Efficient Cross Database Data Transaction Processing Between SAS® Software and ORACLE® using SAS®

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 convert the ORACLE tables to SAS tables, perform intensive data processing in SAS, 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 ORACLE tables.

The objective is to achieve speed and efficiency in dealing with the large ORACLE database using SAS software. This paper addresses how such a SAS application is designed and implemented. It automatically converts data from ORACLE to SAS including appropriate labels and formats. The data converted are then processed in such a way that only necessary data transactions will later be passed dynamically to ORACLE. In addition, SQL procedure pass-through facility provided by SAS/ACCESS® is used to manipulate the data in ORACLE, because it avoids unnecessary sorting in ORACLE data and thus works in an efficient fashion.

1.0 Introduction

In an international, multicenter epidemiological study of more than 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* record. After the *final* record is created, if the two original entries are updated in any of the 57 tables, the *final* record will be modified accordingly as soon as the update is completed and the 2 entries are compared equal.

2.0 Case Study

Seven sample patients, 1001 ~ 1007, and 2 sample tables, *table_1* and *table_2* in Table 2.1 and Table 2.2 are used to illustrate the scenario and construct sample codes throughout this paper. *Table_1* represents the first table out of the 57 tables. It contains the s*tatus* column indicating a patient's entire CRF data entry status. The value *pass* on *status* column means the 7 patients' *entry1* and *entry2* records in both tables are completed. The value *update* on *status* column indicates the *final* records of patients 1001 ~ 1004 and 1006 are pending for update in at least one table, since their 2 original entries have been modified after the *final* records were created.

A grouping of *id* and *entry* columns is the unique identifier among records to be used to compare data entries. The comparison between *entry1* and *entry1* records shows that patients 1001 ~ 1005 have no discrepancy found in *table_1* and *table_2*. Thus the *final* records of patients 1001 ~ 1004 will be updated based on their *entry1* records, and a copy of patient 1005's *entry1* record will be saved as *final* record. The comparison result also shows that patients 1006 and 1007 have different *entry1* and *entry1* records in *table_1* and *table_2*, respectively. Thus the discrepancy report for these 2 patients will be produced for data entry clerks to make corrections.

Table 2.1 : *Table_1* before Data Transaction Processing

Id	Entry	Name	Status	Var11	Var12	
1001	Entry1	PX	Pass	1	No	
1001	Entry2	PX	Pass	1	No	
1001	Final	PX	Update	1	No	
1002	Entry1	ASG	Pass	3	Unk	
1002	Entry2	ASG	Pass	3	Unk	
1002	Final	ASG	Update	***7***	Unk	← To be updated
1003	Entry1	RL	Pass	6	Yes	
1003	Entry2	RL	Pass	6	Yes	
1003	Final	***RN***	Update	6	Yes	← To be updated
1004	Entry1	RSA	Pass	2	Unk	
1004	Entry2	RSA	Pass	2	Unk	
1004	Final	***RSM***	Update	2	***Yes***	← To be updated
1005	Entry1	MP	Pass	3	Yes	← Final record to be created
1005	Entry2	MP	Pass	3	Yes	
1006	Entry1	HMS	Pass	8	***No***	← Discrepancy between Entry1 and Entry2
1006	Entry2	HMS	Pass	8	***Yes***	
1006	Final	HMS	Update	8	No	
1007	Entry1	AG	Pass	0	Yes	
1007	Entry2	AG	Pass	0	Yes	

Table 2.2 : *Table_2* before Data Transaction Processing

Id	Entry	Name	Var21	Var22	
1001	Entry1	PX	PostOp	No	
1001	Entry2	PX	PostOp	No	
1001	Final	PX	PostOp	***Yes***	← To be updated
1002	Entry1	ASG	PreOp	Unk	
1002	Entry2	ASG	PreOp	Unk	
1002	Final	ASG	PreOp	Unk	
1003	Entry1	RL	Pod1	Yes	
1003	Entry2	RL	Pod1	Yes	
1003	Final	***RN***	Pod5	***No***	← To be updated
1004	Entry1	RSA	Pod2	Unk	
1004	Entry2	RSA	Pod2	Unk	
1004	Final	***RSM***	Pod6	Unk	← To be updated via Table_1
1005	Entry1	MP	Admisn	Yes	← Final record to be created
1005	Entry2	MP	Admisn	Yes	
1006	Entry1	HMS	PostOp	Yes	
1006	Entry2	HMS	PostOp	Yes	
1006	Final	HMS	PostOp	Yes	
1007	Entry1	AG	***Pod1***	Yes	← Discrepancy between Entry1 and Entry2
1007	Entry2	AG	***PostOp***	Yes	

