

Do you need second sight to use the LIBNAME ORACLE?

Olivier LECONTE, SERVIER, Courbevoie, France

ABSTRACT

Up to 2002, SERVIER used SAS® to validate clinical data in a Clintrial™ database. In order to have a real-time access to these databases, SAS views were created using the SQL PASS-THROUGH facilities. Once set-up, these views were also used by the Biostatisticians to make their statistical analysis.

In SAS version 8, LIBNAME ORACLE was introduced as a new way to access ORACLE data. It allows users to access ORACLE databases as if they were regular datasets and to execute any SAS procedure on that data.

This paper describes how SERVIER implemented this new functionality, the issues encountered, and the choices made to resolve them.

BASICS

The following information is required to access a specific ORACLE database using LIBNAME :

- the name of the ORACLE server,
- a user account and a password
- the name of an ORACLE user group which has access to the data

For the purpose of this paper, we will access the server BASE1 with the account USER1 and password PW_USER. We will work on panels granted by the ORACLE user group STUDY1. In order to access these data from SAS®, you can enter the following code :

```
LIBNAME mylib ORACLE USER=user1 pw=pw_user schema=STUDY1 path='BASE1';
```

You can then access data from any panel with a simple DATA step. If your account gives you INSERT or DELETE privileges, you can update data directly from SAS® using an SQL procedure.

This solution is simpler than the SQL PASS-THROUGH technique which obliges you to create a view for each panel you want to access. It is also faster. For a middle-sized panel (52000 observations and 60 items), the CPU TIME is reduced from 50% and the REAL TIME is divided by 4. The explanation for this speediness is given later. This solution is also more secure, because the user account/password is not enclosed in each SQL statement and can be centralized in an autoexec file.

Because of these features, we decided to replace all our SQL views with views using ORACLE libnames :

```
proc sql;
    connect to ORACLE (user=USER1
pw=PW_USER1 path='CBDF18');
    create view BIOVAL as
    select *
    from connection to ORACLE (select
* from STUDY1.BIOVAL_DATA);
    disconnect from ORACLE;
quit;
```

```
proc sql;
    create view BIOVAL as
    select *
    from MYLIB.BIOVAL_DATA;
quit;
```

Our main error at this step was to not inquire about all of the options described in the documentation. As a SAS guru once said: if there is an option documented, it is because there is a good reason to use it.

AN UNSUSPECTED ADVERSE REACTION CAUSED BY LIBNAME

Our first issue was that, at variable times during the day, we were unable to connect to the server. There was no obvious reason. We contacted the ORACLE administrator who explained that more than 600 ORACLE connections were activated, although there were fewer than 150 SAS® users. After a few calls to SAS® technical support, we understood that each time a ORACLE libname was set up, a permanent connection was made to ORACLE server. So if a user had two SAS sessions opened, or wanted to access two different studies in the same session, two permanent connections were set up.

The solution recommended by SAS® Support was to use the option CONNECTION, which allows control over the way that SAS® connects to ORACLE.

```
LIBNAME mymib ORACLE USER=user1 pw=pw_user schema=STUDY1 path='BASE1'  
connection=unique;
```

Using the value UNIQUE, SAS® creates a connection each time a table is opened and closes it once the table is closed. Later, we discovered that the CONNECTION=UNIQUE did not directly solve the issue. The real solution was the option DEFER which controls the time when SAS connects to ORACLE. The DEFER option is set to YES by CONNECTION=UNIQUE, so that the connection to ORACLE occurs only when the table is opened.

The second issue was detected by a Biostatistician. In applying a WHERE clause when working directly with a n view, he observed results that were different from those obtained applying a WHERE clause on a dataset containing the same data.

<pre>/* WHERE clause included in the SQL */ 580 proc sql; 581 create table bioval as 582 select * 583 from mylib.bioval_data 590 where dsampl ne 1 591 order by patident, visit ; NOTE: Table WORK.BIOVAL created, with 48780 rows and 59 columns. 592 quit;</pre>	<pre>/* WHERE clause in a data step */ 595 proc sql; 596 create table bioval as 597 select * 598 from mylib.bioval_data 599 order by patident, visit ; NOTE: Table WORK.BIOVAL created, with 52094 rows and 59 columns. 600 quit; 601 602 data bioval2; 603 set bioval; 604 where dsampl ne 1; 605 run; NOTE: There were 50593 observations read from the data set WORK.BIOVAL2. WHERE dsampl not = 1;.</pre>
--	--

This issue was due to the fact that when submitting SAS code on ORACLE data, SAS translates this code into SQL code and sends it to the DBMS server for processing. In our example, the WHERE clause is then processed by the DBMS server and follows ORACLE SQL syntax rules which means that the applied WHERE clause is DSAMPL NE 1 AND IS NOT NULL, which explains the difference.

We had an another example of a valid SAS code which was invalid in ORACLE SQL. We tried to create a view which was the union of two panels. We wanted to create a variable which indicated the panel from which the data came. We had no problem to create and use this view in SAS®:

```

4      proc sql;
5          create view bioval as
6          select 'D' as _flag_,*
7          from mylib.bioval_data
8
9          union
10
11         select 'U' as _flag_,*
12         from mylib.bioval_update
13
14         where dsampl ne 1
15         order by patident, visit ;
NOTE: SQL view WORK.BIOVAL has been defined.
16      quit;
17
18      data bioval2;
19          set bioval;
20      run;
NOTE: There were 0 observations read from the data set MYLIB.BIOVAL_DATA.
NOTE: There were 48780 observations read from the data set MYLIB.BIOVAL_UPDATE.
      WHERE DSAMPL not = 1;
NOTE: There were 48780 observations read from the data set WORK.BIOVAL.
NOTE: The data set WORK.BIOVAL2 has 48780 observations and 60 variables.

