Overview of Efficient OC/RDC Validation/Derivation Procedure Development and Validation

Sunil G. Singh/Tomomi Saito
DBMS Consulting/RadPharm
12 October 2008
Tutorial Presentations
Session T04

Acknowledgements

- Many thanks to the OCUG for this opportunity to present this tutorial.
- Many thanks to the OCUG Planning and Review Committee and Chairs for their infinite patience in receiving and expeditious review of this presentation
- Many thanks to everyone who participated in the development of presentation.

Assumptions/Scope/Disclaimer

- Assumption: Audience has a basic understanding of creating and using Validation and Derivation Procedures in Oracle Clinical
- Scope: OC 4.5.x/RDC 4.5.x
- Disclaimer: Nothing in this presentation should be construed as a replacement for internal compliance audit and 21 CFR Part 11 internal compliance audit processes. Each user of Oracle Clinical and RDC must internally decide what levels of validation are sufficient for procedure development

RadPharm
The Imaging Core Lab

Agenda

- Overview of the Validation/Derivation Procedure development cycle
  - General Development Steps
  - Procedure Building in OC
- Testing the Procedures
  - Overall Validation Strategy for Procedures
  - Spreadsheet Model for capturing test results
- Use of a Procedure Library to reduce validation cycles
- Developing efficient procedures from a performance perspective

Part I: Efficiencies in Procedure Validation

Procedure Definition Process in OC

- Define the Procedure in Definition => Validation Procedures => Procedures or Definition
- Assign Question Groups with the Question Group Button
- For Each Question Group, assign Questions. Modify Question attributes as required
- Define Details for the Procedure with the Details button
- For Each Detail, select Variables. Modify Variable properties as required.
- Create User Variables
- Create Custom Code

Procedure Definition Process in OC (2)

- Special => Generate Procedure to compile the Procedure
- Perform Test Data entry to test the Procedure
- Special => Execute in Test
  - Run in Debug Mode
  - db_env_setting:_DEFAULT_:RXC_DEBUG_BUFFER_SIZE:1000000
- Action => Batch Jobs to view status of execution and log file.
- Document Test results and Logs. Check Test Discrepancies to see if they are created as expected.
- Correct errors/change logic and re-execute in test mode
- Change Status to 'A' for production data and production Batch Validation visibility.

Batch Validation and Execution of Validation Procedures

- During OC => Conduct => Data Validation Batch Validation Session, all Active Validation and Derivation Procedures are Executed for any Patient Positions with changed data
- Batch Validation also creates Univariate Discrepancies for:
  - Indicator Questions
  - Changes to DVGs and DVG subset mappings in a study where these new values would now generate a discrepancy
  - Changes to DCM Questions properties where these changes would also cause a discrepancy, such as high and low ranges for a question.
- At the end of each Batch Validation log file, there is a summary line stating how many discrepancies
  - were now created
  - were examined but left in their current status
  - were obsoleted, or no longer valid discrepancies

Batch Validation and Creation of Discrepancies

- Examples of how Batch Validation checks and creates various types of discrepancies
- Note that Univariate and DVG changes execute on all patients, not only modified patients

```
Connecting... Connected.
Batch Start: 11-APR-2006 08:27:21
Batch Validation at OCDEV for Study ORAQA owned by OCDEV
Number of patients with modified data: 1

DVG, DVG Subset, Thesaurus Resolution
0 DVG discrepancies processed, 0 became obsol
0 DVG Subset discrepancies processed, 0 became obsolete
0 Thesaurus Terms discrepancies processed, 0 became obsolete
0 Data Type discrepancies processed, 0 became obsolete
0 Length discrepancies processed, 0 became obsolete

Modified Univariate Validation Resolution
DCM DEMOG, Question SEX: 8 new, 0 remain curre
DCM DEMOG, Question WEIGHT: 4 new, 1 remain current, 0 became obsolete
No Thesaurus Management options installed.

Derivation Procedure Execution
Validation Procedure Execution
Procedure CHECKWT(1), version 1 start t:
Procedure CHECKWT: 0 new, 0 remain current, 0 became obsolete
Completed procedure CHECKWT at 11-APR-2006 08:27:22
Procedure CHK PE(101), version 0 start time is 11-APR-2006 08:27:22
Procedure CHK PE: 0 new, 0 remain current, 0 became obsolete
Completed procedure CHK_PE at 11-APR-2006 08:27:22

Indicator Question Checks for Patients with modified data
0 new, 0 remain current, 0 became obsolete
No Thesaurus Management options installed.

Batch Validation completed successfully at 11-APR-2006 08:27
```