Table 2.3 and Table 2.4 show the results after the *final* records are modified. The values on *status* column have been reset automatically. The *final* records of patients 1001 ~ 1005 are in *pending* status, because they have been updated and are pending for the next step of data processing. The *entry1* and *entry1* records of patients 1006 and 1007 are in *fail* status, because the 2 entries differ in at least one table.

The first 3 columns, *id*, *entry* and *name* in *table_1*, compose 2 user-written ORACLE database triggers, UPDATE and INSERT. Because the 3 columns exist in all tables, the 2 triggers ensure that the ORACLE SQL UPDATE and INSERT commands performed on any of the 3 columns in the first table will be carried out implicitly on the rest tables. For example, in *table_1* the update on column *name* of patients 1003's and 1004's *final* records will cause the corresponding *final* records in *table_2* to be updated. Similarly, when patient 1005's *final* record is inserted into *table_1*, the corresponding *final* record is implicitly inserted into *table_2* along with the values on the first 3 columns inherited from *table_1*.

Table 2.3 : *Table_1* after Data Transaction Processing

Id	Entry	Name	Status	Var11	Var12	
1001	Entry1	PX	Pass	1	No	
1001	Entry2	PX	Pass	1	No	
1001	Final	PX	**Pending**	1	No	
1002	Entry1	ASG	Pass	3	Unk	
1002	Entry2	ASG	Pass	3	Unk	
1002	Final	ASG	**Pending**	***3***	Unk	← Updated
1003	Entry1	RL	Pass	6	Yes	
1003	Entry2	RL	Pass	6	Yes	
1003	Final	***RL***	**Pending**	6	Yes	← Updated
1004	Entry1	RSA	Pass	2	Unk	
1004	Entry2	RSA	Pass	2	Unk	
1004	Final	***RSA***	**Pending**	2	***Unk***	← Updated
1005	Entry1	MP	Pass	3	Yes	
1005	Entry1	MP	Pass	3	Yes	
1005	***Final***	***MP***	***Pending***	***3***	***Yes***	← Inserted
1006	Entry1	HMS	**Fail**	8	***No***	← Discrepancy
1006	Entry2	HMS	**Fail**	8	***Yes***	
1006	Final	HMS	Update	8	No	
1007	Entry1	AG	**Fail**	0	Yes	← Discrepancy in Table_2
1007	Entry2	AG	**Fail**	0	Yes	

Table 2.4 : *Table_2* after Data Transaction Processing

Id	Entry	Name	Var21	Var22	
1001	Entry1	PX	PostOp	No	
1001	Entry2	PX	PostOp	No	
1001	Final	PX	PostOp	***No***	← Updated
1002	Entry1	ASG	PreOp	Unk	
1002	Entry2	ASG	PreOp	Unk	
1002	Final	ASG	PreOp	Unk	
1003	Entry1	RL	Pod1	Yes	
1003	Entry2	RL	Pod1	Yes	
1003	Final	***RL***	Pod5	***Yes***	← Updated (Name via Table_1)
1004	Entry1	RSA	Pod2	Unk	
1004	Entry2	RSA	Pod2	Unk	
1004	Final	***RSA***	Pod6	Unk	← Updated via Table_1
1005	Entry1	MP	Admisn	Yes	
1005	Entry2	MP	Admisn	Yes	
1005	***Final***	***MP***	***Admisn***	***Yes***	← Inserted
1006	Entry1	HMS	PostOp	Yes	
1006	Entry2	HMS	PostOp	Yes	
1006	Final	HMS	PostOp	Yes	
1007	Entry1	AG	***Pod1***	Yes	← Discrepancy
1007	Entry2	AG	***PostOp***	Yes	

3.0 Problems

A Visual Basic module making use of ORACLE Views, Functions and PL/SQL Procedures has been developed to access the ORACLE data through ODBC, compare the values on the 7000 variables between the 2 entries, and modify the *final* records and reset data entry status in ORACLE tables accordingly. The performance of the Visual Basic module is not satisfactory, because 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 complete the job.

