Paper 239-2007

Talking to Your RDBMS Using SAS/ACCESS®

Dianne Louise Rhodes, BAE-IT, Washington, DC

ABSTRACT

SAS/Access has grown in complexity as SAS continues to make it easier to use and expand its functionality. There are now many different ways to access your Relational Database Management Systems (RDBMS) data – by using a Libname or SQL Pass-Through facility. What's the difference? What's going on "under the covers?" Understanding the underlying process that SAS uses to talk to an RDBMS will help you program more efficiently. When is it better to use a Libname, and when is it better to use SQL Pass-Through. Did you know that you can talk more directly to your RDBMS (Oracle, specifically) using Oracle hints and the Bulk Loader? Do you have old SAS/Access view and access descriptors lying around from the old days of writing descriptors and using DBLOAD? Don't fret. You can convert them into version 9 SAS views, either one-at-a-time or for an entire library, using CV2VIEW.

INTRODUCTION

This paper covers the basics of using SAS/ACCESS to RDBMS, progressing with more detailed and complex concepts. This should provide the reader with a good starting point in learning how to use SAS/ACCESS.

ACCESS USING LIBNAME

You can access your RDBMS using Access in several different ways. Using a LIBNAME is usually the fastest and most direct method. The exception is when you need to use non-ANSII standard SQL. SAS/ACCESS requires ANSII standard SQL; the pass-through facility, discussed below, allows all the extensions to SQL provided by your RDBMS.

It is common to use macro variables to provide the security sensitive values, such as password and schema. These keywords may be defined behind the scenes in your SAS autoexec. At The Bureau of Labor Statistics (BLS), we use %Windows and allow the user to enter them from the %Window display. The syntax for a libname statement is:

```
Libname Oracle-name Oracle [Engine]
User = &User
Path = &Path
Schema = &Schema
Pwd = &Pwd
Readbuff = 1000
Connection = Unique
;
```

Where User is the Oracle user name, Pwd is the password associated with the user name, Path is the name of the Oracle DB, Schema is the name of the schema that defines the tables and views the user can access. Readbuff is an option which increases efficiency of access, and Connection = Unique solves another performance issue discussed below.

PROS

- Less lines of SAS Code. A single libname statement establishes connection and allows you to view tables.
- You can use SAS datasteps and procedures and don't need to know the PL/SQL of your RDBMS
- Provides more control over RDBMS operations including locking, spooling, data type conversions
- Engine can optimize processing of joins and where clauses by passing them directly to the DB
- Takes advantage of RDBMS indexing
- Engine can pass some functions directly to RDBMS
- Use threaded reads

CONS

- Performance may be degraded with multiple users

Performance problems have been noted; your users may be unable to connect to the Oracle instance. The reason behind this problem is that SAS creates a permanent connection each time a libname is setup. Leconte (2005) found in his shop, over 600 Oracle connections were created among only 150 users. The solution is to specify the option

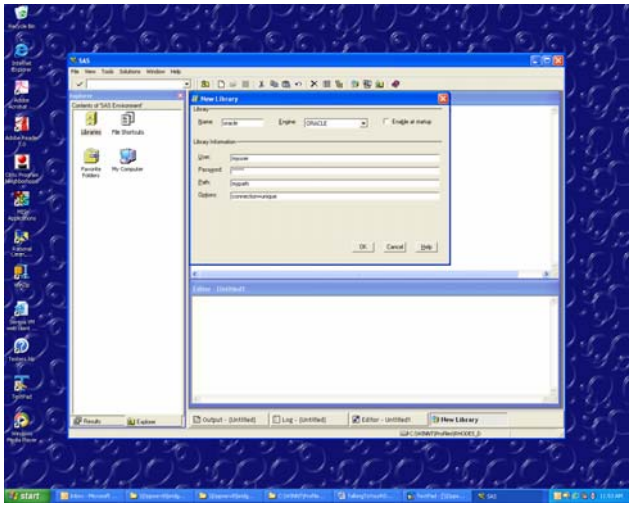
connection=unique on the libname . This way, it only opens and closes each table as each table is referenced. To improve performance, specify Readbuff=1000. This activates a feature in SAS to exploit the native API-controlled multi-row read capabilities (Levin, 2001). Multi-row writes are similarly activated by setting insertbuff=100.