Checks for DVG Changes

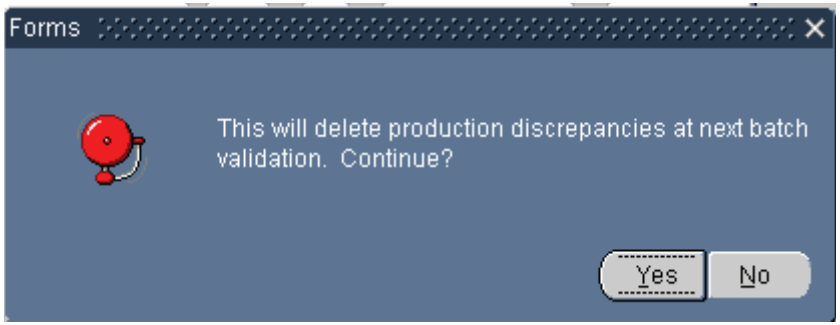
Checks for DCM Question Changes

Runs Procedures and Identifies new and existing discrepancies

Checks for Indicator Question Discrepancies

Versioning Validation Procedures to Prevent Deletion of Valid Discrepancies

- If a Validation Procedure is Active and there already exists Production mode Data Entry, the Validation Procedure should **never** be reset to a status of "P" for Provisional if changes need to made to this Procedure.
- OC correctly provides the following warning if this operation is attempted: "This will delete production discrepancies at next batch validation. Continue?" (see below, left)
- Instead, Procedure Versioning should be used, with the Special => Create New Version option (see below, right)
- This will create a new Provisional Mode Procedure which can be tested first and then changed to Status "A" for Active.
- Changing this new version of the Procedure will then automatically set the status of the existing Procedure to "R" for Retired, but the previously created discrepancies will remain.



RDC and Procedure Execution Status

- ON-LINE/DCM:
  - Run when a page containing ALL of the responses involved in the discrepancy is Saved Complete
  - Run during Validation of a Patient's or Site's Data from RDC
  - Run during Batch Validation
- ON-LINE:
  - Run during Validation of a Patient's or Site's Data from RDC
  - Run during Batch Validation
- OFF-LINE
  - Run during Batch Validation
- Changing all Procedures to ON-LINE/DCM is NOT necessarily good for site performance!

General Approach to Validated Procedure Development

- Protocol document contains explicit complex rules which can be translated to procedures ("explicit procedures")
- CRF design will yield cases where 2 or more data values should be compared ("implicit procedures")
- These requirements should be documented in a master testing spreadsheet
- Each rule should be explained in a clear description with the conditions for testing each rule
- For each rule, both a positive and negative test scenario should be defined
  - Applies for both Validation and Derivation procedures
- These rules should then be mapped to the DCM Question Groups from the CRFs

General Approach to Validated Procedure Development (2)

- In the master testing spreadsheet, the DCM Question Variables, User Variables and details should be identified
- For each detail, the test data values for the positive and negative test result should be documented
- For each Test Mode execution in Debug Mode, both the Log File and the Output File should be referenced
- The failing Discrepancy IDs should also be listed
- The Log and Output File should be retained in the study archive documentation and can be made part of the Data Management protocol

General Approach to Validated Procedure Development (3)

- Procedures requiring TMS or Lab Range Dependent values cannot be tested in Test Mode and MUST be executed in Production Debug Mode
- In this situation, a "placeholder" patient must be used, with the corresponding Log and Output files, and Discrepancy IDs documented in the master testing spreadsheet
- These Log and Output files need to be archived similar to the Test mode files, even if the "placeholder" patient is deleted using Delete Study Information (hard-delete) functionality

General Approach to Validated Procedure Development (4)