SAS software is brought up because of its optimal performance with data processing. For example, 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 will consist of three parts. First of all the 57 ORACLE tables are converted to SAS. Then the data converted are processed to determine what need to be modified, and the result is translated to ORACLE SQL commands. Lastly the ORACLE SQL commands are dynamically passed to ORACLE for data processing without leaving the SAS session.

4.0 Objective

The goal is to achieve speed and computational efficiency in dealing with such a large ORACLE database. In short the following objectives have been identified.

- Automatic generation of a data conversion program.

The development and testing of this application started before the Case Report Form (CRF) was finalized. Changes were constantly made on the form, and new variables were added into and invalid ones were dropped out of the 57 tables in ORACLE. Thus it is desired to have an automatically generated SAS program that always reflects the current database structure in ORACLE such as column names, labels, and data types.

- Efficient data conversion and data comparison.

The CRF data are split into 57 tables because of the tremendous amount of data collected. However, they have to be processed all at once to determine whether a patient's 2 entries are identical. Thus it is critical to avoid unnecessary sorting procedure and duplicate copy of data files.

- Avoid unnecessary data transactions passed to Oracle.

Update only those records that need to be updated. For example, patient 1001's data correction was made to *table_2* only, not *table_1*. Thus no update on the *final* record in *table_1* is needed.

In addition, the ORACLE UPDATE database trigger has to be handled with caution to avoid a series of redundant updates throughout the rest 56 tables. For example, in *table_1*, the values on the 3 trigger columns for patient 1002's *final* record do not need to be updated. Thus the 3 trigger columns should be excluded from the ORACLE UPDATE command, in order not to fire the ORACLE UPDATE database trigger.

- Commit data changes to ORACLE immediately.

This releases data redo logs in ORACLE database and makes room for the next data transaction. It minimizes the chance of data loss caused by a DBMS failure.

- Avoid complex SAS coding.

For example, for patient 1004's *final* record in *table_1*, instead of whether individual column needs update, all we want to know is whether the record needs update. If yes, the 6 columns will be modified all at once in one update operation without loosing much efficiency, because it is believed that an update operation in ORACLE can be carried out with almost equal efficiency despite the number of columns involved. In turn this may save lots of SAS coding in data processing.