LIBNAME OPTIONS

DBPROMPT set to "YES" prompts you for the password or other information.

PRESERVE_COL_NAMES set to "YES" allows the use of column names that are not SAS compliant. You need to use 'column-name'N notation to access them. PRESERVE_TAB_NAMES performs similar functions for the table names.

USING THE NEW LIBRARY WINDOW



You get to the New Library Window by issuing DMLIBASSIGN from the command line, or by right-clicking on the file cabinet icon in SAS Explorer toolbar. This technique is useful if you don't know all the ins and outs of allocating a new library, and find the prompts in the window helpful.

USING SQL

SQL statements are passed to the RDBMS whenever possible. This reduces data movement, which can improve performance. This can be significant when you are accessing large RDBMS tables and the SQL subsets the table to reduce the amount of rows. SAS/ACCESS sends operations to the RDBMS for processing when you:

- Submit RDBMS-specific SQL statements that are sent directly to the RDBMS for execution. For example, when you submit an EXECUTE statement.
- Use the SAS/ACCESS LIBNAME statement, you submit SAS statements that are translated into SQL that are RDBMS specific.

By using the automatic translation abilities, you can often achieve the performance benefits of the Pass-Through Facility without needing to write RDBMS-specific SQL code.

Certain conditions prevent operations from being passed to the RDBMS. For example, when you use an INTO clause or any data set options (eg., KEEP= or DROP=), operations are processed in SAS instead of being passed to the RDBMS. Re-merges, union joins, and truncated comparisons also are processed in SAS instead of the RDBMS.

When you join tables across multiple tables, implicit pass through utilizes the first connection; subsequent connections are ignored.

You can use the SASTRACE= System Option to determine whether an operation is processed by SAS or is passed to the RDBMS for processing. More on SASTRACE, below.

To prevent operations being passed to the RDBMS, use the LIBNAME option DIRECT_SQL=NO.

USING SQL PASS THROUGH

The Pass-Through Facility uses SAS/ACCESS to connect to a RDBMS and to send statements directly to the RDBMS for execution. This facility is an alternative to the SAS/ACCESS LIBNAME statement. It enables you to use the SQL syntax of your RDBMS, and it supports any non-ANSI standard SQL that is supported by your RDBMS.

PROS

- The RDBMS optimizer can optimize queries, taking advantage of indexes
- Optimizes queries that use summary functions (Count, AVG, etc.)
- Using SAS/AF you can use Commit and Rollback transactions

CONS

- Threaded reads are not supported where RDBMS-specific SQL is passed directly to the RDBMS for processing.

With SQL Pass-Through you can:

- Establish and terminate connections with a RDBMS using CONNECT and DISCONNECT
- Send dynamic, non-query, RDBMS-specific SQL statements to a RDBMS using EXECUTE
- Retrieve data directly from a RDBMS using CONNECTION TO in the FROM clause of a PROC SQL SELECT statement.

You can use Pass-Through Facility statements in a PROC SQL query or you can store them in an SQL view. When you create an SQL view, any arguments that you specify in the CONNECT statement are stored with the view. Therefore, when the view is used in a SAS program, SAS can establish the appropriate connection to the RDBMS.

SYNTAX

PROC SQL <options>;

CONNECT TO ***DBMS-NAME*** **<AS *ALIAS*> <(<*DATABASE-CONNECTION-ARGUMENTS*> <*CONNECT-STATEMENT-ARGUMENTS*>)>;**

DISCONNECT FROM ***DBMS-NAME*** **|** ***ALIAS*****;**

EXECUTE **(*DBMS-SPECIFIC-SQL-STATEMENT*) BY *DBMS-NAME* | *ALIAS*;**

SELECT *COLUMN-LIST* FROM CONNECTION TO ***DBMS-NAME* | *ALIAS (DBMS-QUERY)***

EXAMPLE

```
proc sql;
 connect to oracle (user= &user password= &PWD path= &PATH )  ;
 exec(alter table official_index disable constraint FK1_INDEX_SID) by oracle;
 exec(truncate table official_index) by oracle;
 exec (alter table official_index enable constraint FK1_INDEX_SID) by oracle;
disconnect from oracle ;
quit;
```