Derivation Procedures									
			Derived			Supporting			
Spec #	Procedure Name	Detail #	DCM	Question Group	Question Name	DCM	Question Group	Question Name	Description of Derivation
15906	D_NEW_LSN	1	NEW LESIONS	LESIONS	LSN_DSCRP_D	NEW LESIONS	LESIONS	LSN_CODE	The entered site code will derive the description from the supplied Site Code
15907	D_NEW_LSN	2	NEW LESIONS	LESIONS	LSN_AREA_D	NEW LESIONS	LESIONS	LSN_LENGTH	LSN_AREA_D = LSN_LENGTH x LSN_WIDTH
						NEW LESIONS	LESIONS	LSN_WIDTH	
15908	D_NEW_LSN	3	NEW LESIONS	LESIONS	RSP_YN_D	NEW LESIONS	LESIONS	LSN_LENGTH	IF LSN_LENGTH IS NOT NULL THEN RSP_YN_D = 'Y' ELSE RSP_YN_D = 'N'

		Details				Test Data		
DCM Question Variable	User Variable	Type	Description	Desc Question	Expression	Positive	Negative	Failing Disc ID
LSN_LENGTH, LSN_WIDTH, LSN_AREA_D, LSN_CODE, LSN_DSCRP_D, LSN_ID, RSP_YN_D, LSN_YN, LSN_RSP	N_AREA, V_DESCRIPTION, V_RSP_YN	CALCULATION	New Lesion Site Code	A.LSN_DSCRP_D	UPPER(V_DESCRIPTION)			
		CALCULATION	New Lesion Area Derivation	A.LSN_AREA_D	N_AREA			
		CALCULATION	New Lesion RSP_YN Derivation	A.RSP_YN_D	V_RSP_YN			

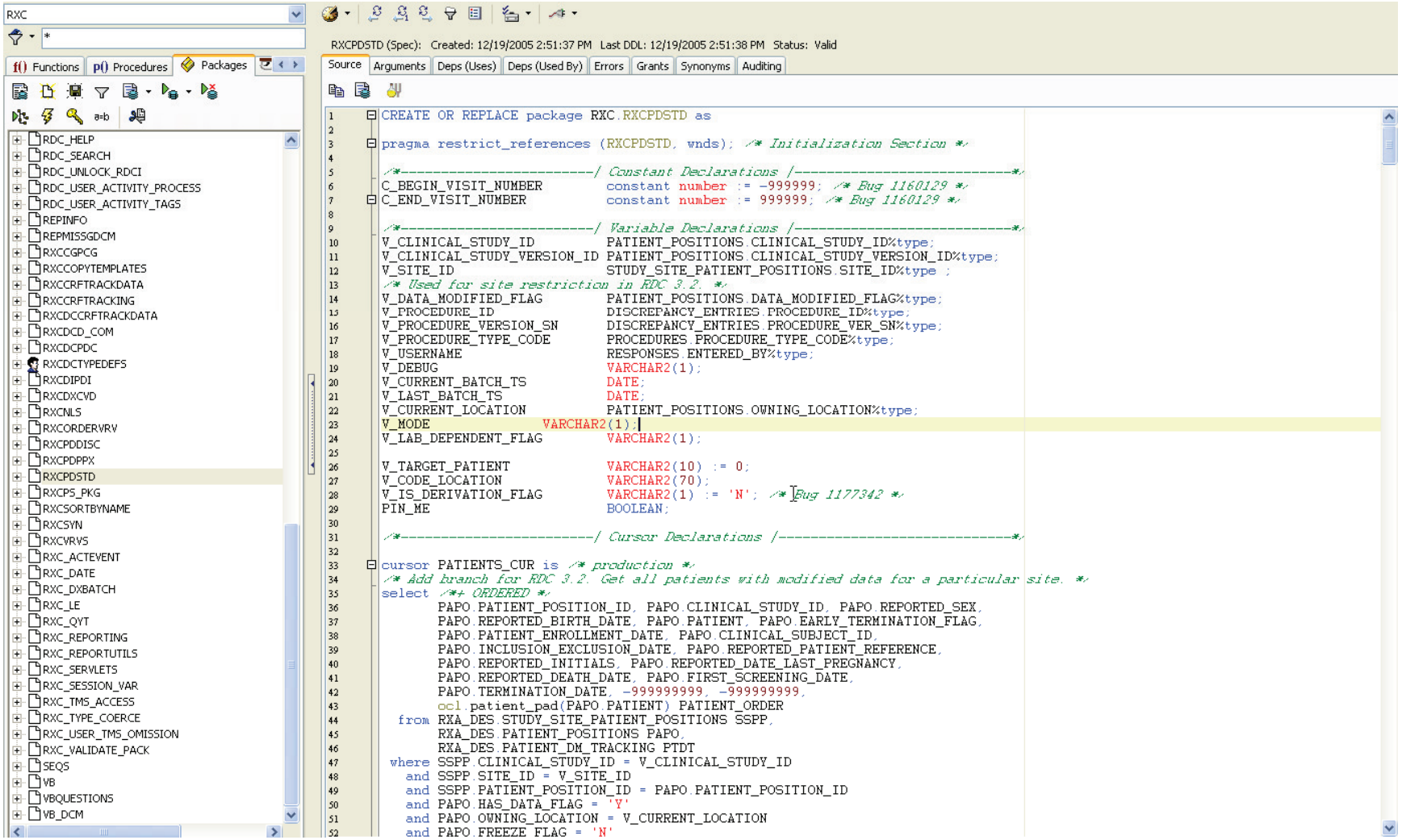
Standard OC Packages which are used in Procedures