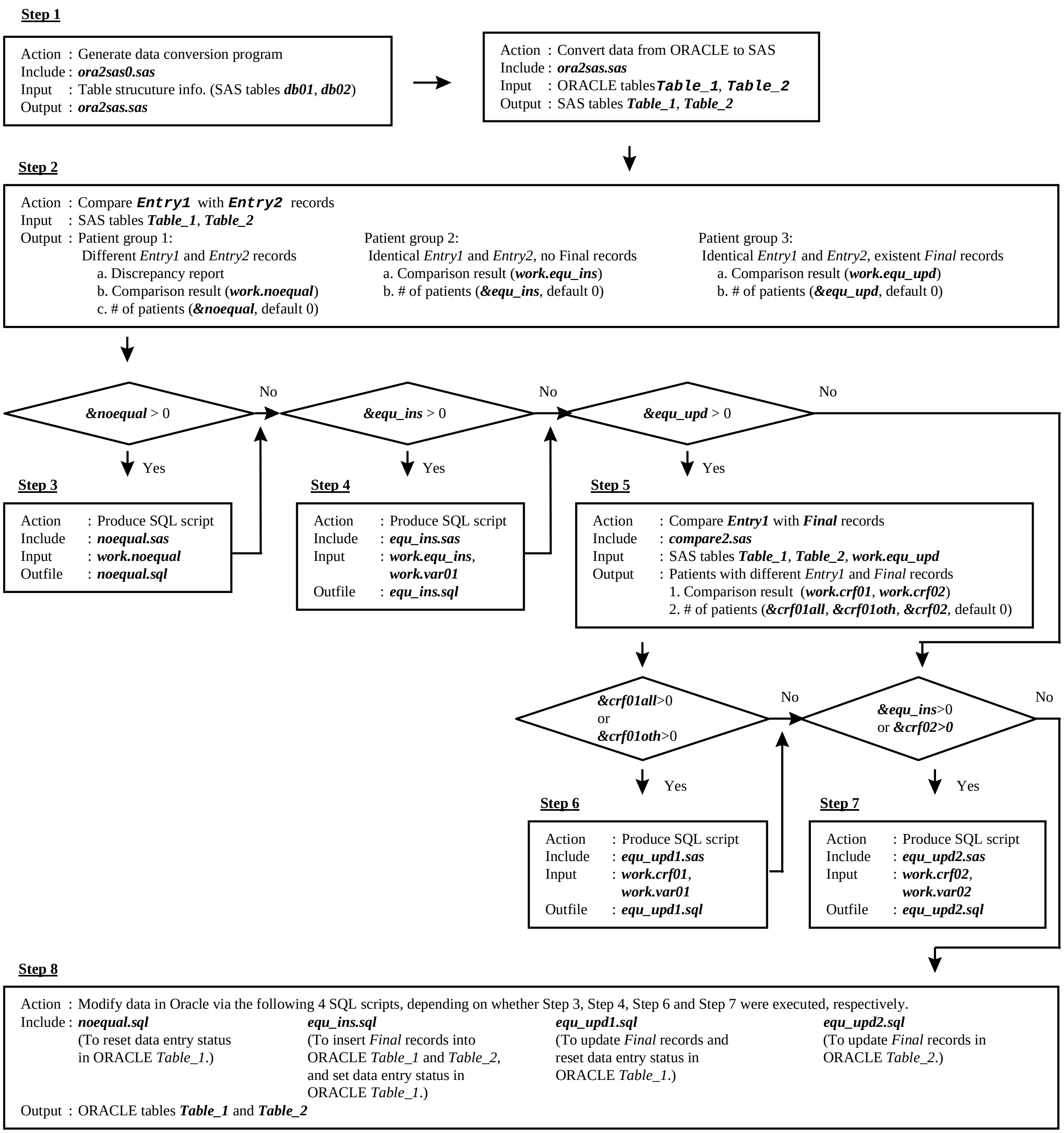
- An efficient method to dynamically pass SQL commands to ORACLE.

In most cases the methods provided by SAS/ACCESS can be used with equal efficiency. However, the performance may vary in dealing with large ORACLE tables.

5.0 Design

Figure 5.1 illustrates the SAS application design that satisfies the above objectives. The entire application is a SAS program named *compare.sas*. It makes use of SAS global macro variables to form *if-then* conditions and to determine whether a series of child programs, DATA steps and PROCs should be executed. For example, in Step 3 the program *noequal.sas* is included and executed only if the global macro variable *&noequal* from Step 2 is greater than 0.

Figure 5.1: Data Flow Diagram For The SAS Application (*compare.sas*)



The hardware setup to development the SAS application is a HP Pentium 200 MHz PC connected to a HP 9000/755 workstation server. The software installed includes ORACLE 7, SAS V6.12 and SAS/ACCESS, SQL*Net V2, and HP-UX 10.*.

6.0 Implementation

6.1 ORACLE-to-SAS Data Conversion using SAS/ACCESS

Step 1 in Figure 5.1 :

The purpose of this step is to generate a data conversion program that always reflects the current database structure in ORACLE, and then to convert the data from ORACLE to SAS. All the up-to-date database structure information such as table names, variable names and labels is stored in SAS tables. Table 6.1 shows an example of such SAS tables, *db01.ssd* for *table_1* and *db02.ssd* for *table_2*. These SAS tables are the source files of constructing tables in ORACLE. Note that the SAS syntax such as on length of variable names applies here to prevent confusion in case longer names get truncated during data conversion process.