Note that when you want to execute a stored procedure you need to specify

```
Exec (exec stored_procedure) by oracle;
```

SYSTEM TABLES

You can access some of the system tables to get metadata . For example, if you did not know the names of the stored procedures, indexes, or functions, they can be found in user_objects. You can look in user_tab_columns to get the names of the columns in your table, This example uses a correlated subquery. This technique is required when selecting variables from a connection to Oracle.

EXAMPLE

```
proc sql  ;
 connect to oracle (user= &user password= &PWD path= &PATH )  ;
 select * from connection to oracle
(select table_name, column_name from user_tab_columns
where table_name = 'MY_INDEX' );
disconnect from oracle ;
quit;
```

Some systems have a master data_dictionary table, but I could not access one with my user privileges. Ask your Oracle DBA.

DEBUGGING TECHNIQUES

Any error return codes and error messages are written to the SAS log. These codes and messages are in the following two SAS macro variables:

SQLXRC contains the RDBMS return code .
SQLXMSG contains a description of the RDBMS error.

The contents of the SQLXRC and SQLXMSG macro variables can be printed to the SAS log using the %PUT macro statement. They are reset after each Pass-Through Facility statement has been executed.

EXAMPLE

```
%macro oracon ;
proc sql;
connect to oracle as oracon1
   (server=SERVER1 database=PERSONNEL
    user=testuser password=testpass
    connection=global);
%put &sqlxmsg &sqlxrc;
Quit ;
%mend oracon ;
%oracon
```

DEBUGGING TECHNIQUES - SASTRACE

SASTRACE= ',,,d' | ' ,,d,' | ' d,' | ' ,,,s' | ' ,,,sa' | ',,t,'

Prior to Version 9, SASTRACE was available to show the SQL statements that were being generated and sent to the RDBMS. The syntax is

',,,d'

The statements sent to the log include SELECT, CREATE, DROP, INSERT and UPDATE.

With Version 9, the following new options are available**.**

',,d,'

All routine calls are sent to the log. All function enters and exits, as well as pertinent parameters and return codes, are traced**.**

'd,'

All RDBMS calls, such as API and Client calls, connection information, column bindings, column error information, and row processing are sent to the log.

',,,s'

A summary of timing information for calls made to the RDBMS is sent to the log.

',,,sa'
Timing information for each call made to the RDBMS, along with a summary, is sent to the log.
',,t,'

All threading information is sent to the log. This information includes the number of threads spawned, the number of observations each thread contains, and the exit code of the thread, should it fail.

Use ‘,,d,’ or ‘d,’ or some combinations of the two ‘d,,d,’ when you are having a problem and need to send a SAS log to technical support for troubleshooting**.**

When using SASTRACE= on PC platforms, you must also specify SASTRACELOC=.’File_name’ or SASLOG

Turning the tracing off in version 8.2 was somewhat cumbersome. In version 9 you can simply specify:

```
options sastrace=off;
```

Log output is much easier to read if you specify NOSTSUFFIX. (NOSTSUFFIX is not supported on MVS.)

