An Investigation of the Efficiency of SQL DML Operations Performed on an ORACLE® DBMS using SAS/ACCESS® Software

Annie Guo, Ischemia Research and Education Foundation, San Francisco, California

Abstract

In an international epidemiological study of 2000 cardiac surgery patients, the data of 7000 variables are entered through a Visual Basic® data entry system and stored in 57 large ORACLE® tables. A SAS® application is developed to automatically convert the ORACLE tables to SAS data sets, perform a series of intensive data processing, and based upon the result of the data processing, dynamically pass ORACLE® SQL Data Manipulation Language (DML) commands such as UPDATE, DELETE and INSERT to ORACLE database and modify the data in the 57 tables.

The modification of ORACLE data using SAS software can be resource-intensive, especially in dealing with large tables and involving sorting in ORACLE data. To select an efficient method for the SAS application, this paper focuses on the investigation of the efficiency of these four methods provided by SAS/ACCESS® software.

1. SQL Procedure Pass-Through Facility
2. DBLOAD Procedure SQL Statement
3. SQL Procedure with View Descriptor
4. DATA Step MODIFY Statement with View Descriptor

The SQL procedure pass-through facility is discovered to be the most efficient one. The second method is acceptable. The performance of the last 2 methods is unsatisfactory, because they involve view descriptors and sorting in ORACLE data.

1.0 Background

In an international, multicenter epidemiological study of about 2000 cardiac surgery patients enrolled over 3 years, each patient's Case Report Form (CRF) data of about 7000 variables are entered twice via a Visual Basic data entry system by two data entry clerks to ensure consistent quality and stored in 57 ORACLE tables. When a patient's 2 entries are completed and no discrepancy between the 2 entries is detected in any of the 57 tables, a copy of the patient's data from the first data entry clerk is inserted into the same ORACLE tables and saved as the patient's final records. After the final records are created, if the two original entries are updated in any of the 57 tables, the final records will be modified accordingly as soon as the update is completed and the 2 entries are compared equal. Table 1.1 and Table 1.2 illustrate such a scenario, before and after the modification of the final records. A grouping of the first five columns, *Id*, *Entry*, *MedCode*, *Period1* and *Period2*, is an unique identifier among records.

Table 1.1: Before Modification Of Final Records

Id	Entry	MedCode	Period1	Period2	Indication	
AG1001	Entry1	AN312	Postop	Day1	Routine	
AG1001	Entry2	AN312	Postop	Day1	Routine	
AG1001	Final	AN312	Postop	Day1	**Non-routine**	← To be updated
AG1001	**Final**	**HC527**	**Intraop**	**PostCPB**	**Maintenance**	← To be deleted
AG1002	**Entry1**	**PV946**	**Intraop**	**PreCPB**	**Non-routine**	← To be inserted as 'Final'
AG1002	Entry2	PV946	Intraop	PreCPB	Non-routine	

Table 1.2: After Modification Of Final Records

Id	Entry	MedCode	Period1	Period2	Indication	
AG1001	Entry1	AN312	Postop	Day1	Routine	
AG1001	Entry2	AN312	Postop	Day1	Routine	
AG1001	Final	AN312	Postop	Day1	**Routine**	← Updated
AG1002	Entry1	PV946	Intraop	PreCPB	Non-routine	
AG1002	Entry2	PV946	Intraop	PreCPB	Non-routine	
AG1002	**Final**	**PV946**	**Intraop**	**PreCPB**	**Non-routine**	← Inserted

A Visual Basic module making use of ORACLE Views, Functions and PL/SQL Procedures has been developed to access the ORACLE data through ODBC driver, compare the 7000 variables between the 2 entries, and modify the final records in ORACLE. The performance of the Visual Basic module is not satisfactory. It takes approximately 1 ½ minutes per patient and the total time is in linear proportion to the number of patients being processed. For example, if 100 patients' data are processed all at once, it may take up to 150 minutes to do the job.

SAS software is brought up because of its optimal performance with data processing of SAS data files. The built-in COMPARE procedure compares data in an efficient fashion. In addition, the SAS System installed at our site does not go through ODBC driver and is expected to speed up the processing.