Table 6.1: Table Structure Information

db01.ssd for Table_1

Table	Variable	Label	Format	Type
table_1	id	Patient ID		Char
table_1	entry	Record Version	$version.	Char
table_1	name	Patient Initial		Char
table_1	status	Record Status	$status.	Char
table_1	var11	Hospital stay		Num
table_1	var12	Smoking	$yesno.	Char

db02.ssd for Table_2

Table	Variable	Label	Format	Type
table_2	id	Patient ID		Char
table_2	entry	Record Version	$version.	Char
table_2	name	Patient Initial		Char
table_2	var21	Period	$period.	Char
table_2	var22	On medication	$yesno.	Char

Table 6.2 : Data Conversion Program *ora2sas.sas*

```
libname crf  '~/epi2/basessd';
libname vlib '~/epi2/oracle';
options fmtsearch=(crf);
x    'setenv SASORA V7';

title 'E2CRF01';

proc access dbms=oracle;
   /* Access Descriptive file */
   create vlib.Table_1.access;
      user=asg;
      orapw=XXXXX;
      table=epi2_owner.Table_1;
      path='@product';
      list all;
   /* View Descriptive file */
   create vlib.Table_1.view;
      select all;
      list view;
      subset where (id like '100%');
   run;

/* SAS table converted from Oracle */
data crf.Table_1 (label=
      'EPI2/CRF01:Enrollment');
   set vlib.Table_1;
   attrib
   ID       label  = 'Patient ID'
   ENTRY    format = $version.
            label  = 'Record Version'
   NAME     label  = 'Patient Initial'
   STATUS   format = $status.
            label  = 'Record Status'
   VAR11    label  = 'Hospital stay'
   VAR12    format = $yesno.
            label  = 'Smoking'
   run;
```

```
title 'E2CRF02';

proc access dbms=oracle;
   /* Access Descriptive file */
   create vlib.Table_2.access;
      user=asg;
      orapw=XXXXX;
      table=epi2_owner.Table_2;
      path='@product';
      list all;
   /* View Descriptive file */
   create vlib.Table_2.view;
      select all;
      list view;
      subset where (id like '100%');
   run;

/* SAS table converted from Oracle */
data crf.Table_2 (label=
      'EPI2/CRF02:Screening');
   set vlib.Table_2;
   attrib
   ID       label  = 'Patient ID'
   ENTRY    format = $version.
            Label  = 'Record Version'
   NAME     label  = 'Patient Initial'
   VAR21    format = $period.
            Label  = 'Period'
   VAR22    format = $yesno.
            Label  = 'On medication'
   run;
```

The template program *ora2sas0.sas* has been developed and is included to process *db01.ssd* and *db02.ssd* in Table 6.1 and to generate the data conversion program *ora2sas.sas* in Table 6.2. The o*ra2sas.sas* is updated every time *compare.sas* is run and thus always reflects the current structure in ORACLE. It is then executed to convert data from Oracle to SAS.

Access descriptive files and view descriptive files are deleted after SAS tables are created to save disk space. All the SAS tables converted are saved permanently for data processing later discussed in Section 6.2, in order to avoid inefficient, frequent data transfer between ORACLE and SAS

6.2 Data Comparison in SAS

Step 2 in Figure 5.1 :

After all the data are converted to SAS, we would like to find out whether the data entered by the 2 data entry clerks are identical throughout all the tables. This is achieved by running COMPARE procedure with OUTNOEQUAL and OUTALL options between *entry1* and *entry2* records for each table. In result, the 7 sample patients are classified into the following 3 groups. The comparison result is summarized in Table 6.3 and will be used to construct SQL scripts later in Step 3, Step 4, Step 6 and Step 7.

Group 1: Patients who have different *entry1* and *entry1* records in at least one table.

Patients 1006 and 1007 are in this group. The discrepancy report from COMPARE procedure is generated for data entry clerks to make correction. The number of patients in this group, 2, is set to the global macro variable *&noequal*, and the comparison result is saved in *work.noequal* SAS table.