EXAMPLE

```
7          options sastrace=',,,d' sastraceloc=saslog nostsuffix ;
8
9
10         libname oracle oracle
11         path= &PATH
12         schema= &SCHEMA
13         user= &USER
14         password= &PWD connection=unique;
NOTE: Libref ORACLE was successfully assigned as follows:
      Engine:        ORACLE
      Physical Name: oradev
15
16         /* EXTRACTING THE LATEST DATA ELEMENTS INTO WORK LIBRARY FROM JANUS
17         TABLES in ORACLE */
18

ORACLE_1: Prepared:
SELECT * FROM path1.TREE_VERSION

19         data tree_version_copy;
20         set oracle.tree_version;
21         length  TCVT $10
22                     TCV $9
23                     TC $8
24                     VERSION_NUM_temp $1;
25         VERSION_NUM_temp = put(VERSION_NUM,1.);
26         TCVT = TREE_ID||TREE_CYCLE_CODE||VERSION_NUM_temp||tree_type;
27         TCV = TREE_ID||TREE_CYCLE_CODE||VERSION_NUM_temp;
28         TC = TREE_ID||TREE_CYCLE_CODE;
29         run;


ORACLE_2: Executed:
SELECT statement  ORACLE_1


NOTE: There were 10708 observations read from the data set ORACLE.TREE_VERSION.
NOTE: The data set WORK.TREE_VERSION_COPY has 10708 observations and 27 variables.
NOTE: DATA statement used (Total process time):
      real time           5.67 seconds
      cpu time            0.31 seconds
```

MORE EXAMPLES

More verbose examples may be found in Appendix A.

ORACLE HINTS

HOW TO USE HINTS

Oracle hints look like SAS PL/1 style comments; you need to specify “Preserve_comments” in your connection to Oracle (either in the LIBNAME statement or in the connect statement). Oracle hints are provided by Oracle, not SAS. You can find a complete list at the Oracle website (Oracle.com). Hints are for optimization, access, join order, joint operations, and parallel execution. These can be optimization hints, access paths, cluster, full, has, rowed, index.

EXAMPLE

```
Libname Oracle Oracle "\my path" preserve_comments  ;

Proc sql ;
Create table joined_t as
Select (ORHINTS='/* + full(t) parallel(c,2) */')
c.customer, c.key, t.card
from extra_cust c
customer_detail t
;
```

WHY USE HINTS?

Shouldn't the Oracle optimizer (Cost Based Optimizer or CBO) be sufficient? Not if your data are changing rapidly and the Oracle statistics are out of date. When the statistics are ideal, Oracle locks them, making the hints superfluous.

COMMON HINTS

ALL_Rows – chooses a cost-based approach (optimization) to reduce the resources used to read an entire table.
Example:

```
Data Employees ;
Set  Oracle.Employee_units
     (ORHINTS = '/*+ All_rows */') ;
Run;
```

INDEX tells Oracle to use a specific index with a table.

```
Data Employees ;
Set  Oracle.Employee_units
     (ORHINTS = '/*+ Index(Employee_units, EU_PK */') ;
By Emp_id ;
Run;
```

PARALLEL tells Oracle to use multiple processors

```
Data Employees ;
Set  Oracle.Employee_units
     (ORHINTS = '/*+ PARALLEL (Employee_units, 8) */') ;
By Emp_id ;
Run;
```

This hint requires you to specify the Oracle table and the number of processors (8 in the above example). See Chapman(2006) and the Oracle web site for more hints and examples.

ORACLE INDEXES

Using Indexes in Oracle can result in significant time savings as well as resources. It serves as an alternative to sorting data and retrieving subsets of data (see Raithel, 2006). The Oracle optimizer figures out which index to use, but you can assist it by identifying the available indexes and which to use.

When an index exists, using a BY statement causes the data to be retrieved by index, which eliminates the need to sort (or sorts behind the scenes). See Chapman (2005) for benchmarks using un-indexed vs. indexed data.

The LIBNAME option DBINDEX set to "YES" tells SAS to contact Oracle to determine if there is an Oracle index.

LOADING DATA

When you need to load data into your permanent Oracle RDBMS, there are four approaches:

- Proc DBLOAD
- Data Step
- Proc SQL
- Proc Append

USING DBLOAD

Dbload uses transaction update, a standard Oracle insert. The syntax is:

```
Proc dbload dbms=oracle data=mydata ;
```

```
Path = &path; user=&user; pw=&pwd; table=temp ;
Commit=10000;
Limit=0 ;
Load;
Run;
```

Use a large number for the commit parameter. This tells Oracle how many rows to process before a commit. This makes it run faster, but may not be the safest approach if your data are not clean.

USING A DATA STEP

Using a libname to connect to Oracle, you can use any valid SAS Data step to load your data.

