

Building bridges between Oracle ® and SAS®

Bill Calvert - FDA Rockville, MD

Feng Zhou - FDA Rockville, MD

ABSTRACT:

In the context of a data-rich client-server environment, techniques are presented showing how to use SAS with SQL and ODBC to connect to Oracle Databases, develop views, and extract working SAS data sets. In addition, ways to survey data tables, prepare output and format fields are shown. Emphasis is placed on automated approaches (using SAS/MACRO for example) to minimize otherwise laborious and time consuming tasks.

INTRODUCTION:

We work in a data-rich environment. In the post-market drug safety program, over 250,000 spontaneous adverse drug reaction reports were put into our Oracle database in the last fiscal year. With 1.6 million records, this is just one of several data sources that we must work with every day. Frequently, fast and accurate access to data is required.

The part of the infrastructure that we work with consists of a VAX/ALPHA cluster with thousands of PCs, WAN connections for multiple buildings, etc. Unfortunately, since SAS is used only by isolated pockets of analysts with some notable concentrations in the statistical areas, SAS is only available on PCs or UNIX workstations. In effect SAS runs on the local PC or fat-client and the data is obtained using the services of the ODBC driver from Oracle database servers across the network.

The typical program that we use connects to a data source, accesses and selects a portion of the data, and creates a report for printing or creates a local data set for further analysis. In the subsequent sections of this paper we share how we get to the data, and prepare it for further use.

MAKING THE CONNECTION:

There are several ways to connect to Oracle using SAS. In our environment, SAS does not exist on the data servers, only on the client PC (or sometimes UNIX computer). Consequently we are limited to only one way to do it. The statements used to connect and disconnect as well as the other code presented here are applicable in other environments.

First, to connect to an Oracle database, you need to know the name of the database and its server and have appropriate security permissions. Also be sure you have the ODBC driver appropriately installed on your system.

Here is the code for a simplified example of the type of code you would use. To get started, within your SAS program call the SQL procedure.

```
PROC SQL;
CONNECT TO ODBC
(DSN=TESTPROD
UID=FENG
PWD=PASSCODE);
```

Note that the connection is by a statement within the SQL procedure and that the DSN or database name is included in the parameter list with the user identifier and password. Be aware that having the password written in your code may be a security problem, especially if you are not careful with printouts and listings.

Once you have made the CONNECT statement, you can write SQL statements as you usually would. In a simple case, you might create a permanent SAS data set, SELECT the entire data structure, and procedure to qualify the data to be extracted and saved for later use.

```
CREATE TABLE DRUG1.SAMPLE1 AS
SELECT *
FROM CONNECTION TO ODBC
(SELECT /* SQL/PLUS CODE */
...);
```

Continuing for the simple case, after the SQL used to qualify the data you would DISCONNECT from the data source and use the QUIT statement to end PROC SQL.

```
DISCONNECT FROM ODBC;
QUIT;
```

At this point you should have successfully obtained data from the Oracle source and stored it locally in a permanent SAS data set. You can now handle the data as your skill allows in SAS.

If you were not successful, there are several kinds of problems. First, if someone moved the data, changed access codes or you spelled a name or password wrong, the connection will not work. Also consider that the installation and parameterization of the ODBC driver may be flawed.

For example, in our Windows95/DEC environment, error messages have indicators relating to these problems such as:

[ODBC driver manager]

Data source name is not found - need to go to ODBC administrator to add or change the data source name.

[Oracle ODBC driver]

TNS: could not resolve the service name - need to add or change the value stored in the file called TNSNAMES.ORA.

If you suspect this sort of problem, you may need to get some help from your local technical support staff.

There are other common problems associated with SQL statements. Make sure your connection works before you do any 'fancy' SQL. This will make it easier to debug your program.