A SAS application in place of the Visual Basic module consists of three parts. Firstly, convert data from ORACLE to SAS. Then compare the 2 data entries to produce SQL command scripts. Lastly dynamically pass the SQL commands to ORACLE and modify the final records.

Much effort has been made to optimize the first two parts. However, the performance of the SAS application can be further improved in the last part by selecting a method that modifies ORACLE data efficiently. This can make a significant difference especially in dealing with large ORACLE tables. This paper addresses the investigation of the efficiency of the 4 methods provided by SAS/ACCESS software. The advantages and disadvantages of each method will be discussed.

2.0 Objective

The goal is to achieve speed and computational efficiency in passing non-query SQL commands to ORACLE dynamically using SAS software. In short, the following 2 objectives have been identified.

- Robust performance.

The CRF data are stored in 57 tables of different sizes, up to hundreds of M-bytes. We are looking for a method that can deal with large ORACLE tables efficiently.

- Capability of handling dynamic data transaction.

The amount of data transaction passed to ORACLE is dynamic. Thus an ideal method will be able to minimize the chance of data loss caused by a DBMS failure or other errors during the processing.

3.0 Case Study

Two test tables are created in ORACLE to make a study of the 4 methods. The first table represents a large one. It is composed of 13 columns, including an unique identifier of 6 index columns, total about 780,000 rows, about 30MB. The other one represents a small table. It consists of 39 columns, including an unique identifier of 4 index columns, total 3,300 rows, about 600 KB.

Two sets of data transaction are created for each test table to assess the capability of handling different amount of data transactions. The first set is composed of 120 transactions, including 15 updates, 15 deletes and 90 inserts. The other one consists of 1200 transactions, including 150 updates, 150 deletes and 900 inserts. They are performed on 120 and 1200 unique records, respectively, so that the ORACLE data buffer cache can not help the transactions run faster.

The examples presented in Table 1.1 will be used to construct sample SAS codes throughout this paper to achieve the results in Table 1.2. The hardware set up to conduct the testing is a HP Pentium 200 MHz PC connected to a HP 9000/755 workstation server. The software installed is ORACLE 7, SAS V6.12 and SAS/ACCESS, SQL*Net V2, and HP-UX 10.*.

4.0 SQL Procedure Pass-Through Facility

The SQL procedure pass-through facility enables you to use the SAS/ACCESS interface view engine to connect SAS software to ORACLE. The following sample code shows how the connection is established with CONNECT statement. It passes non-query SQL commands to ORACLE dynamically with EXECUTE statement. The SQL commands are enclosed in parentheses and passed to ORACLE as you have typed. ORACLE checks for syntax and other errors. A message is returned if the SQL command does not execute successfully. The automatic macro variable, *&sqlxmsg*, contains the ORACLE return code and the text of the message generated by ORACLE or the SQL procedure pass-through facility. In the end, the connection is terminated with DISCONNECT statement.

```
x    'setenv SASORA V7';

proc sql;
   connect to oracle as asg
      (user=asg orapw=XXXXX path='@product');
   execute (set transaction use rollback segment r01) by asg;
   execute (update OraTable Final
               set (indication) =
                  (select indication from OraTable Entry1
                      where Entry1.id=Final.id and
                            Entry1.entry='Entry1' and
                            Entry1.medcode=Final.medcode and
                            Entry1.period1=Final.period1 and
                            Entry1.period2=Final.period2)
                  where Final.id='AG1001' and
                        Final.entry='Final' and
                        Final.medcode='AN312' and
                        Final.period1='Postop' and
                        Final.period2='Day1'
            ) by asg;
   %put &sqlxmsg;
   execute (commit) by asg;
   execute (set transaction use rollback segment r01) by asg;
   execute (delete from OraTable
               where id='AG1001' and
                     entry='Final' and
                     medcode='HC527' and
                     period1='Intraop' and
                     period2='PostCPB'
            ) by asg;
   %put &sqlxmsg;
   execute (commit) by asg;
   execute (set transaction use rollback segment r01) by asg;
   execute (insert into OraTable
               select id,
                      'Final',
                      medcode,
                      period1,
                      period2,
                      indication from OraTable
                  where id='AG1002' and
                        entry='Entry1' and
                        medcode='PV946' and
                        period1='Intraop' and
                        period2='PreCPB'
            ) by asg;
   %put &sqlxmsg;
   execute (commit) by asg;
   disconnect from asg;
   quit;
```

These ORACLE SQL UPDATE, DELETE, and INSERT commands do not involve sorting in ORACLE data. Data records are only selected with WHERE clause. In addition, the WHERE clause is composed of the index columns to make use of ORACLE optimizer. Thus it saves a great deal of resource and time.