EXAMPLE

```
Data Oracle.Myoratable ;
  Set mywork_data ;
Run;

Or

Data Mywork_data ;
  Set Oracle.myoratable ;
Run;
```

USING SQL

Using a libname to connect to Oracle, you can use SQL to load your data.

```
Proc sql ;
Create table oracle.mytable as
Select * from mywork.data ;
Quit;
```

Or you can build an insert statement

```
Proc sql ;
Insert into oracle.mytable (var1, var2)
Select var1, var2 from mywork.data
Where var1 = '1'
;
```

USING APPEND

Proc append is faster than SQL and you can use the force option if your data are somewhat different.

```
Proc append base=oracle.mytable data=mydata ;
Run;
```

USING BULKLOAD

SAS/ACCESS engine for Oracle can call Oracle's native load utility SQL*Loader. The option DIRECT=TRUE is activated to execute a direct path load for optimal performance. BULKLOAD can be use with a Data Step, Proc SQL, or Proc Append. The best performance is with Proc append.

```
Proc append base=oracle.mytable
(BULKLOAD=YES
 BL_OPTIONS='ERRORS=50'
 BL_DELETE_DATAFILE=YES
)
data=mydata ;
Run;
```

SAS will generate a control file (.CNT) and a text datafile (.DAT) or you can supply your own. Specifying BL_DELETE_DATAFILE=YES will delete them after a successful load operation. (This is the default behavior). The .DAT file can be quite large.

To supply your own CNT and DAT files, use the Dataset options

- BL_CONTROL='file location\textfile.cnt'
- BL_DATA='file location\textfile.dat'

These are data set options and can be specified on any dataset assigned to an Oracle library.

USING THREADS

In Version 8 and earlier, SAS opened a single connection to the RDBMS to read a table. SAS statements requesting data were converted to an SQL statement and passed to the RDBMS. The RDBMS processed the SQL statement, produced a result set consisting of table rows and columns, and transferred the result set back to SAS on the single connection.

With a threaded read, the table read time can be reduced by retrieving the result set on multiple connections between SAS and the RDBMS. SAS is able to create multiple threads, and a read connection is established between the RDBMS and each SAS thread. The result set is partitioned across the connections, and rows are passed to SAS simultaneously (in parallel) across the connections, improving performance.

SAS steps called threaded applications are automatically eligible for a threaded read. Threaded applications are bottom-to-top fully threaded SAS procedures that perform data reads, numerical algorithms, and data analysis in threads. Only some SAS procedures are threaded applications. Here is a basic example of PROC REG, a SAS threaded application:

```
libname oracle oracle user=&user password=&pwd;
proc reg simple
data=oracle.salesdata (keep=salesnumber maxsales);
var _all_;
run;
```

There are two options in SAS/ACCESS for threaded reads from RDBMSs: DBSLICE= Data Set Option and DBSLICEPARM= Data Set Option.

DBSLICE= is a data set option which lets you code WHERE clauses to partition table data across threads, and is useful when you are familiar with your table data. For instance, if your RDBMS table has a CHAR(1) column Gender, and your clients are approximately half female, Gender equally partitions the table into two parts. Therefore, an example DBSLICE= might be:

```
proc print data=oracle.dbtable (dbslice=("gender='f'" "gender='m'"));
where dbcol>1000;
run;
```

SAS creates two threads and about half of the data is delivered in parallel on each connection.

Using DBSLICEPARM=ALL, SAS attempts to "autopartition" the table for you. DBSLICEPARM=ALL extends threaded reads to more SAS procedures, specifically steps that only read tables. It can be specified as a data set option, LIBNAME option, or as a global SAS option. See further examples in Appendix A.

USING ACCESS DESCRIPTORS AND VIEWS

CONVERT USING CV2VIEW

If you have legacy SAS/ACCESS descriptors and views, you should convert them. The CV2VIEW procedure converts SAS/ACCESS view descriptors and access descriptors into SQL views. You should consider converting your descriptors because:

- Descriptors are no longer the recommended method for accessing relational database data. Converting to SQL views enables you to use the LIBNAME statement instead of PROC SQL. This is usually faster. The LIBNAME statement provides greater control over RDBMS operations such as locking, spooling, and data type conversions. The LIBNAME statement can also handle long field names, whereas descriptors cannot.
- SQL views are platform-independent. SAS/ACCESS descriptors are not.