The same approach can be used to create a 'view' of the data. The difference is that you change 'TABLE' to 'VIEW' as shown in the lower case part of the code below.

```
CREATE view DRUG1.SAMPLE1 AS
SELECT *
FROM CONNECTION TO ODBC
(SELECT /* SQLPLUS CODE */
...);
```

When working with large and sometimes real time data sets as we do, it is important to consider the benefits of the data view as well as the advantages of the extract of the data that you may store locally or on a server. For ever changing data, the data view is attractive because it provides current results for your extraction query. If you write a data table, you must explicitly update it to keep it synchronized with the source data tables. Unfortunately when you use a view, with very large data systems, your program may run very long each and every time you use it, and if you are not careful with your SQL, your program may impact overall system performance and affect many other people. We have over a million records for example in hundreds of tables, with hundreds of users on our Adverse Event data base. For us the choice of a view or a table must be made with careful thought.

Of course, one can simply write query results to a file or table and perform subsequent work only on the extracted data. The initial run to get the data takes about the same time as if it were a data view, but subsequent runs go against a much smaller file and go faster and usually don't impact any other users. The down side of this approach is that the data may become obsolete before the analysis is completed.

In our circumstances, we usually use the data view for well tested code that is at least semi-routinely used and where the results must be current as of the date of the output. We use the 'extract to a table stored locally' approach to develop new code or first time analyses, keeping in mind the need to update the data from the source and rerun all programs when we are ready to produce the final results. We are also routinely careful how SQL is written and used.

MAKING IT EASIER:

One advantage we have in working with our very large data sets is that we use the same structures time and again. Having the need alluded to above to have well tested tools and techniques, we have developed some ways to make things easier using formats and macros.

Often codes are stored in tables with one or more definitions, descriptions or symbols that may be used to help make the data understandable to readers/users of reports, screens and other output form the database. For example you may have a table such as this:

CODE	DESCRIPTION
001	MALE
002	FEMALE
003	UNKNOWN

We use PROC FORMAT to associate the code with the label or description. Then, when the data is printed or displayed we get an understandable 'string' instead of just the code. For example:

```
PROC FORMAT;
VALUE $ COD
```

```
'001'    'MALE'
'002'    'FEMALE'
'003'    'UNKNOWN'
```

In this case, the value 1 is associated with the string 'MALE' and the value 2 is associated with 'FEMALE' and so forth.

Listing out the tables and typing in all of this stuff for lots and lots of codes, and not making any mistakes is at least laborious and usually very time consuming to verify. To add to the difficulty the codes sometimes change, expand or are augmented with new spellings or information. It is difficult to be sure your output is correct and up to date. The technique we have developed uses an ODBC data view of the Oracle database reference tables where the code is stored, a subprogram parses and converts the information in the reference tables into SAS code (a big PROC FORMAT). The SAS code generated by the subprogram is executed under MACRO control and generates a format library that can be updated as necessary and used for all jobs running off of the same database.

The subprogram (See PROGRAM 1) is for the parsing and conversion of the code data stored in the Oracle reference tables. Note that the PROC FORMAT statement is written out using a PUT statement. Then the values of codes and the meanings (or descriptions) of those codes are interpreted from the incoming data set and written out into proper SAS syntax using a BY controlled loop structure. This useful piece of code is set up as a MACRO.

MACROs are very useful for repeating carefully coded processes. See PROGRAM 2 for an example of how the subprogram just described may be called by another program. In PROGRAM 2 note that the ODBC connection is made, data is obtained using a data view for two tables. Subsequently, the data is sorted in preparation for conversion to a SAS format library. There are also parameters that are set to accommodate differences in the names of fields in ORACLE tables, etc. The subprogram is called for each Oracle data source and the formats that are generated are appended in the same SAS format library.

Keep in mind that this type of process almost always looks simple and easy when it is done. There are a number of problems that can occur at different levels, and it may be very difficult to separate them out if you are not sure of any of your component programs. It is best to develop your code from the bottom up. That is, build the subprogram and test it. Build that connects to the data base and test it, then add the MACRO features and elevate your code to the higher level of functionality and flexibility.

