PharmaSUG 2014- AD19

Tips for Creating SAS®-Based Applications for Oracle Clinical

Kunal Agnihotri, PPD LLC, Morrisville, NC
Ken Borowiak, PPD LLC, Morrisville, NC

ABSTRACT

Oracle Clinical (OC) is database management system used to support clinical trials processes. There is an enormous amount of data and metadata stored in OC beyond the data points collected on a case report form, which can be used as the basis for developing applications to support a variety of activities. This paper offers some tips for creating SAS® macros and standard programs against OC. Topics include using PROC SQL Pass-Through facility, resolving macro variables, handling of dates, and regular expression support. Some useful queries against OC are used to demonstrate each topic.

INTRODUCTION

Oracle Clinical (OC) is a database management system used to support clinical trials processes. For a clinical trials programmer who consumes study data for mapping to SDTM or analyzing the data points, they may be familiar with TEST, CURRENT or STABLE OC views in the Oracle space or the extracted data to the SAS environment. However, the data collection and storage is done so in an hierarchical fashion and is discussed in a paper by Lee (2008). There is a large collection of tables in the RXA_DES (study design tables), RXA_LR (lab and reference ranges) and RXC (study data and data definitions) schemas. There is also the Oracle system tables, which are akin to the SAS dictionary tables, which contain metadata on tables, variables, indexes and other objects.

With all of the information available in OC, there exists the potential to create SAS-based applications to support clinical trial tasks beyond the data points collected on the (electronic) Case Report Forms. This paper offers some tips for creating SAS® macros and standard programs against OC.

PROC SQL PASS-THROUGH FACILITY

In order to access data residing in an Oracle database through requires a license to SAS/ACCESS to Oracle. There are then to two ways to access the data in the Oracle space:

- libname engine
- PROC SQL Pass-Through facility

The simple query as shown in Figure 1 demonstrates a PROC SQL Pass-Through facility query to determine the version of Oracle is installed for that instance.

Figure 1 - Query showing the PROC SQL Pass-Through facility

```
proc sql _method feedback ;
    connect to oracle( path=&lesstaken. user=&me. pw=&myob. ) ;
    select   *
    from     connection to oracle
             (  select   *
                from    v$version  )
    ;
    disconnect from oracle ;
quit ;
```

The user feeds in the path, username and the appropriate password via the macro variables `lesstaken`, `me` and `myob`, respectively. This form of PROC SQL Pass-Through queries are a type of subquery: after the connection credentials are established, the inner query does all the processing in the Oracle space and returns the results to the outer query in SAS environment. The query in Figure 1 is useful for determining the attributes of the Oracle instance.

BANNER
Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - 64bit Production
PL/SQL Release 11.2.0.2.0 - Production
CORE 11.2.0.2.0 Production
TNS for Linux: Version 11.2.0.2.0 - Production
NLSRTL Version 11.2.0.2.0 - Production

Using the PROC SQL Pass-Through facility over the libname engine has the advantage of allowing the Oracle query optimizer to execute the commands. This is important when the query is complex, such as those joining multiple tables together. Oracle is a more sophisticated database system, with B-tree and bitmap indexes and partitions, and is more equipped to handle these queries. The disadvantage of using the Pass-Through facility is that you must use the syntax native to Oracle (i.e. PL/SQL). We'll be exploring some the PL/SQL functions in this paper, as the remaining examples in the paper will strictly use PROC SQL Pass-Through facility. The libname engine method is easy to implement and discussed in Kelly (2005). Once the libref to the Oracle space is established then you can use the tables as if they were SAS data sets.

RESOLVING MACRO VARIABLES

Building applications in SAS typically involves using entails macros or template programs, which implies macro variables are used to collect input values either through macro parameters or global macro variables. When using the PROC SQL Pass-through facility to execute queries, double quotes are not allowed in the Oracle, so getting macro variables to resolve must be addressed. To mend this, macro quoting functions are needed around the macro variables. In the example in Figure 2, the macro variable `study` is a character value is passed into the WHERE clause in order to capture the value of the CLINICAL_STUDY_ID into a macro variable.

Figure 2 - Resolving macro variables in a PROC SQL Pass-Through query with %BQUOTE

```
proc sql feedback ;
   connect to oracle( path=&lesstaken.  ) ;

   %* Create macro variable for CLINCIAL_STUDY_ID ;
   select  clinical_study_id
   into    :csi
   separated by  ''
   from    connection to oracle
            (  select  *
               from    rxa_des.clinical_studies
               where   study =%bquote('&study.')   )
   ;
 disconnect from oracle ;
 %put %str(N)OTE- Macro variable CSI resolves to:  &csi. ;
 quit ;
```

The SAS macro function %BQUOTE are used as a proxy for double quotes. The macro function of %BQUOTE masks any character strings/special characters, which are the single quotes in this case. By using the FEEDBACK option on the PROC SQL statement you can see the code that SAS passes to Oracle to execute in the SAS log, including the resolution of any macro variables.

```
NOTE: Statement transforms to:

       select CLINICAL_STUDY_ID
       from connection to oracle
                     /* dbms=oracle, connect options=(path="OCDB2P") */
                     ( select * from rxa_des.clinical_studies where study ='ABC123' );
```

HANDLING DATES

Oracle stores dates in the DATE data type using it’s own internal format. DATE data is stored in fixed-length fields of seven bytes each, corresponding to year, month, day, hour, minute, and second. One way to reference dates in PROC SQL Pass-Through queries is to use the TO_DATE function, which converts a character string to DATE value based on the format provided in the second argument. The example in Figure 3 returns records from the `CLINICAL_STUDIES` table where the `CREATE_TS` (creation timestamp) is greater than December 01, 2008.