- RXC_DATE to convert internally stored OC Swedish Date (YYYYMMDD) and Time (HH24MI) formats to Oracle RDBMS date and time, including partial date handling
- RXC_PAT_SYNCH to populate PATIENT_POSITIONS table. An example of how to create a validation procedure to populate Patient Enrollment (DOB, Gender, Initials) is provided in the "Create a Study Guide" pp 15-52 to 15-58 in the OC 4.5.1 version manual
- DCAPI packages which can be used to force DCI Book Switching in RDC
- RXCPDSTD package which is the core of all Validation and Derivation Procedures

Using Procedure Libraries: Reduce Validation Time for Procedures

- If both the Oracle RDBMS and OC use standard PL/SQL Procedure and function libraries as part of the validated OC application, this can also be done by users across multiple studies or in the GLIB
- Important to use an SDLC methodology for developing these functions, must be change controlled and in a separate schema (NOT the RXC schema)
- Functions and Packages MUST be parameterized. They must NOT contain any study-specific hard-coded values
- Documentation Library, proper comments in Package Headers, and Standard nomenclature are required
- Validate once, promote to GLIB, and reuse without retesting at the study level.

Using Procedure Libraries: Reduce Validation Time for Procedures (2)



Part II: Efficiencies in Procedures Design and Performance

Procedure Definition Mapping to Internal OC Tables

- Define the Procedure
  - Definition => Validation Procedures => Procedures
  - Definition => Derivation Procedures => Procedures
- Main form corresponds to RXC.PROCEDURES table. Every column on this form is stored in this table.
- It is possible to develop a Procedure Mapping for each fields' use and corresponding database table columns.

General Internal Structure of the Validation Procedure

- Each Procedure must have at least one Question Group, one Question, one Detail and one variable associated with it.
- Additionally, the package RXCPDSTD already defines the patient cursor.
- So each procedure starts with a nested set of loops, one for the patient cursor and one for the Question Groups
- Batch Validation is a patient driven process. So there is no way to have a procedure that does not execute without a loop.

General Internal Structure of the Procedure (2)

- When a value is retrieved by a DCM Question Group cursor, it must do a group by on the received_dcms table joining to responses to obtain a value. This is because it tries to always have unique values from the joining of Responses and Received_DCMs.
- Therefore, it already performs a sort via the group by.
- Limiting number of fetches is the key to improving procedure performance.

Procedure Performance Considerations (PPC)

- From the Oracle Clinical 4.5.1 On-Line Help for Procedures and Performance Issues

Performance issues

The choices you make in this window have important implications for the performance of the Procedures you create. This section gives tips on how to design your Procedure for maximum performance. For more detailed information see "How Procedures work internally" .

Each cursor, or Question Group alias, is nested inside those declared before it. So for each A record fetched, the B cursor must be opened and closed. To minimize need for cursors and maximize performance, use these design strategies:

- Use correlation whenever possible (See "Correlation").