The CV2VIEW procedure in SAS 9.1 will convert both 64-bit SAS/ACCESS view descriptors (created in either 64-bit SAS Version 8 or 64-bit SAS 9.1) and 32-bit SAS/ACCESS access descriptors (created in 32-bit SAS Version 6 and Version 8).

If the descriptor that you want to convert is READ, WRITE, or ALTER protected, then those properties are applied to the output SQL view. For security reasons, these values do not appear if you save the generated SQL to a file. The PASSWORD portion of the LIBNAME statement is also not visible. This prevents the generated SQL statements from being manually submitted without modification.

EXAMPLES

Converting One View Descriptor

CV2VIEW converts the view descriptor and the SQL statements that are generated and are saved to the view.

Syntax:

```
Libname input 'my-descriptors';
Libname output 'my sql views';
Proc CV2VIEW dbms = oracle;
From_view = input.myview (alter=apwd) ;
To_view   = output.newview;
Saveas    = 'username\vsql\sql.sas';
Submit ;
Replace file;
Run;
```

This code generates the SQL to create the view. The Replace File statements overwrites the existing file SQL.SAS; otherwise the new code is appended to the file. With the Oracle rdbms, it adds preserve_tab_names to the embedded libname.

Converting a Library of View Descriptors ;

```
Libname input 'my-descriptors';
Libname output 'my sql views';
Proc CV2VIEW dbms = oracle;
From_libref = input ;
To_libref   = output ;
Saveas    = 'username\vsql\manyviews.sas';
Submit ;
Run;
```

SAVEAS causes all the generated SQL to be stored in manyviews.sas

CONCLUSION

SAS version nine has added a lot of functionality that makes it easier to work with RDBMS. The powerful debugging techniques make it easier to find your mistakes (and sometimes SAS mistakes). The variety of ways to access the data give a flexible approach which can accommodate everyone from SQL-Heads to Data Step Diehards.

REFERENCES

Chapman, David D. (2006). "Using the Oracle Libname Engine to Reduce the Time it Takes to Extract Data from an Oracle Database." Proceedings, NESUG 17.

Fogelman, Stanley (2006). "SAS and Oracle PL/SQL: Partners or Competitions." Proceedings, NESUG 17.

Leconte, Olivier (2005). "Do you need second sight to use the Libname Oracle." Proceedings: First Pharmaceutical Users Software Exchange (PHUSE).

Levin, Lois (2005). "Methods of Storing SAS Data into Oracle Tables." Proceedings, NESUG 16

Levine, Freed (2001) "Using the SAS/ACCESS Libname Technology to Get Improvement in Performance and Optimizations in SAS/SQL Queries." Proceedings, SUGI 26.

Loren, Judy (2003). "SAS /ACCESS to External Databases: Wisdom for the Warehouse User." Proceedings, SUGI 28.

Olsen, Diana and David Wiehle (2003). "Proc Migrate: How to Migrate Your Data and Know You've Done it Right." Proceedings, SUGI 28.

Plemmons, Howard and Andrew Holdsworth (2003). "Scaling SAS Data Access to Oracle RDBMS." Proceedings, SUGI 28.

Raithel, Michael (2006). The Complete Guide to SAS Indexes. SAS Press.

SAS OnLine Documentation, Version 9.1.3

Whittier, Sarah (2006). "Multi-Threaded Reads in SAS/Access for Relational Databases." Proceedings, NESUG 17.

ACKNOWLEDGMENTS

I would like to thank my employer, BAE Systems, and specifically Todd Baylor for encouraging me to participate in the SAS Global Forum.