Group 2 : Patients who have identical *entry1* and *entry1* records in all tables, and no *final* records ever created.

Patient 1005 is in this group. The number of patient in this group, 1, is set to the global macro variable *&equ_ins*, and the comparison result is saved in *work.equ_ins* SAS table.

Group 3 : Patients who have identical *entry1* and *entry1* records in all tables, and existent *final* records.

Patients 1001 ~ 1004 are in this group. The number of patients in this group, 4, is set to the global macro variable *&equ_upd*, and the comparison result is saved in *work.equ_upd* SAS table.

Table 6.3: Comparison Result between *Entry1* and *Entry1*

Group 1:
work.noequal

ID	Result
1006	Discrepancy
1007	Discrepancy

&noequal=2

Group 2:
work.equ_ins

ID	Result
1005	Identical

&equ_ins=1

Group 3:
work.equ_upd

ID	Result
1001	Identical
1002	Identical
1003	Identical
1004	Identical

&equ_upd=4

Step 3 in Figure 5.1 :

For those patients in Group 1, the only data modification required is to set the data entry status of their *entry1* and *entry2* records to *fail* in ORACLE *table_1*. Thus, given that there is at least one patient in Group1, i.e. *&noequal* > 0, the purpose of this step is to create a SQL script that can pass the desired operation to ORACLE. This is achieved by including the template program *noequal.sas* that processes *work.noequal* in Table 6.3 and generates the SQL script *noequal.sql* shown in Table 6.4. This output script consists of a SQL procedure

EXECUTE statement and will be included later in Step 8 to pass SQL commands to ORACLE via SQL procedure pass-through facility. Note that the COMMIT command saves data changes and releases redo logs in ORACLE.

Table 6.4: SQL Script *noequal.sql*

```
execute (update epi2_owner.Table_1
            set status='Fail'
            where id in (
                 '1006',                ←ID in work.noequal, Table 6.3
                 '1007') and entry in ('Entry1','Entry2')) by asg;
execute (commit) by asg;
```

Step 4 in Figure 5.1 :

For patient 1005 in Group2, there are 2 steps of data modification needed in ORACLE.

Part 1: Make a copy of patient 1005's *entry1* record in *table_1* and insert into *table_1* as *final* record.

The purpose of Step 4 is therefore to create a SQL script that will perform the Part 1 operation in ORACLE. This is achieved by including the template program *&equ_ins.sas* that processes *work.equ_ins* in Table 6.3 and *work.var01* in Table 6.5, and generates the *equ_ins.sql* SQL script shown in Table 6.6. Note that the *work.var01* SAS table is created via CONTENTS procedure of *table_1* SAS table, and helps construct the INSERT command in *equ_ins.sql* in 2 ways: (1) the values on *position* column retain the variable position as stored in ORACLE *table_1*, since the INSERT command requires the variables to be listed in the same order as they appear in ORACLE *table_1*, and (2) the character values on *varvalue* column, including the single quotes and commas, are to comply with ORACLE SQL syntax.

Table 6.5: Variable List for ORACLE Tables

work.var01 for ORACLE Table_1

VarName	Position	VarValue
ID	1	ID,
ENTRY	2	'Final',
NAME	3	NAME,
STATUS	4	'Pending',
VAR11	5	VAR11,
VAR12	6	VAR12

work.var02 for ORACLE Table_2

VarName	Position	VarValue
VAR21	4	VAR21,
VAR22	5	VAR22

Table 6.6: SQL Script *equ_ins.sql*

```
execute(insert into Table_1
           select ID,                    ←VarValue in work.var01, Table 6.5
                  'Final',
                  NAME,
                  'Pending',
                  VAR11,
                  VAR12 from Table_1
           where id in (                     ←ID in work.equ_ins, Table 6.3
                  '1005') and entry='Entry1') by asg;
%put &sqlxmsg;
execute (commit) by asg;
```

After *equ_ins.sql* is included and executed later in Step 8, the *entry1* and *final* records in *table_1* are identical except *entry* and *status* columns which are set to the values *final* and *pending*, respectively, as illustrated in Table 2.3. As soon as the *final* record is created in *table_1*, because of the ORACLE INSERT database trigger in *table_1*, the corresponding *final* record is inserted implicitly into *table_2* with the values on the 3 trigger columns, *id*, *entry* and *status*, inherited from *table_1*, and the values on the 2 non-trigger columns left blank.