The EXECUTE statement can give you complete control of the data processing in ORACLE. For example, the COMMIT command can be issued to save data changes and release memory right after a data transaction is finished. This can minimize the chance of data loss caused by a DBMS failure or other problems during processing. In addition, a specific ORACLE rollback segment can be assigned to the next data transaction with the SET TRANSACTION command. Usually in ORACLE there are several rollback segments of different sizes brought online to store data necessary to reverse or undo changes. Depending on availability, one will be randomly assigned to the next data transaction. When lots of data are being processed all at once, specifying a larger rollback segment can minimize the chance of a DBMS failure caused by insufficient rollback segment size.

The COMMIT and SET TRANSACTION commands are critical to the SAS application discussed in this paper, since the amount of data transaction passed to ORACLE is dynamic. For example, a SQL command associated with an ORACLE database trigger may cause a series of data transactions to be executed implicitly. Thus the 2 commands minimize the probability of a DBMS failure in case the amount of data transactions exceeds the storage parameters set up in ORACLE DBMS.

5.0 DBLOAD Procedure SQL Statement

The DBLOAD procedure enables you not only to create and load ORACLE tables with data from SAS data files, but also to submit dynamic non-query SQL commands to ORACLE for processing. The following sample code shows how to perform the same operation as in Section 3.0 using DBLOAD procedure SQL statement.

```
x    'setenv SASORA V7';

proc dbload dbms=oracle ;
   user=asg;
   orapw=XXXXX;
   path="@product";
   sql update OraTable Final
          set (indication) =
              (select indication from OraTable Entry1
                 where Entry1.id=Final.id and
                       Entry1.entry='Entry1' and
                       Entry1.medcode=Final.medcode and
                       Entry1.period1=Final.period1 and
                       Entry1.period2=Final.period2)
          where Final.id='AG1001' and
                Final.entry='Final' and
                Final.medcode='AN312' and
                Final.period1='Postop' and
                Final.period2='Day1';
   sql delete from OraTable
          where id='AG1001' and
                entry='Final' and
                medcode='HC527' and
                period1='Intraop' and
                period2='PostCPB';
   sql insert into OraTable
          select id,
                 'Final',
                 medcode,
                 period1,
                 period2,
                 indication from OraTable
             where id='AG1002' and
                   entry='Entry1' and
                   medcode='PV946' and
                   period1='Intraop' and
                   period2='PreCPB';
   run;
```

This method is similar to the SQL procedure EXECUTE statement described in section 4.0, but lacks the control of data transaction processing. For example, the COMMIT and SET TRANSACTION commands can not be passed to ORACLE with this method. In addition, no automatic macro variables such as *&sqlxmsg* will be created.

6.0 SQL Procedure with View Descriptor

An ORACLE table can be modified through a view descriptor of the table with SQL procedure. The view descriptor is created by SAS/ACCESS software that issues an ORACLE Call Interface (OCI) to ORACLE in order to connect to the database. Once the view descriptor is created, it can be used with PROCs and DATA steps similar to a SAS data file. Every time you execute a SAS program that references the view descriptor, the SAS System's ORACLE interface view engine issues dynamic SQL calls to ORACLE and the table is read. So the program's output always reflects the latest updated level of the ORACLE table.