AUTOMATING ANOTHER PROCESS:

Now that we can get data and have an automated means of capturing the meanings of arbitrary codes for use in our SAS programs, what about checking out various data sources to see if they really are what we need to analyze? In a fast paced environment with lots of people and lots of data, it is sometimes not possible to get sufficiently detailed documentation, much less an overview or snapshot of the data. Sometimes a programmer analyst must service multiple projects for multiple people, and sometimes we are so rushed we need to go back and 'remember' the content of various temporary files as we clean up and organize for the next rush.

PROC CONTENTS is a handy tool for finding out the structure of a SAS data set, data view, or some other special cases. It is especially useful to us for surveying data sets because it allows

you to get an output file that can be parsed and used to control other processes which explore the data set.

If you wanted to check out some data, the usual process is to run PROC CONTENTS. Look at the output and set up a 'survey'

Program 1:

```

*****
*** PROGRAM:      FMTWRITE.MAC:                ***
***                                                     ***
*** PROJECT:      Create a format library from data sets ***
***                                                     ***
*** WRITTEN BY:   proc_1                        ***
*** DATE:        3/25/98                       ***
***                                                     ***
*****

FILENAME SASCODE "C:\ZHOU\SASCODE\FMTLIB.SAS";

%MACRO FMTWRITE;
/*****Generate proc format statement *****/

DATA _NULL_;
FILE SASCODE &APPEND;
SET &DSIN END=EOF;
BY &FMT_NAME;

RETAIN NFORMAT 0;
/* Number of formats in current procedure */

RETAIN LFORMAT;          /* Number of line of value */
RETAIN T_CODE;           /* Format name */

IF _N_ EQ 1 THEN DO;
PUT "***** Create a format library from SAS data or view
***";
PUT "/* PROC FORMAT LIBRARY=LIBRARY ; ";
END;

IF FIRST.&FMT_NAME THEN DO;
T_CODE = &FMT_NAME;
LFORMAT = 0;
NFORMAT = NFORMAT + 1;
END;

LFORMAT = LFORMAT + 1;

IF LFORMAT = 1 THEN DO;

PUT "/* VALUE $" @;
PUT &FMT_NAME @;
END;

IF LFORMAT > 0 THEN DO;
PUT "/* " @;
PUT &FMT_VALU @;
PUT "/* " @;
PUT &FMT_MEAN $30. @;
PUT "/* " @;
END;

IF LAST.&FMT_NAME THEN DO;
PUT "/* ";
END;

T_CODE = &FMT_NAME;

IF (EOF AND NFORMAT GT 0) THEN DO;
PUT "RUN;" ;

```

```

NFORMAT = 0;
END;

RUN;
/***** End of proc format statements *****/
%MEND FMTWRITE;
%FMTWRITE;

```

Program 2:

```

*****
PROGRAM:      AERS_FMTS.sas
*
* PROJECT:      This SAS program codes are used to create
*a format library according to the AERS REFERENCE TABLE:
*
* 1. AERS_CG_REFS
* 2. AERS_CG_REF_TYPES
*
*FUNCTION:
* 1. Create SAS view through ODBC CONNECT TO ORACLE
* 2. Using view write the sas code
* 3. Run the sas code to create a format library
*
* WRITTEN BY:   proc_1
* DATE:        12/10/97
*
*****

OPTIONS PAGENO=1 LS=155;
LIBNAME AR97 'C:\AERS_97';

* Create a connection to ORACLE through ODBC;
PROC SQL;
CONNECT TO ODBC (
DSN=AERSTEST
UID=USER1
PWD=PASSWORD);

* 1. Create a AERS_CG_REFS view;
CREATE VIEW CG_REFS AS /*AERS_CG_REFS Table */
SELECT * FROM CONNECTION TO ODBC
(SELECT * FROM AERS_CG_REFS);

* 2. Create a AERS_ROLES view;
CREATE VIEW ROLES AS /*AERS_ROLES Table */
SELECT * FROM CONNECTION TO ODBC
(SELECT * FROM AERS_ROLES);

* Delete a connection to ORACLE through ODBC;
DISCONNECT FROM ODBC;

* Sorting the view;
PROC SORT DATA=CG_REFS OUT=REFS;
BY TYPE_COD;
PROC SORT DATA=ROLES OUT=ROLE;
BY ACTIVE_C;

RUN;

%LET APPEND = ;
%LET DSIN = REFS;
%LET FMT_NAME = TYPE_COD;
%LET FMT_VALU = CODE;
%LET FMT_MEAN = MEANING;
%INCLUDE 'C:\ZHOU\MACRO\FMTWRITE.MAC';
%INCLUDE 'C:\ZHOU\SASCODE\FMTLIB.SAS';

DATA ROLE;
SET ROLE
IF ACTIVE_C = '1' THEN ACTIVE = 'ROLE';

%LET APPEND = MOD ;