Note
You can correlate a question group only to question groups listed above it in the Question Groups window, so order question groups accordingly.

- Use qualifying expressions and Where clause extensions whenever possible (See "Qualifying Expression" and"Where Clause Extension").

- If you need two types of aggregate functions — for example, sum and count — from the same Question Group, use only one cursor. You can specify as many aggregates as you need using one alias for all the Questions in that DCM Question Group.

- Use Event Range and First/Last Event Only whenever possible.

- Bear in mind the fact that Oracle Clinical automatically processes aggregate Question Groups first, followed by the primary reference Question Group. Then the DCM cursor fetches remaining Question Groups in the order they appear on the screen, which is determined by the order in which you define them.

PPC Background

- For each and every validation or derivation procedure which is created in Oracle Clinical, a bona-fide Oracle RDBMS PL/SQL package is created of the form RXC_PD.RXCPD_<Procedure_ID>_<Procedure_Version_Num>
- Each PL/SQL package consists of a minimum of two sets of cursors, or queries with conditions, one cursor is required to fetch the patients which will be processed by the procedure, and the second cursor is the data from the DCM question group.

PPC Background (2)

- These cursors are "nested" cursors, which means that the external set of data from the first is processed, and then the DCM Question Group cursor is processed for each loop of the external cursor. The Detail lines are the inner most IF...THEN conditions in the center of the nested cursors:

--> Outside Cursor Loop: Patients from Patient Positions (x Patients)
--> Inside Cursor Loop: Data from a DCM Question Group (y DCM Question Responses)
--> Detail Line 1 [IF ... THEN]
.
.
.
--> Detail Line n [IF ... THEN]
--> End Cursor Loop: Data from DCM Question Group
--> End Cursor Loop: Patients from Patient Positions

In this situation, the number of loop executions is (x Patients) times (y DCM Question Response).

PPC: Cost of Cursor Opening, Reduction of Loop Executions

- The opening of the cursor and loading of a separate package is the most expensive operation for the Oracle RDBMS. This process requires the most memory and Parse/Fetch/Execution cycles within the Oracle Shared Pool (SHARED_POOL) part of the System Global Area (SGA). The population of the cursor itself requires use of the Database Block Buffers (DB_BLOCK_BUFFERS). From a pure PL/SQL and RDBMS perspective, it is more efficient to reduce the number of times these cursors are opened.
- Therefore, the goal of writing efficient validation procedures is to REDUCE the number of LOOP EXECUTIONS.

PPC: Impacts of Loading Separate Procedures

- Furthermore, the execution of each PL/SQL package itself when they are separated means that the RDBMS must use more Shared Pool to execute each procedure individually.
- Considering that the size of the smallest of a validation procedure with a single detail line is 400 lines of generated code, making multiple separate procedures means that the Shared Pool and the RDBMS PL/SQL engine must execute a disproportionate amount of code for each separated procedure.
- Combining multiple detail lines introduces only 8-10 lines of code for each additional detail line, and the PL/SQL package itself is loaded only once for this type of procedure.

PPC: Reducing PL/SQL Package Loading Costs

- Once the RDBMS has to go through the work of opening the cursors and fetching the records, it makes the most sense to increase the number of Details lines to the maximum number possible. The reason for this is that the comparison of an IF ... THEN condition is an efficient operation for the RDBMS once the data is actually retrieved. So the difference between the execution ONE IF ... THEN statement vs. 100 IF ... THEN statements is extremely small.
- It might appear that since the number of the detail comparisons is (x Patients) times (y DCM Question Response) times (n Detail Comparisons), this should be equivalent performance if "n" separate procedures are used with a single detail each, but in reality, the difference in performance for the execution of the detail lines themselves is really minute, and it is the Loop Executions which are taking the most time and resources, as well as loading of separate PL/SQL packages in the RDBMS SGA.

PPC: Where Clause Extensions and Qualifying Expressions

--> Outside Cursor Loop: Patients from Patient Positions (x Patients)

--> Inside Cursor Loop: Data from a DCM Question Group (y DCM Question Responses) where (WHERE CLAUSE EXTENSION IS TRUE)

--> CONTINUE IF (QUALIFYING EXPRESSION IS TRUE)

--> Detail Line 1 [IF ... THEN]

.

.

.

