse designer will build materialized views upon materialized views to get the required level of performance.

**GETTING THE BEST OUT OF THE OPTIMIZER**

The Oracle Cost Based optimizer is used to determine optimum access to the query data. In order to get good optimization it is important to keep and maintain statistics on the tables in any query. Schema statistic as best maintained by execution of the package dbms_stats(). This package as well as allowing generation of schema statistics also allows statistics backup and restore.

## CONCLUSION

The new SAS I/O model enhancements, extensions to the SAS/ACCESS tools set in SAS SYSTEM 9 and Oracle 9i enhancements are all methods that can be used to help achieve scalable data access. The ability to climb the wall of data confronting your company using scalable tools may be directly proportional to how you can resolve the equation:

```
Scalability success = (data design + data
access) * scalable tools factor
```

Some of the most significant performance gains we have seen from customer reported issues have been achieved by correcting data design and access method (including indexing) issues in the data. Note: armed with scalable tools does not guarantee scalable performance without serious consideration into how you will be processing and organizing your data.

## CONTACT INFORMATION

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

Howard Plemmons
SAS Institute Inc
SAS Circle
Cary, NC 27513
Work Phone: 919-531-7779
Email: Howard.Plemmons@sas.com

Andrew Holdsworth
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
Work Phone: 650-506-2938
Email: Andrew.Holdsworth@oracle.com