```

```
%LET DSIN = ROLE ;
%LET FMT_NAME = ACTIVE;
%LET FMT_VALU = NAME;
%LET FMT_MEAN = DESCRIPT;
%INCLUDE 'C:\ZHOU\MACRO\FMTWRITE.MAC';
%INCLUDE 'C:\ZHOU\SASCODE\FMTLIB.SAS';
```

program that would do PROC MEANS on the numeric variables, PROC FREQ on the categorical variables, and PROC PRINT on a sample of records so you can get the 'feel' of the data. Hand coding each of these programs for each data set on separate occasions separated by long (or short) periods of time takes time and attention to detail that is becoming increasingly precious in today's work environment. If we think carefully, we can get the computer to help and save some time, while boosting accuracy.

Program 3 is an example of a data 'survey' or data checking program. This program uses ODBC to access Oracle tables and create a data view. After disconnecting from the Oracle source, PROC CONTENTS is run on the data view and an output file is created that holds the structural information about the data view. The data checking program then calls the SAS code writing MACRO program and subsequently the code that was generated to do the actual survey work on the data. Note that there are MACRO parameters to allow the program to be adjusted to read different tables, put useful information in titles and footers, etc. For simplicity, this program only reads from one data source in any given run.

Program 4 is a portion of the SAS code writing MACRO program referenced above. There are three segments in this program, one which writes the PROC MEANS code, one to write the PROC PRINT code and the final one which writes the PROC FREQ code. The last segment has been omitted to conserve space (and provide an exercise for the reader?). Each segment uses the same input file, the output from PROC CONTENTS.

The main problem in doing PROC MEANS, (refer to the first segment of the program) is to determine the variable names that will be in the VAR statement and write them out so that the statement has good form, regardless of the number of variables in the statement. Numeric variables are identified by the value of the variable TYPE. If there is a numeric variable the code is started with the PROC MEANS statement, accumulators are established to control the number of variables on a line, and the VAR statement is established. For subsequent numeric records, the variable name is added to the list. At the end of the file, the final semicolon is written for the VAR statement and the title is set.

The second segment of the program creates a PROC PRINT. This is done without regard to the character or numeric qualities of the variables. Note that the data set option is set to (OBS=50). This is about one page of data on our printer and serves to give a 'feel' for the data.

The third segment of Program 4 (which is not shown) is very similar to those just described. While it may be of some value in the PROC MEANS or PROC PRINT segment to account for formats, it is much more important to display formatted values in the PROC FREQ segment. We have relied on creating a format library from Oracle as already described and associating these with the format names from the PROC CONTENTS output.

Looking back at Program 3 briefly, notice that the last statement before the RUN is a commented out %INCLUDE which would, if active pull in the code as written by the MACRO referred to as WRITEFRQ.MAC as displayed in PROGRAM 4. It is often useful to generate the code and check it before execution, especially when you are expecting large data files and long run times.