```

However, the ORACLE administrators informed us that there were many abnormal ghost connections to the ORACLE server. We then used the SASTRACE option which provides information on the SAS/ACCESS engine calls to a DBMS server :

```

options SASTRACE=',,,d'
      sastraceloc=file "\\cbsw22\sas\USERS\OLTE_CB\log1.log";

```

Using SASTRACE, we discovered that ORACLE cannot handle an item beginning by an underscore :

```

DEBUG: PREPARE SQL statement:  28 1441539158 no_name 0 DATASTEP
      select 'D' as _flag_, CL306790008.BIOVAL_DATA."MERGE_DATETIME",
      union select 'U' as _flag_, CL306790008.BIOVAL_UPDATE."RESU_E",
from CL306790008.BIOVAL_UPDATE where CL306790008.BIOVAL_UPDATE."DSAMPL" <> 1 29
1441539158 no_name 0 DATASTEP
DEBUG: Close Cursor - CDA=53909824 30 1441539158 no_name 0 DATASTEP
TRACE: DBMS engine returned an error - NO Implicit Passthru 31 1441539158 no_name 0 DATASTEP
DEBUG: PREPARE SQL statement:  32 1441539158 no_name 0 DATASTEP
      SELECT  "MERGE_DATETIME", "STATUS", "ENTRY_ID", "ENTRY_DATETIME", "CT_RECID",
"DB_ID",

```

With all these issues in mind, we began to dig deeper into the documentation looking for more information concerning the way SAS handles the connection with the ORACLE server.

We found the good option : DIRECT_SQL.

THE DIRECT_SQL OPTION

DIRECT_SQL is an option of the LIBNAME statement which enables the user to specify if generated SQL is passed to the DBMS for processing. Using this option, you can decide what will be executed by the ORACLE server and what will be executed by SAS®.

The DIRECT_SQL option can have the following values :

YES	Specifies that generated SQL from PROC SQL is passed directly to the DBMS for processing (default value).
NO	Specifies that generated SQL from PROC SQL is not passed to the DBMS for processing. This is the same as specifying the value NOGENSQL.
NONE	Specifies that generated SQL is not passed to the DBMS for processing. This includes SQL that is generated from PROC SQL, SAS functions that can be converted into DBMS functions, joins, and WHERE clauses.
NOGENSQL	Prevents PROC SQL from generating SQL to be passed to the DBMS for processing.
NOWHERE	Prevents WHERE clauses from being passed to the DBMS for processing. This includes SAS WHERE clauses and PROC SQL generated or PROC SQL specified WHERE clauses.

For the WHERE clause issue, the use of DIRECT_SQL=NOWHERE can force SAS to perform the WHERE clause :

```
67      libname libora oracle user= USER1 pw=XXXXXXXXXXXX schema= STUDY1
path='BASE1'
67 ! access=readonly connection=unique direct_sql=nowhere;
NOTE: Libref LIBORA was successfully assigned as follows:
      Engine:          ORACLE
      Physical Name: cbdpl8
68
69      proc sql;
70          create table bioval3 as
71              select *
72              from libora.bioval_data
73              where dsample ne 1
74              order by patient, visit ;
NOTE: Table WORK.BIOVAL3 created, with 50801 rows and 59 columns.
```

The results were then the same as with a DATA step. However the use of a variable with an underscore was still a problem. In addition, the real time was 6 times longer because all of the data were transferred to SAS® and then selected.

For the underscore issue, the use of DIRECT_SQL= NOGENSQL allowed us to create variables beginning with an underscore. The performances were still good. However the problem with the WHERE clause remained.

The final decision was to use the option DIRECT_SQL=NOGENSQL. With this option, we can create and use variables with an underscore in their name in all our views, with no effect on performance. The problem with the WHERE clause still remains. An audit of users and standard programs showed that negative WHERE clauses were not used very often. However we are obliged to be very careful with WHERE clauses and to explain this issue to all users.

IN SAS 9 ?

The SAS 9 provides two new values for the DIRECT_SQL option.

NOFUNCTIONS	prevents SQL statements from being passed to the DBMS for processing when they contain functions.
NOMULTOUTJOINS	specifies that PROC SQL will not attempt to pass any multiple outer joins to the DBMS for processing. Other join statements may be passed down however, including portions of a multiple outer join.

Of course the use of these options will affect performance. However, we have not identified any difference in the way data are merged, for example using an SQL between SAS and ORACLE

If you want to improve the performance, you can try the new option DBSLICEPARM which controls the scope of DBMS threaded reads and the number of threads. If it takes the values THREADED_APPS or ALL, you can set the number of connection used by SAS to read the ORACLE data. On our test server, the default option was to use two connections. By using three and four connections, we saved respectively 15% and 20% of the time required to read data. However, there may be a limit to the number of connexions allowed, and there are

diminishing returns when increasing the number of connections. In our case, using five connections was slower than using two.

CONCLUSION

With the LIBNAME ORACLE, SAS has proposed a simple and effective way to access ORACLE databases. However, the bridge between our favourite software and ORACLE is not so intuitive as we may think. There are many options described in the documentation. Each one must be understood before starting to use this engine in order to ensure that the results are the same, whatever SAS code you are using.

REFERENCES :

SAS OnlineDoc version Eight- SAS/ACCESS Software for Relational Databases: Reference.

SAS 9.1.3 Help and Documentation - Data Integrity and Security - Potential Result Set Differences When Processing Null Data.

CONTACT INFORMATION (HEADER 1)

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

Olivier LECONTE
Institut de Recherches Internationales SERVIER
8 Places des Pléiades
92415 Courbevoie Cedex
Work Phone: 33 1 55 72 72 08
Email: olivier.leconte@fr.netgrs.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.