Part 2: Based upon the *entry1* record in *table_2*, update the values on the 2 non-trigger columns of patient 1005's newly inserted *final* record in *table_2*. This part will be implemented later in Step 7.

Step 5 in Figure 5.1 :

For those patients in Group 3, i.e. *&equ_upd* > 0, instead of blindly updating all the existent *final* records with the values from the current *entry1* records for all the tables, we would like to find out in which tables the *entry1* and *entry1* records have been modified since the *final* records were updated, so that the *final* records need to be updated again. Furthermore, if the *final* records in *table_1* need to be updated, we want to know if any of the 3 trigger columns is involved. If yes, the update operation will be performed on all the 6 columns for *table_1* and the ORACLE UPDATE database trigger is fired. If no, the update operation will be only on the 3 non-trigger columns for *table_1* and the database trigger is not fired.

To achieve the objective, c*ompare2.sas* program is included to compare the *entry1* records with the *final* records using COMPARE procedure with OUTALL and OUTNOEQUAL options. The comparison result is saved in the 2 SAS tables, *work.crf01* and *work.crf02*, shown in Table 6.7. The 3 global macro variables, *&crf01all*, *&crf01oth* and *&crf02* are set to the numbers of patients who need update: (1) on at least one trigger columns in *table_1*, despite the non-trigger columns, (2) on at least one non-trigger columns in *table_1*, but not any of the 3 trigger columns, and (3) on at least one non-trigger column in *table_2*, but not any of the 3 trigger columns, respectively. Note that the 3 trigger columns are taken care of by *table_1* via the ORACLE UPDATE database trigger, so they are excluded from the COMPARE procedure of *table_2*.

Table 6.7 : Comparison Result between *Entry1* and *Final* Records for Patient Group 3

work.crf01 for Table_1

ID	Trigger columns	Other columns
1001	Identical	Identical
1002	Identical	Discrepancy
1003	Discrepancy	Identical
1004	Discrepancy	Discrepancy

&crf01all=2 (i.e. 1003,1004)
&crf01oth=1 (i.e. 1002)

work.crf02 for Table_2

ID	Other columns
1001	Discrepancy
1002	Identical
1003	Discrepancy
1004	Identical

&crf02=2 (i.e. 1001, 1003)

Step 6 in Figure 5.1 :

According to the comparison result in *work.crf01* in Table 6.7, three types of update operation will be performed in ORACLE *table_1*. First of all, the only data modification required for patient 1001's *final* record is to reset the data entry status to *pending*. Next, patient 1002's *final* record will be updated on non-trigger columns only. Lastly, for patient 1003 and 1004, the entire *final* records in *table_1* will be updated, so that the ORACLE UPDATE database trigger is fired and the values on the 3 trigger columns in *table_2* can be updated implicitly.

Table 6.8: SQL Script *equ_upd1.sql*

```
execute(update Table_1
   set STATUS = 'Pending'
   where id in (
        '1001') ← ID in work.crf01
   and status='Final') by asg;
%put &sqlxmsg;
execute(commit) by asg;

execute(update Table_1 a
   set ( STATUS,← VarName in work.var01
         VAR11,
         VAR12)
   = (select
         'Pending',
         VAR11, ← VarValue in work.var01
         VAR12
      from Table_1 b
      where entry='Entry1'
      and a.id=b.id)
   where id in (
         '1002' ← ID in work.crf01
   ) and status='Final') by asg;
%put &sqlxmsg;
execute(commit) by asg;
```

```
execute(update Table_1 a
   set ( ID, ← VarName in work.var01
         ENTRY,
         NAME,
         STATUS,
         VAR11,
         VAR12)
   = (select
         ID,← VarValue in work.var01
         'Final',
         NAME,
         'Pending'
         VAR11,
         VAR12
      from Table_1 b
      where a.id=b.id and
      entry='Entry1')
   where id in (
         '1003',  ← ID in work.crf01
         '1004'
   ) and status='Final') by asg;
%put &sqlxmsg;
execute(commit) by asg;
```