Program 3:

```
*****
* PROGRAM:  AERS_CHK.SAS
*
* PROJECT:  AERS database checking
*
* FUNCTION:  Produce the preliminary frequencies and
             checking *           the duplicate in AERS table
*
* WRITTEN BY: tester
* DATE:     3/24/98
*****/
%LET PGM = AERS_CHK;
%LET PATH = C:\ZHOU\SASCODE;
%LET A_TABLES = AERS_REACTIONS;
%LET FILTER = ;

LIBNAME IN "&PATH";
OPTIONS LS=155 PS=50 PAGENO=1 NODATE;

***** Start program *****

* Main title and foot notes
TITLE1 ' ';
TITLE2 ' ';
TITLE3 "AERS DATABASE - &A_TABLES ";
TITLE5 ' ';
TITLE6 ' ';

FOOTNOTE;
FOOTNOTE1 ' ';
FOOTNOTE2 "FENG ZHOU, &SYSDATE,
PGM=&PATH.\&PGM..SAS";
FOOTNOTE3 "INPUT = &A_TABLES";
FOOTNOTE4 "OUTPUT = DATA CHECKING RESULTS ";
FOOTNOTE5 ' ';
FOOTNOTE6 ' ';

* Read in data from ORACLE database - AERS tables
*
PROC SQL;
CONNECT TO ODBC(
    DSN=AERSTEST
    UID=USER1
    PWD=PASSWORD);

CREATE VIEW _TEMP1 AS
SELECT * FROM CONNECTION TO ODBC
(SELECT *
FROM &A_TABLES
&FILTER);

DISCONNECT From ODBC;
QUIT;

* Create a proc contents output file
PROC CONTENTS DATA=_TEMP1 OUT=_TEMP;

* Call macro WRITEFRQ.MAC to create a SAS code

%LET CHAR_LEN = 4;
%LET DSIN = _TEMP1;
%INCLUDE "C:\ZHOU\MACRO\WRITEFRQ.MAC";
*%INCLUDE "C:\ZHOU\SASCODE\FREQUEN.SAS";
RUN;
*****End program*****
```

Program 4:

```

*****
*** PROGRAM:   WRITEFRQ.MAC          ***
***          ***                    ***
*** PROJECT:   Create a data screen checking SAS code
***          ***                    ***
*** FUNCTION:  Automatically write the SAS code
***          ***                    ***
*** Programmer: pro_1
*** Date of Written: 3/25/98
***          ***                    ***
*****
FILENAME SASCODE "C:\ZHOU\SASCODE\FREQUEN.SAS";

%MACRO WRITECOD;

/***** Generate proc means code *****/
%LET NVARLINE=7;
/* Number of variables listed per source line */

DATA _NULL_;
FILE SASCODE;
SET _LAST_END=EOF;

RETAIN NUNIVL 0;
/* Number of variables on current line */

RETAIN NUNIV 0;
/* Number of freqs in current procedure */

IF _N_ EQ 1
  THEN PUT
    /****** MEANS ON CONTINUOUS VARIABLES *****/;

IF TYPE = 1
  THEN DO;
    NUNIV = NUNIV + 1;
    NUNIVL = NUNIVL + 1;
  END;

IF NUNIV = 1 AND TYPE = 1
  THEN DO;
    PUT 'PROC MEANS
      DATA=&DSIN N NMIS MIN MAX MEAN STD ;';
    PUT ' VAR ' @;
  END;

IF TYPE = 1
  THEN PUT NAME $9. @;      /* Output variable name */

IF NUNIVL EQ &NVARLINE
  THEN DO;
    PUT / ' ' @;
    NUNIVL = 0;
  END;

IF (EOF AND NUNIV GT 0)
  THEN DO;
    PUT ";";
    PUT ' TITLE4 "MEANS OUTPUT ";';
    PUT "RUN;" /;
    NUNIV = 0;
  END;
/***** END OF PROC MEANS CODE *****/
RUN;