--> Detail Line n [IF ... THEN]

--> End Cursor Loop: Data from DCM Question Group

--> End Cursor

Procedure Performance Considerations: Where Clause Extensions and Qualifying Expressions (2)

- By studying this example, it becomes clear the Where Clause Extensions are most useful because this limits the number of records fetched in the Inside Cursor Loop from the DCM Question Group, but the Qualifying expression is not as useful because it skips the details but does not PREVENT the DCM QG cursor from being fully populated. So the Qualifying expression itself behaves similar to an additional FIRST detail line where "Continue if Discrepancy?" is NOT checked.
- Also observe that RESPONSE DATA can only be limited in OC by Qualifying Expressions and NOT by Where Clause Extensions, except for RSN (Repeat SN).
- Where Clause expressions limit data by conditions available in RECEIVED_DCMS table, and not RESPONSES table.
- Therefore, in cases where Qualifying Expressions are used, multiple sets of Loop Executions are still being incurred

PPC: Examining the efficiency of Where Clause Extensions of multiple procedures vs. a Single Procedure with multiple detail lines for a SINGLE QUESTION GROUP

- Suppose that a Where Clause extension is true 50% of the time, and three separate procedures are made:

Procedure 1:
--> Outside Cursor Loop: Patients from Patient Positions (x Patients)
  --> Inside Cursor Loop: Data from a DCM Question Group (y DCM Question Responses) where (WHERE CLAUSE EXTENSION IS TRUE 50% of the time)
    --> Detail Line 1 [IF ... THEN]
  --> End Cursor Loop: Data from DCM Question Group
--> End Cursor
Procedure 2:
--> Outside Cursor Loop: Patients from Patient Positions (x Patients)
  --> Inside Cursor Loop: Data from a DCM Question Group (y DCM Question Responses) where (WHERE CLAUSE EXTENSION IS TRUE 50% of the time)
    --> Detail Line 1 [IF ... THEN]
  --> End Cursor Loop: Data from DCM Question Group
--> End Cursor
Procedure 3:
--> Outside Cursor Loop: Patients from Patient Positions (x Patients)
  --> Inside Cursor Loop: Data from a DCM Question Group (y DCM Question Responses) where (WHERE CLAUSE EXTENSION IS TRUE 50% of the time)
    --> Detail Line 1 [IF ... THEN]
  --> End Cursor Loop: Data from DCM Question Group
--> End Cursor

PPC: Examining the efficiency of Where Clause Extensions of multiple procedures vs. a Single Procedure with multiple detail lines for A SINGLE QUESTION GROUP (2)

- The number of loop executions is [Procedure 1: (x Patients) times (50% y DCM Question Response)] + [Procedure 2: (x Patients) times (50% y DCM Question Response)] + [Procedure 3: (x Patients) times (50% y DCM Question Response)] = 1.5 times the number of loop executions as a single procedure with multiple detail lines, or 50% more loop executions.
- Further, it can be observed that:
- If there is even ONE procedure WITHOUT a WHERE clause extension when multiple separate procedures are used, the efficiency MUST be less than the single procedure with multiple detail lines, since the additional procedures must perform more LOOP executions (unless the WHERE clause is always FALSE ALL of the time, which is nearly impossible).
- If there are TWO OR MORE procedures WITHOUT a WHERE clause extension when multiple separate procedures are used, the efficiency will be at least 50% less than the single procedure with multiple detail lines, since the TWO OR MORE additional procedures must perform at least TWICE AS MANY LOOP executions
- If there are n Multiple Procedures used, where EACH Procedure DOES have a WHERE clause extension, the percentage of records (data) where the WHERE clause extension is TRUE must be (1/n) to match the efficiency of the single procedure with multiple detail lines. So if there are 3 Single Procedures, all with Where clause extensions, then the WHERE clauses extension have to be true only 33% of the time. If there 10 Single Procedures, all with Where clause extensions, then the WHERE clause extensions have to be true only 10% of the time. As more Multiple Procedures are used, even if they ALL have WHERE clause extensions, the probability of this method being more efficient decreases with each new single procedure.