The sample code below shows how to use SAS/ACCESS software to create the view descriptor *viewlib.OraTblV* of the ORACLE table *OraTable*. The resource spent on creating the view descriptor is not taken into account for investigating this method's efficiency. It is because the view descriptor has to be created to perform the desired data processing, despite the method chosen.

```
x    'setenv SASORA V7';
libname viewlib '~/SUGI23';

proc access dbms=oracle;
   create viewlib.OraTblA.access;   * access descriptor;
      user=asg;
      orapw=XXXXX;
      table=OraTable;
      path='@product';
      list all;
   create viewlib.OraTblV.view;     * view descriptor;
      select all;
      list view;
      subset where (id like 'AG%');
   run;
```

The sample code listed below is to perform the desired data manipulation as shown in Table 1.1 using this mthod. The option UNDO_POLICY=none must be specified. It enables you to insert or update a row with the values of another row selected from the same ORACLE table.

```
x    'setenv SASORA V7';

proc sql undo_policy=none;
   update viewlib.OraTblV Final
          set indication =
              (select indication from viewlib.OraTblV Entry1
                 where Entry1.id=Final.id and
                       Entry1.entry='Entry1' and
                       Entry1.medcode=Final.medcode and
                       Entry1.period1=Final.period1 and
                       Entry1.period2=Final.period2)
          where Final.id='AG1001' and
                Final.entry='Final' and
                Final.medcode='AN312' and
                Final.period1='Postop' and
                Final.period2='Day1';
   delete from viewlib.OraTblV
          where id='AG1001' and
                entry='Final' and
                medcode='HC527' and
                period1='Intraop' and
                period2='PostCPB';
   insert into viewlib.OraTblV
          select id,
                 'Final',
                 medcode,
                 period1,
                 period2,
                 indication from viewlib.OraTblV
          where id='AG1002' and
                entry='Entry1' and
                medcode='PV946' and
                period1='Intraop' and
                period2='PreCPB';
   quit;
   run;
```

This method lacks the control of data processing in ORACLE. In addition, the UPDATE, DELETE and INSERT statements are carried out through separate connection to ORACLE every time the view descriptor is referenced.

7.0 DATA Step MODIFY Statement With View Descriptor

DATA step MODIFY statement can be applied to the *viewlib.OraTblV* view descriptor to modify the data in ORACLE. First of all the *transctn* data file consisting of the desired data transactions listed in Table 1.1 is created via the following sample code. This part is not taken into account for testing this method's efficiency.

```
x    'setenv SASORA V7';

data transctn;
   set viewlib.OraTblV;
                    /* To be updated */
   if id='AG1001' and entry='Entry1' and medcode='AN312' and
      period1='Postop' and period2='Day1' then type='U';
                    /* To be deleted */
   else if id='AG1001' and entry='Final' and medcode='HC527' and
      period1='Intraop' and period2='PostCPB' then type='D';
                    /* To be inserted */
   else if id='AG1002' and entry='Entry1' and medcode='PV946' and
      period1='Intraop' and period2='PreCPB' then type='I';
   if type in ('U','D','I');
                    /* For Final records only */
   entry='Final';
   run;
```

The *viewlib.OraTblV* view descriptor is then updated with data from the *transctn* data file via the following sample code.

```
x    'setenv SASORA V7';

data viewlib.OraTblV;
   modify viewlib.OraTblV transctn;
   by id entry medcode period1 period2;
   select (_iorc_);
                  /* No match in MASTER - Insert */
      when (%sysrc(_dsenmr)) do;
           if type='I' then output viewlib.OraTable;
           _error_=0;
      end;
                  /* Match located - Update or Delete */
      when (%sysrc(_sok)) do;
           if type='U' then replace viewlib.OraTable;
           else if type='D' then remove viewlib.OraTable;
      end;
                  /* Traps unexpected outcomes */
      otherwise do;
           put 'Unexpected Error condition: _IORC_ = ' _iorc_;
           put _all_;
           _error_=0;
      end;
   end;
   run;
```

This method does not take advantage of the index columns in ORACLE. Neither does it have any control of the data processing in Oracle. The BY statement specified in the DATA step causes the SAS/ACCESS interface view engine to automatically generate an ORACLE BY clause for the variables and pass it dynamically to ORACLE for processing before the DATA step uses the data. Sorting data in ORACLE can be resource-intensive and can adversely affect the performance.

8.0 Result

The four methods provided by the SAS/ACCESS software and discussed in this paper are listed below.

1. SQL procedure pass-through facility
2. DBLOAD procedure SQL statement
3. SQL Procedure with view descriptor
4. DATA step MODIFY statement with view descriptor

They enable you to perform ORACLE SQL Data Manipulation Language (DML) commands such as UPDATE, INSERT and DELETE, in ORACLE database without leaving the SAS session. In most cases, they can be used with equal efficiency. However, when the ORACLE table referenced is large, choosing an appropriate method to avoid unnecessary sorting can improve the performance dramatically.