```

```

/***** Generate proc print code *****/
%LET NVARLINE=7;
/* Number of variables listed per source line */

DATA _NULL_;
FILE SASCODE MOD;
SET _LAST_END=EOF;

RETAIN NPRINTL 0;
/* Number of variables on current line */

RETAIN NPRINT 0;
/* Number of freqs in current procedure */

IF _N_ EQ 1
  THEN PUT /****** PRINT SOME VARIABLES *****/;

IF LENGTH > &CHAR_LEN
  THEN DO;
    NPRINT = NPRINT + 1;
    NPRINTL = NPRINTL + 1;
  END;

IF NPRINT = 1 AND LENGTH > 2
  THEN DO;
    PUT 'PROC PRINT DATA=&DSIN(OBS=50)
      ROWS=PAGE WIDTH=MIN LABEL;';
    PUT ' VAR ' @;
  END;

IF LENGTH > &CHAR_LEN
  THEN PUT NAME $9. @;      /* Output variable name */

IF NPRINTL EQ &NVARLINE
  THEN DO;
    PUT / ' ' @;
    NPRINTL = 0;
  END;

IF (EOF AND NPRINT GT 0)
  THEN DO;
    PUT ";";
    PUT
      ' TITLE4 "PRINT FOR REST OF THE VARIABLES ";';
    PUT "RUN;" /;
    NPRINT = 0;
  END;
/***** End of proc print code *****/
RUN;

/***** Generate proc freq code *****/

/* [ SAS code not shown to conserve space ] */

/***** End of proc freq code *****/
RUN;

%MEND WRITECOD;

%WRITECOD;

```

DISCUSSION:

Our culture is becoming more information intensive. Rapid and accurate access to data is more and more taken for granted. In fact, lack of access to data may be seen as negligence. In this milieu, data-rich environments will be prevalent and the need to bridge between systems will increase.

As a very small part of this cultural trend, we have described how to use SAS and ODBC to connect to very large Oracle databases. In sharing some of the ways we have used SAS to overcome the difficulties inherent with large volumes of data in a complex infrastructure, we have also demonstrated a less obvious trend that must be part of our future; programs that write other programs. We have developed and used the tools shown here in conjunction with others in our organization to ease the repetitive burdens of high volume high accuracy work. More importantly we are advancing the quality of our work by using the computer to do what humans do not do well; highly accurate repetitive work.

On another level, the code we present here demonstrates the necessary features of code that can be reused. For example, the identification block at the beginning of the program file, in-stream comments, careful spacing and indentation to delineate the logic of the code, and white space to enhance readability. The increase in the sheer volume of data and hence code to manipulate and make use of data, requires a higher standard of documentation just to avoid chaos like we all have experienced in times of overwhelming work loads.

We have also provided some insight into the process by which we build code, test and validate it. Clearly much more can be done in this area. As computing systems become increasingly complex and the demands for accuracy and accountability expand, some of the techniques used here may need to be modified to provide a more complete audit trail. Sometimes a good and diligent effort is not enough, unless you can document precisely what you did.

Some mention was made of performance issues in considering the data view versus the permanent data set. While it is important to understand the differences and trade-offs; neither one is better than the other. The efficacy of SQL queries was mentioned in relation to performance as well. Often performance issues and concerns can only be resolved by a dedicated team of experts. Individual programmers and users are often isolated from the realities of moving bits around and there are worlds of hardware capacity that keeps getting cheaper. Often we just get more hardware to solve our performance problems. Hardware capacity that is not used wisely is wasted. We need to do more to evaluate performance and identify the things that can be done by the individual to encourage good performance from our programs and systems without losing sight of the need to also get good programmer (or user) performance.

On a detail level, more work also needs to be done to fully link coded information to appropriate SAS formats. While we have had some success, we are relying on what may prove to be unique features of our system.

All of the programs presented here make use of SAS MACROS to carry parameters across DATA step and PROC boundaries, as well as to allow repetition of very similar processes. MACROS are also used to simplify maintenance and to make code truly reusable. Creating SAS programs to write SAS programs allows us to have data driven programs that leverage programmer time and effort. In our environment this means that the client (PC) provides the performance boost to the programmer/user without loading up the server.

We want to encourage the extension of the approaches used here to other systems. Perhaps the code provided here can inspire more efforts to write programs that write programs. Ultimately we hope to help others who must bridge from SAS to a range of diverse data sources. We believe some of this code will be similar for other ODBC compliant data bases.

SAS is a registered trademark of SAS Institute, Inc. Cary, NC

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

CONTACT:

The authors may be reached by electronic mail

CALVERTW@CDER.FDA.GOV
ZHOUF@CDER.FDA.GOV