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 status 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

<table>
<thead>
<tr>
<th>Id</th>
<th>Entry</th>
<th>Name</th>
<th>Var21</th>
<th>Var22</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Entry1</td>
<td>PX</td>
<td>Pass</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>Entry2</td>
<td>PX</td>
<td>Pass</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>Final</td>
<td>PX</td>
<td>Update</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Entry1</td>
<td>AG</td>
<td>Pass</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Entry2</td>
<td>AG</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Final</td>
<td>AG</td>
<td>Update</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Entry1</td>
<td>RL</td>
<td>Pass</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Entry2</td>
<td>RL</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Final</td>
<td>RL</td>
<td>Update</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Entry1</td>
<td>SA</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Entry2</td>
<td>SA</td>
<td>Pass</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Final</td>
<td>SA</td>
<td>Update</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Entry1</td>
<td>MP</td>
<td>Pass</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Entry2</td>
<td>MP</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Final</td>
<td>MP</td>
<td>Update</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Entry1</td>
<td>HMS</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Entry2</td>
<td>HMS</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Final</td>
<td>HMS</td>
<td>Update</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>Entry1</td>
<td>AG</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>Entry2</td>
<td>AG</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 : Table 2 before Data Transaction Processing

<table>
<thead>
<tr>
<th>Id</th>
<th>Entry</th>
<th>Name</th>
<th>Var21</th>
<th>Var22</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Entry1</td>
<td>PX</td>
<td>Pass</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>Entry2</td>
<td>PX</td>
<td>Pass</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>Final</td>
<td>PX</td>
<td>Update</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Entry1</td>
<td>AG</td>
<td>Pass</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Entry2</td>
<td>AG</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Final</td>
<td>AG</td>
<td>Update</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Entry1</td>
<td>RL</td>
<td>Pass</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Entry2</td>
<td>RL</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Final</td>
<td>RL</td>
<td>Update</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Entry1</td>
<td>SA</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Entry2</td>
<td>SA</td>
<td>Pass</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Final</td>
<td>SA</td>
<td>Update</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Entry1</td>
<td>MP</td>
<td>Pass</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Entry2</td>
<td>MP</td>
<td>Pass</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Final</td>
<td>MP</td>
<td>Update</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Entry1</td>
<td>HMS</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Entry2</td>
<td>HMS</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Final</td>
<td>HMS</td>
<td>Update</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>Entry1</td>
<td>AG</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>Entry2</td>
<td>AG</td>
<td>Pass</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

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.
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 procedure and duplicate copy of data files.

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)

Step 1

Action: Generate data conversion program  
Include: ora2sas.sas  
Input: Table structure info. (SAS tables db01, db02)  
Output: ora2sas.sas

Action: Convert data from ORACLE to SAS  
Include: ora2sas.sas  
Input: ORACLE tables Table_1, Table_2  
Output: SAS tables Table_1, Table_2

Step 2

Action: Compare Entry1 with Entry2 records  
Input: SAS tables Table_1, Table_2  
Output: Patient group 1: Patient group 2: Patient group 3:  
- Different Entry1 and Entry2 records  
  a. Discrepancy report (work.noequal)  
  b. Comparison result  
    (work.noequal)  
  c. # of patients (&noequal, default 0)

Patient group 2:  
- Identical Entry1 and Entry2, no Final records  
  a. Comparison result (work.equ_ins)  
  b. # of patients (&equ_ins, default 0)

Patient group 3:  
- Identical Entry1 and Entry2, existent Final records  
  a. Comparison result (work.equ_upd)  
  b. # of patients (&equ_upd, default 0)

&noequal > 0  
No

&equ_ins > 0  
No

&equ_upd > 0  
No

Step 3

Action: Produce SQL script  
Include: noequal.sas  
Input: work.noequal  
Outfile: noequal.sql

Step 4

Action: Produce SQL script  
Include: equ_ins.sas  
Input: work.equ_ins, work.var01  
Outfile: equ_ins.sql

&equ_ins > 0  
No

Step 5

Action: Produce SQL script  
Include: compare2.sas  
Input: SAS tables Table_1, Table_2, work.equ_upd  
Output: Patients with different Entry1 and Final records  
1. Comparison result (work.crfr01, work.crfr02)  
   2. # of patients (&crfr01all, &crfr01oth, &crfr02, default 0)

&crfr01all > 0  
No  
&crfr01oth > 0

&crfr02 > 0  
No

Step 6

Action: Produce SQL script  
Include: equ_upd1.sas  
Input: work.crfr01, work.var01  
Outfile: equ_upd1.sql

Step 7

Action: Produce SQL script  
Include: equ_upd2.sas  
Input: work.crfr02, work.var02  
Outfile: equ_upd2.sql

&crfr01all > 0  
No  
&crfr01oth > 0

&crfr02 > 0  
No

Step 8

Action: Modify data in Oracle via the following 4 SQL scripts, depending on whether Step 3, Step 4, Step 6 and Step 7 were executed, respectively.  
Include:  
- noequal.sql  
  (To reset data entry status in ORACLE Table_1.)  
- equ_ins.sql  
  (To insert Final records into ORACLE Table_1 and Table_2, and set data entry status in ORACLE Table_1.)  
- equ_upd1.sql  
  (To update Final records and reset data entry status in ORACLE Table_1.)  
- equ_upd2.sql  
  (To update Final records in ORACLE Table_2.)