Figure 3 - Date logic using TO_DATE function

```
proc sql  _method feedback ;
  connect to oracle ( path=&lesstaken. user=&me. pw=&myob. ) ;
  select   *
  from     connection to oracle
           ( select  *
             from    rxa_des.clinical_studies
             where   creation_ts > to_date('2008-12-01', 'yyyy-mm-dd')  )
   ;
  disconnect from oracle ;
quit ;
```

The default value for DATE data type fields is DD-MON-YYYY. If you provide a character string in this format then the second argument in the TO_DATE function is not required. Figure 4 demonstrates passing in a character string using the default via a macro variable to ascertain a list of batch validation statuses since a given date. Batch validation is the process of checking data in a study for types of data discrepancies, including univariate re-validation and Discrete Value Group (DVG) resolution.

Figure 4 - Default DATE format in TO_DATE function

```
%let date1=01-DEC-2013;
proc sql feedback ;
   connect to oracle ( path=&lesstaken. ) ;
    create table bv_status as
    select  *
    from    connection to sup
            ( select  study
                    , batch_job_id
                    , execution_status
                    , submission_ts
                    , user_name
                    , failure_text
                    , log_file_name
               from   rxc.batch_jobs
               where  study like '%XYZ123%'
                     and submission_ts > to_date( %bquote('&date1.') ) )
    ;
   disconnect from oracle;
   quit ;
```

REGULAR EXPRESSION SUPPORT

SAS users are afforded the ability to harness the power of flexible pattern matching and text substitution with Perl Regular Expressions through the PRX family of functions. Fortunately, there is regular expression support for Pass-Through queries against OC tables with the REGEXP functions beginning with the Oracle 10g release. The REGEXP functions are explored in detail by Agnihotri and Borowiak (2012), drawing on the similarities with the PRX functions in SAS. The primary difference between the two family of functions is the order of the arguments. The query in Figure 7 against the Oracle system table ALL_IND_COLUMNS returns records with the COLUMN_NAMEs that begin with `COORD_` followed by either `AXIS` or `SYS`. The beginning of the string anchor (i.e. the ^ symbol) and the parenthesis and **|** symbol for alternation is syntax from Perl, which both the Oracle REGEXP and SAS PRX functions mimic. The third argument ‘c’ in the REGEXP_LIKE operator requests that the search is case-sensitive.

Figure 7 - Query to confirm version of the Oracle client using REGEXP_LIKE function

```
proc sql_method feedback ;
  connect to oracle( path=&lesstaken. ) ;
   select  *
   from    connection to oracle
           ( select  column_name
             from    sys.all_ind_columns
             where   regexp_like ( column_name, '^COORD_(AXIS|SYS)' , 'c' ) )
   ;
   disconnect from oracle ;
 quit ;
```

COLUMN_NAME
COORD_SYS_ID
COORD_AXIS_NAME
COORD_AXIS_NAME_ID
COORD_SYS_TYPE
COORD_SYS_NAME

CONCLUSION

There is a rich amount of data and metadata in Oracle Clinical beyond what is in the extracted clinical data into SAS. The examples in this paper were rather rudimentary, but they can serve as a basis for more complex queries and applications for clinical trials programs executing them in a SAS environment.

REFERENCES

Agnihotri, Kunal and Kenneth Borowiak (2012), "A SAS® Users Guide to Regular Expressions When the Data Resides in Oracle".
http://www.pharmasug.org/proceedings/2013/PO/PharmaSUG-2013-PO12.pdf

Borowiak, Kenneth (2012), "A Closer Look at PROC SQL's FEEDBACK Option"
http://analytics.ncsu.edu/sesug/2012/CT-05.pdf

Fogelman, Stanley (2005), "SAS and Oracle PL/SQL: Partners or Competitors?"
www.lexjansen.com/pharmasug/2005/technicaltechniques/tt07.pdf

Kelley, Joseph F. (2005), "Getting Started with SAS/Access for Oracle"
http://analytics.ncsu.edu/sesug/2005/ETL05_05.PDF

Lee, Kevin (2009), "Oracle Clinical for SAS Programmers"
http://www.lexjansen.com/pharmasug/2009/po/po20.pdf

ACKNOWLEDGEMENTS

The authors would like to thank Renuka Mirani, Hunter Everton and Archana Chaudhari for their insightful comments on this paper.

DISCLAIMER

CONTACT INFORMATION

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

Kunal Agnihotri
3900 Paramount Parkway
Morrisville NC 27560
kunal.agnihotri@ppdi.com

Ken Borowiak
3900 Paramount Parkway
Morrisville NC 27560
ken.borowiak@ppdi.com