CONTACT INFORMATION

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

Dianne Louise Rhodes
BAE Systems IT
1260 21st St NW
Apartment 305
Washington, DC 20036
(202) 691-6338
Dianne.rhodes@baesystems.com
Rhodes.dianne@bls.gov

Appendix A

Basic SQL statements

```
7          options sastrace=',,,d' sastraceloc=saslog nostsuffix ;
8
9
10         libname oracle oracle
11         path= &PATH
12         schema= &SCHEMA
13         user= &USER
14         password= &PWD connection=unique;
NOTE: Libref ORACLE was successfully assigned as follows:
      Engine:        ORACLE
      Physical Name: oradev
15
16         /* EXTRACTING THE LATEST DATA ELEMENTS INTO WORK LIBRARY FROM JANUS
17         TABLES in ORACLE */
18

ORACLE_1: Prepared:
SELECT * FROM path1.TREE_VERSION

19         data tree_version_copy;
20         set oracle.tree_version;
21         length  TCVT $10
22                     TCV $9
23                     TC $8
24                     VERSION_NUM_temp $1;
25         VERSION_NUM_temp = put(VERSION_NUM,1.);
26         TCVT = TREE_ID||TREE_CYCLE_CODE||VERSION_NUM_temp||tree_type;
27         TCV = TREE_ID||TREE_CYCLE_CODE||VERSION_NUM_temp;
28         TC = TREE_ID||TREE_CYCLE_CODE;
29         run;


ORACLE_2: Executed:
SELECT statement  ORACLE_1

NOTE: There were 10708 observations read from the data set ORACLE.TREE_VERSION.
NOTE: The data set WORK.TREE_VERSION_COPY has 10708 observations and 27 variables.
NOTE: DATA statement used (Total process time):
      real time           5.67 seconds
      cpu time            0.31 seconds
```

Threaded information and Basic SQL Statements

```
30
31         options sastrace=',,t,d' ;

ORACLE_3: Prepared:
SELECT * FROM path1.TREE_VERSION

32         proc print data=oracle.tree_version(dbsliceparm=(all,3)) ;
33         where tree_category='23';
34         var tree_category tree_id tree_type ;
35         run;

ORACLE:  DBSLICEPARM option set and 3 threads were requested
ORACLE:  No application input on number of threads.

ORACLE_4: Executed:
```

```
SELECT  "TREE_CATEGORY", "TREE_ID", "TREE_TYPE" FROM PATH1.TREE_VERSION  WHERE
("TREE_CATEGORY" = '23' ) AND  ABS(MOD("TREE_VERSION_SID", 3))=0


ORACLE_5: Executed:
SELECT  "TREE_CATEGORY", "TREE_ID", "TREE_TYPE" FROM PATH1.TREE_VERSION  WHERE
("TREE_CATEGORY" = '23' ) AND  ABS(MOD("TREE_VERSION_SID", 3))=1

ORACLE_6: Executed:
SELECT  "TREE_CATEGORY", "TREE_ID", "TREE_TYPE" FROM PATH1.TREE_VERSION  WHERE
("TREE_CATEGORY" = '23' ) AND  ABS(MOD("TREE_VERSION_SID", 3))=2


ORACLE:  Thread 3 contains 1206 obs.
ORACLE:  Thread 1 contains 1193 obs.
ORACLE:  Thread 2 contains 1169 obs.
ORACLE:  Threaded read enabled. Number of threads created: 3
NOTE: There were 3568 observations read from the data set ORACLE.TREE_VERSION.
      WHERE tree_category='23'
```