PPC: Examining the efficiency of Where Clause Extensions of multiple procedures vs. a Single Procedure with multiple detail lines for a SINGLE QUESTION GROUP (3)

- Therefore, generally single procedures with multiple details are more efficient in PL/SQL and RDBMS terms for a Single Question Group Case.
- This might seem moot at the initial installation time for an OC system, but recall that the outside cursor of all OC validation and derivation procedures is the number modified patients (in Production mode Batch Validation; it is ALL Patients in Execute Single Procedure mode and also in Test Mode Batch Validation).
- So starting with efficient Validation/Derivation Procedures will save large processing times in the future when there are multiple Batch Validations and the number of modified patients increases in production.
- These differences in coding standards and methodology will only become apparent in the future when there is significant data volume and increased use of the system, but at that time, it will be costly and time consuming to change all of the required procedures and can severely impact the on-going discrepancy management if not done carefully using Procedure Versioning only.

PPC: Multiple DCM Question Group Procedures, considerations in efficient for multiple detail lines vs. separate procedures with single detail lines.

--> Outside Cursor Loop: Patients from Patient Positions (x Patients)

--> Inside Cursor Loop: Data from a DCM Question Group #1 (y DCM Question Responses)

.

.

--> Inside Cursor Loop: Data from a DCM Question Group #Z (y DCM Question Responses)

--> Detail Line 1 [IF ... THEN]

.

.

--> Detail Line n [IF ... THEN]

--> End Cursor Loop: Data from DCM Question Group #Z

.

.

--> End Cursor Loop: Data from DCM Question Group #1

--> End Cursor Loop: Patients from Patient Positions

- In this situation, the number of loop executions is (x Patients) times (y DCM Question Response) times (Z DCM Question Groups). This is a natural cartesian product which can be devastating for performance if the following rules are not used:

PPC: Multiple DCM Question Group Procedures

- 1. Do not add multiple DCM Question Groups to a single validation/derivation Procedure unless absolutely required by the comparison rule.
- 2. Use as many detail lines possible for the same grouping of DCM Question Groups. This minimizes the number of nested Loop Executions (similar to the single DCM Question Group case)
- 3. Wherever possible, use a Correlation Event to join the various DCM Question Groups together
- 4. Wherever possible, use a Correlated Question to join the various DCM Question Groups together
- 5. If #3 and #4 are not possible, try to use a Where Clause extension for each DCM Question Group where possible
- 6. Consider Procedures w/Edit to join nested DCM Question Groups if the join criteria is not available from the Procedures interface itself.

Procedure Cases: Setting Placeholder Flag

- Sets the HAS_DATA flag in the inside cursor loop for nth DCM Question Group cursor, where n > 1 (more than one DCM QG)

```
if I_MODE = 'P'
      then  /* Production */
        if B_CUR%found then
           B$HAS_DATA := 'Y';
        else if B$HAS_DATA = 'Y' then exit;
             else B := NULL_B_RECORD; end if;
        end if;
      else  /* Test */
        if B_CURT%found then
           B$HAS_DATA := 'Y';
        else if B$HAS_DATA = 'Y' then exit;
             else B := NULL_B_RECORD; end if;
        end if;
    end if;
```

- This setting is very useful to determine if an nth DCM Question Group cursor is empty, where the HAS_DATA variable can be interrogated

Procedure Cases: Setting First Repeat Flag

- Sets the FIRST_REPEAT variable and "jumps" out of DCM QG Loop after the first repeat. This is useful for cases where only the first repeat of a repeating question group needs to be checked, such as an Indicator Question.

```
A$FIRST_REPEAT := true;
      if I_MODE = 'P'
        then open A_CUR(RXCPDSTD.PATIENTS_REC.PATIENT_POSITION_ID, A$BEGIN_SEQNUM, A$END_SEQNUM);
        else open A_CURT(RXCPDSTD.PATIENTS_REC.PATIENT_POSITION_ID, A$BEGIN_SEQNUM, A$END_SEQNUM);
      end if;
      loop << fetch_A_cur>>
        RXCPDSTD.V_CODE_LOCATION := 'Fetching A data';
        if I_MODE = 'P'
          then fetch A_CUR into A;
          else fetch A_CURT into A;
        end if;
        if I_MODE = 'P'
          then exit when A_CUR%notfound;
          else exit when A_CURT%notfound;
        end if;

        if A$FIRST_REPEAT
          then A$ACTUAL_EVENT_ID := A.ACTUAL_EVENT_ID;
               A$QUALIFYING_VALUE := nvl(A.QUALIFYING_VALUE,'ZYXZYX');
               A$FIRST_REPEAT := false;
          elsif (A.ACTUAL_EVENT_ID = A$ACTUAL_EVENT_ID) and
                (nvl(A.QUALIFYING_VALUE,'ZYXZYX') = nvl(A$QUALIFYING_VALUE,'ZYXZYX'))
                then goto fetch_A_cur;
                else A$ACTUAL_EVENT_ID  := A.ACTUAL_EVENT_ID;
                     A$QUALIFYING_VALUE := nvl(A.QUALIFYING_VALUE,'ZYXZYX');
        end if;
```