Table 8.1 shows the performance of each method in terms of CPU and real time spent on performing the 2 sets of SQL operation on the 2 test tables. The numbers are intended to make relative comparison among these methods. They may vary with further database tune-up in ORACLE and different hardware and software set-up.

Table 4.1: Performance of ORACLE SQL DML Operations using SAS Software

Table size	Number of Transactions	Method	CPU mean ± std.	Real time (second) mean ± std.
600 KB	120	SQL Passthrough	0.13 ± 0.02	2.35 ± 0.70
		DBLOAD	0.22 ± 0.03	3.01 ± 1.06
		SQL View	5.33 ± 0.60	41.20 ± 2.68
		MODIFY View	2.48 ± 0.65	29.06 ± 1.06
	1200	SQL Passthrough	0.08 ± 0.03	20.65 ± 9.99
		DBLOAD	0.19 ± 0.03	17.39 ± 7.47
		SQL View	38.51 ± 2.31	336.29 ± 3.07
		MODIFY View	97.97 ± 0.52	201.79 ± 2.90
30MB	120	SQL Passthrough	0.20 ± 0.03	6.7 ± 2.46
		DBLOAD	0.26 ± 0.02	9.94 ± 7.25
		SQL View	0.77 ± 0.04	19.24 ± 11.44
		MODIFY View	1.05 ± 0.09	20.74 ± 0.62
	1200	SQL Passthrough	0.22 ± 0.05	29.55 ± 5.64
		DBLOAD	0.29 ± 0.05	33.07 ± 12.77
		SQL View	5.78 ± 0.48	184.58 ± 5.20
		MODIFY View	145.84 ± 1.66	301.01 ± 3.87

SQL procedure pass-through facility and the DBLOAD procedure SQL statement are more efficient and quicker that the other two methods, regardless of table sizes and the amount of data transactions. They modify ORACLE data by selecting records via WHERE clause to avoid sorting in ORACLE data. All the SQL commands are passed dynamically to ORACLE within one SAS/ACCESS connection to ORACLE database.

To the contrary, the SQL procedure with view descriptor builds the connection to ORACLE each time the view descriptor is referenced and therefore requires a series of SQL interface calls to carry out all the data transactions. The DATA step MODIFY statement of the view descriptor causes the ORACLE data to be sorted before the DATA step uses the data. In addition, they both lack the control of the data processing in ORACLE.

In conclusion, the SQL procedure pass-through facility is chosen over DBLOAD procedure SQL statement because it gives you complete control of the data processing with ORACLE non-query SQL commands such as COMMIT and SET TRANSACTION. These two commands are critical to the application described in this paper, because the amount of data transactions passed to ORACLE is dynamic. They minimize the chance of data loss caused by a DBMS failure or other errors during the processing.

The testing of the SAS application using the SQL procedure pass-through facility shows better performance than the Visual Basic module. It has been tested with test data in the 57 ORACLE tables for 7 patients, 70 patients and 154 patients. It takes approximately 10 minutes, 27 minutes and 53 minutes, respectively. In contrast, with the same test data it takes the Visual Basic module about 10 minutes, 80 minutes and 190 minutes, respectively.

9.0 Reference

Guo, Annie (1998), "Efficient Cross Database Data Transaction Processing Between SAS® Software and ORACLE® using SAS®," *Proceedings of the SAS Users Group International Conference*, SAS Institute., NC.

ORACLE7® Server SQL Language Reference Manual (1992), Oracle Corporation, CA.

SAS/ACCESS® Interface to ORACLE®, Usage and Reference, Version 6, Second Edition (1993), SAS Institute Inc., NC.

10.0 Acknowledgments

The author wishes to thank Ischemia Research and Education Foundation for providing funding and support for this paper to be presented at SUGI 23.

Thanks to Long Ngo for his support of this SAS application and the paper.

SAS and SAS/ACCESS are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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

Author

Annie Guo
Ischemia Research and Education Foundation
250 Executive Park Blvd. #3400, San Francisco, CA 94134
(415) 715-2300
asg@orion.iref.org