All RDBMS Calls, Basic SQL

```
7          options sastrace=',,d,d' sastraceloc=saslog nostsuffix ;
8
9
10         libname oracle oracle
11         path= &PATH
12         schema= &SCHEMA
ORACLE: Try to load SASORA appendage: SASORA
ORACLE: orinit()
ACCESS ENGINE:  Entering dbiconi.
ACCESS ENGINE:  Exiting dbiconi.  rc=0x00000000
ORACLE: orkwin()
ACCESS ENGINE: Entering DBICON
ACCESS ENGINE:  CONNECTION= SHAREDREAD
ORACLE: orcon()
ACCESS ENGINE: Successful physical conn id 0
ACCESS ENGINE: Number of connections is 1
ACCESS ENGINE: Exiting DBICON with  rc=0X00000000
13         user= &USER
14         password= &PWD;
NOTE: Libref ORACLE was successfully assigned as follows:
      Engine:        ORACLE
      Physical Name: oradev
15
16         /* EXTRACTING THE LATEST DATA ELEMENTS INTO WORK LIBRARY FROM JANUS
17         TABLES in ORACLE */
18
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
ORACLE: orqall()
ACCESS ENGINE:  Entering dbidsci()
ORACLE: ordesci()
ORACLE: orprep()
```

```
ORACLE_1: Prepared:
SELECT * FROM path1.TREE_VERSION

ORACLE: ornvar()
ORACLE: No. of columns : 23
ORACLE: ordesca()
ORACLE: ordesc()
ORACLE: ordtfmt()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
```

```
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ORACLE: orcolnm()
ACCESS ENGINE: Generated name/label pools
ACCESS ENGINE:  Exiting dbidsci with success
ACCESS ENGINE: Exit yoeopen with SUCCESS.
ACCESS ENGINE:  Begin yoeinfo
ACCESS ENGINE: Exit yoeinfo with SUCCESS.
ACCESS ENGINE:  Entering yveprjl
ACCESS ENGINE:  Exiting yveprjl with rc=0x00000000
ACCESS ENGINE:  Begin yoeinfo
ACCESS ENGINE: Exit yoeinfo with NOSUPPORT.
ACCESS ENGINE:  Entering yveallv
ACCESS ENGINE:  Entering yveprjd, select variable number 1
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 2
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 3
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 4
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 5
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 6
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 7
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 8
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 9
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 10
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 11
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 12
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 13
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 14
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 15
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 16
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 17
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 18
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 19
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 20
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 21
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Entering yveprjd, select variable number 22
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
```

```
ACCESS ENGINE:  Entering yveprjd, select variable number 23
ACCESS ENGINE:  Exiting yveprjd with rc=0x00000000
ACCESS ENGINE:  Exiting yveallv with rc=0x00000000
ACCESS ENGINE:  Entering yveprje, terminating variable projection
ACCESS ENGINE:  Exiting yveprje with rc=0x00000000
ACCESS ENGINE: Exiting yoepnt() current_rid=0, next_rid=1
19         data tree_version_copy;
20         set oracle.tree_version;
21         length  TCVT $10
22                     TCV $9
23                     TC $8
24                     VERSION_NUM_temp $1;
25         VERSION_NUM_temp = put(VERSION_NUM,1.);
26         TCVT = TREE_ID||TREE_CYCLE_CODE||VERSION_NUM_temp||tree_type;
27         TCV = TREE_ID||TREE_CYCLE_CODE||VERSION_NUM_temp;
28         TC = TREE_ID||TREE_CYCLE_CODE;
29         run;

ACCESS ENGINE:  Begin yoeinfo
ACCESS ENGINE: Exit yoeinfo with SUCCESS.
ACCESS ENGINE:  yoeget before read, current row=0, next row=1
ORACLE: orgbfi()
ORACLE: orubuf()
ORACLE: orlock()

ORACLE_2: Executed:
SELECT statement  ORACLE_1

ACCESS ENGINE:  yoeget read from next>=cur, current =0, next=1
ACCESS ENGINE:  yoeget: current row id=1, next row id=2
NOTE: There were 10708 observations read from the data set ORACLE.TREE_VERSION.
ACCESS ENGINE:  Enter yoeclos
ORACLE: orgbft()
ORACLE: ordesct()
ACCESS ENGINE: Entering dbiclose
ORACLE: orclose()
ACCESS ENGINE: DBICLOSE open_id 0, connect_id 0
ACCESS ENGINE: Exiting dbiclos with rc=0X00000000
ACCESS ENGINE: Successful logical disconnect, id 0
ACCESS ENGINE: Exit yoeclos with rc=0x00000000
NOTE: The data set WORK.TREE_VERSION_COPY has 10708 observations and 27 variables.
NOTE: DATA statement used (Total process time):
      real time           3.92 seconds
      cpu time            0.29 seconds
```