Procedure Cases: Setting Correlated QGs by Event

- In this case, the second DCM QG cursor is opened with an additional parameter of actual_event_id, therefore eliminating a Cartesian Product

```
if I_MODE = 'P'
        then open
S_CUR(RXCPDSTD.PATIENTS_REC.PATIENT_POSITION_ID,
S$BEGIN_SEQNUM, S$END_SEQNUM, A.actual_event_id);
-- Second DCM QG Cursor, uses actual event ID from first "A"
cursor
        else open
S_CURT(RXCPDSTD.PATIENTS_REC.PATIENT_POSITION_ID
, S$BEGIN_SEQNUM, S$END_SEQNUM, A.actual_event_id);
      end if;
-- Second DCM QG Cursor, uses actual event ID from first "A"
cursor
```

Procedure Cases: Aggregate Functions

- In the case below, an average of the number of cigarettes smoked is selected in the Procedure, DCM Questions for Aggregates:
- A separate cursor is introduced to calculate these specific variables, different from the usual DCM QG cursors.

```
select AVG(decode(RES.DCM_QUESTION_ID, 11201, to_number(RES.VALUE_TEXT), null))
 LIGHTCIGARETTES$AV,
AVG(decode(RES.DCM_QUESTION_ID, 11301, to_number(RES.VALUE_TEXT), null))
 REGCIGARATTES$AV,
count(distinct(rdcm.received_dcm_id)) row_count
      from RECEIVED_DCMST RDCM,
           RESPONSEST     RES
     where RDCM.PATIENT_POSITION_ID = I_PATIENT_POSITION_ID
       and RDCM.DCM_ID = 4101
       and RDCM.END_TS = TO_DATE(3000000,'J')
       and RDCM.RECEIVED_DCM_ID+0 = RES.RECEIVED_DCM_ID
       and RES.END_TS = TO_DATE(3000000,'J')
       and RES.DCM_QUESTION_GROUP_ID = 4301 and RES.CLINICAL_STUDY_ID = 2001
       and RDCM.ACCESSIBLE_TS <= SYSDATE
       and RDCM.VISIT_NUMBER between I_BEGIN_VISIT_NUMBER AND I_END_VISIT_NUMBER
       and res.dcm_question_id in (
       11201, 11301)
;
```

Procedure Cases: Aggregate Lag Functions

- In this case, an average across the current and previous visits of alcohol consumption is computed. The procedure stores the previous value in an "L1" variable and then computes the average across the current and previous values.

```
A$ALCOHOLWINE$SM :=
   nvl(A.ALCOHOLWINE,0)
   + nvl(A$ALCOHOLWINE$L1,0);
A$ALCOHOLWINE$CT :=
   nvl(sign(length(A.ALCOHOLWINE) + 1),0)
   + nvl(sign(length(A$ALCOHOLWINE$L1) + 1),0);
RXCPDSTD.V_CODE_LOCATION := 'Lag average calculation';
if A$ALCOHOLWINE$CT != 0 then
   A$ALCOHOLWINE$AV :=
      A$ALCOHOLWINE$SM / A$ALCOHOLWINE$CT;
else
   A$ALCOHOLWINE$AV := null;
end if;
```

Question and Answers

All follow-up questions, please contact:

Sunil G. Singh
singh@clinicalserver.com
+1-860-983-5848
+1-888-463-4751

Tomomi Saito
saito@radpharm.com
+1-609-936-2600 x2246