The purpose of Step 6 is to create a SQL script that can pass the above operation to ORACLE. This is achieved by including the template program *equ_upd1.sas* that processes *work.crf01* in Table 6.7 and *work.var01* in Table 6.5, and generates the SQL script *equ_upd1.sql* in Table 6.8. Note that the UPDATE command for patient 1002 excludes the 3 trigger columns.

Step 7 in Figure 5.1 :

According to the comparison result in *work.crf02* in Table 6.7, the values on the 2 non-trigger columns of patients 1001's and 1003's *final* records in *table_2* need update. In addition, to implement the Part 2 operation described earlier in Step 4, patient 1005's *final* record in *table_2* will be updated as well.

The purpose of Step 7 is to create a SQL script that will pass the above operations to ORACLE. This is achieved by including the template program *equ_upd2.sas* that processes *work.equ_ins* in Table 6.3, *work.var02* in Table 6.5, and *work.crf02* in Table 6.7. Note that *work.var02* is created via CONTENTS procedure of table_2, similar to *work.var01* described earlier in Step 4.

Table 6.9: SQL Script *equ_upd2.sql*

```
execute (update Table_2 a
           set ( VAR21,                              ←VarName in work.var02
                 VAR22)
           = (select
                 VAR21,                              ←VarValue in work.var02
                 VAR22 from Table_2 b
                 where a.id=b.id and status='Final')
           where id in (
                 '1001',                 ←ID in work.crf02 and work.equ_ins
                 '1003',
                 '1005') and status='Final') by asg;
%put &sqlxmsg;
execute (commit) by asg;
```

6.3 Data Manipulation in ORACLE using SQL Procedure Pass-Through Facility

Step 8 in Figure 5.1 :

The 4 SQL scripts, *noequal.sql*, *equ_ins.sql*, *equ_upd1.sql* and *equ_upd2.sql*, created in Step 3, Step 4, Step 6 and Step 7, respectively, can modify the ORACLE data in Table 2.1 and Table 2.2 and the resulting data will be as shown in Table 2.3 and Table 2.4. The purpose of Step 8 is to dynamically pass the 4 SQL scripts to ORACLE for data processing without leaving the current SAS session, as long as these scripts were just updated in the current *compare.sas* session. The sample code in Table 6.10 shows how to achieve the objective.

Table 6.10: SQL Procedure EXECUTE Statement

```
%macro updsql;
proc sql;
   connect to oracle as asg
      (user=asg orapw=XXXXX path='@product');
   %if &noequal > 0 %then %do;                       ← Step 3
      %include 'noequal.sql'/source2;
      %end;
   %if &equ_ins > 0 %then %do;                       ← Step 4
      %include 'equ_ins.sql'/source2;
      %end;
   %if &crf01all > 0 or &crf01oth > 0 %then %do;     ← Step 6
      %include 'equ_upd1.sql'/source2;
      %end;
   &if &equ_ins > 0 or &crf02 > 0 %then %do;         ← Step 7
      %include 'equ_upd.sql'/source2;
      %end;
   disconnect from asg;
   quit;
%mend updsql;
%updsql;
```

SQL procedure pass-through facility provided by SAS/ACCESS software is selected to accomplish the task, because it works efficiently with large ORACLE tables in 4 ways: (1) avoiding unnecessary sorting procedure in ORACLE data, (2) providing complete control of the data transaction in ORACLE, (3) making use of the ORACLE optimizer such as indexes, and (4) passing all SQL commands within one connection to ORACLE.

7.0 Conclusion

The testing of the SAS application 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.

8.0 Reference

Guo, Annie (1998), "An Investigation of the Efficiency of SQL DML Operations Performed on an ORACLE® DBMS using SAS/ACCESS® Software," *Proceedings of the SAS Users Group International Conference*, SAS Institute Inc., 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.

9.0 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.

Author

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