Output: ORACLE tables Table_1 and Table_2
The hardware setup to develop 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

<table>
<thead>
<tr>
<th>Table_1</th>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Patient ID</td>
<td>Char</td>
</tr>
<tr>
<td>entry</td>
<td>Record Version</td>
<td>Char</td>
</tr>
<tr>
<td>name</td>
<td>Patient Initial</td>
<td>Char</td>
</tr>
<tr>
<td>status</td>
<td>Record Status</td>
<td>Char</td>
</tr>
<tr>
<td>var11</td>
<td>Hospital stay</td>
<td>Num</td>
</tr>
<tr>
<td>var12</td>
<td>Smoking</td>
<td>Char</td>
</tr>
</tbody>
</table>

Table 6.2: Data Conversion Program o2sas.sas

```
library crf '<-eqip/oracle';
library vlb '<-eqip/oracle';
options fastmem=crf;
x 'setenv SAMORA V7';
title 'EOCRF01';
title 'EOCRF02';
proc access db=oracle;
proc access db=oracle;
create vlb.TABLE_1;access;
create vlb.TABLE_1;access;
user=&vag; user=&vag;
orange=XXXXX; orange=XXXXX;
table=RP_TABLE_1;table=RP_TABLE_1;
path='PRODUCT';path='PRODUCT';
list all; list all;
/* View Descriptive file */ /* View Descriptive file */
create vlb.TABLE_1;view;
create vlb.TABLE_1;view;
select all; select all;
list view; list view;
```

The template program o2sas0.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 o2sas.sas in Table 6.2. The o2sas.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 entry2 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 entry2 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 &equiv_ins, and the comparison result is saved in work.equ_ins SAS table.

Group 3: Patients who have identical entry1 and entry2 records in all tables, and exist final records.

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

Table 6.3: Comparison Result between Entry1 and Entry2

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>work.noequal</td>
<td>work.equ_ins</td>
<td>work.equ_upd</td>
</tr>
<tr>
<td>ID Result</td>
<td>ID Result</td>
<td>ID Result</td>
</tr>
<tr>
<td>1006 Identical</td>
<td>1007 Identical</td>
<td>1001 Identical</td>
</tr>
<tr>
<td>2 &amp;noequal</td>
<td>4 &amp;equiv_ins</td>
<td>4 &amp;equiv_upd</td>
</tr>
</tbody>
</table>

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 Group 1, 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 table_1 using work.noequal
    set status='Fail'
where id in (1005)
    and varvalue in ('Entry1','Entry2') 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 &eq_var01 that processes work.eq_ins in Table 6.3 and work.var01 in Table 6.5, and generates the eq_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 eq_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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Position</th>
<th>Variable</th>
<th>Position</th>
<th>Variable</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>1</td>
<td>NAME</td>
<td>3</td>
<td>STATUS</td>
<td>4</td>
</tr>
<tr>
<td>ENTRY</td>
<td>2</td>
<td>VAR11</td>
<td>5</td>
<td>VAR12</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 6.6: SQL Script eq_ins_sql
```
execute (update var01 using table_1
    set status='Pending'
where id in (1005)
    and varvalue in ('Entry2') by asg);
```

After eq_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. &eq_upd > 0, instead of blindly updating all the existing 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, compare2.sas program is included to compare the entry1 records with the final records using COMPARE procedure with OUTALL and OUTNONEQUAL 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 column in table_1, despite the non-trigger columns, (2) on at least one non-trigger column 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
```
<table>
<thead>
<tr>
<th>Patient</th>
<th>Trigger status</th>
<th>Other columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Identical</td>
<td>Identical</td>
</tr>
<tr>
<td>1002</td>
<td>Identical</td>
<td>Discrepancy</td>
</tr>
<tr>
<td>1003</td>
<td>Identical</td>
<td>Identical</td>
</tr>
<tr>
<td>1004</td>
<td>Identical</td>
<td>Identical</td>
</tr>
</tbody>
</table>
```

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_upd.sql
```
execute (update table_1 using table_1
    set (ID, var01) = (1005, 'Final')
where id in (1005)
    and status='Pending';
```

Applications Development
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.equi_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 (commit) by asg;
%put &sqlxmsg;
%include 'equ_upd1.sql'/source2;
%if &crf02all > 0 or &crf02oth > 0 %then %do;
%include 'equ_ins.sql'/source2;
%if &equ_ins > 0 %then %do;
%include 'noequal.sql'/source2;
%end;
%include 'compare.sas'/source2;
quit;
%end;
%include 'equ_upd2.sql'/source2;
%if &noequal > 0 %then %do;
%end;
connect to oracle as asg
proc sql;
%macro updsql;
%include 'equ_upd.sql'/source2;
%end;
```

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 SAS session, as long as these scripts were just updated in the current SAS session. The sample code in Table 6.10 shows how to achieve the objective.

Table 6.10: SQL Procedure EXECUTE Statement

```
proc sql;
connect to oracle as asg
   (user=asg orapwd=XXXXX path='@product');
%if &equ_ins > 0 or &crf02 > 0 %then %do;
%include 'equ_ins.sql'/source2;
%if &equ_ins > 0 %then %do;
%include 'noequal.sql'/source2;
%end;
%include 'compare.sas'/source2;
quit;
%end;
```

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**


**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.

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