

Creating a Dynamic SAS®-ORACLE® Interface Using SAS/AF® FRAME

Paul Ehresmann, Paul Ehresmann, Inc., Indianapolis, IN

Abstract

An interface using SAS/AF® FRAME applications may be created to perform various dynamic ORACLE® (or other) database procedures, including table creation/deletion, inserting or deleting rows, and other database processes. Interactive creation of macros, PROC COMPARE screens, SAS® data file creation, SQLPlus and PROC SQL procedures, and similar actions may be invoked within linked programs and SCL lists.

This presentation will include a dynamic application currently being used in a clinical research organization. The visual aids for this presentation, a catalogued library consisting of the overall application, and sample programs are available for downloading as a self-extracting .EXE file from:

<http://www.sasmann.8m.com>

Background

Although a pass-through capability via ODBC using SAS/ACCESS® can result in performing excellent queries against ORACLE® or other database tables, the user is normally unable to invoke the EXECUTE statement as part of an interactive, dynamic connection between SAS® and the database. SAS® is unable to process multiple EXECUTE commands as part of the pass-through query.

The outgrowth of this non-capability is the requirement to perform many query functions within a SAS® session and many dynamic database functions within an interactive database session.

In a clinical research organization located in Indianapolis, IN, this writer was contracted to correct this situation by creating within SAS/AF® FRAME SCL and programs the means to:

- Provide the language to allow invocation of the EXECUTE statement and its dynamic functions.

- Allow creation, editing, and storage of specific SAS® programs and macros, interactive submission of these programs/macros to perform the creation and updating of SAS® data files, and comparison of SAS® data files and display of these PROC COMPARE results within a SAS/AF® FRAME.
- Interactively create and execute various SQL commands by invoking PROC SQL and/or SQLPlus programs.

The client operated in a Windows NT server environment, with SAS® version 6.12 loaded on each workstation, and ORACLE® version 7.0 loaded on one of the servers. Since PROC DBLOAD and SAS/ACCESS® have the capacity to utilize variable names only eight characters in length, this limited-length situation was very important in the dynamic interchange between SAS® and ORACLE®, especially in the creation of tables with column names longer than eight characters.

Version 7 Limitations

Although the latest (as of this writing) version of SAS® maintains the capability of utilizing various files with names lengthier than DOS files, both PROC DBLOAD and SAS/ACCESS® continue to have the capacity to use variable names limited to eight characters. This limitation must be considered when dealing with any database product, which uses large column names. This situation may alter with later releases of SAS®.

Initial Development Stages

The first considerations in this project were the creation and enactment of SAS® modules to process raw laboratory data received from various sources that performed the data analysis for numerous pharmaceutical companies. This data would appear in the form of flat files loaded to diskettes, hardcopy forms processed by optical scanning software, or

electronically transmitted data submitted either as attachments to electronic mail or downloadable files.

Once the programs had been created and processed the data, a flow chart was created which detailed the overall process from initial development of questionnaires thru data gathering techniques thru data verification, processing, and placement of results into SAS® data files and ORACLE® tables. This instrument provided the guidelines by which SAS/AF® FRAMEs/SCL and programs would be designed.

The SAS/AF® Design Process

With all processing requirements fulfilled, it was possible to design the physical format of the primary FRAME from which all other FRAMEs, SCL and programs would eventually be developed. The format of this FRAME is displayed below:

When this SAS/AF® application was invoked by the user, a small FRAME would appear to allow the entry of the ORACLE® userid and password. Using SYMPUT and SYMGET para-meters, these entries (as well as those from the main FRAME) would be carried forth to other SCL and programs. The FRAME displayed above would then appear to allow further pertinent entries.

The *Userid* and *Password* entries were passed to this FRAME. The password remained hidden to meet security requirements.

The *SAS Data File* entry area (1) provided the means to create the data file by interactively submitting a lab program, (2) stated the source from which data would

be transmitted to an ORACLE® table, and (3) noted one of the two SAS® data files to be used for PROC COMPARE.

The *ORACLE Table Name* entry area required the user to include the owner of the table, even if this was the person invoking the SAS/AF® application. As such, and with proper authorization from the ORACLE® DBA, any database table could be appropriately processed.

The *Source Table* would have an entry as needed to allow the user to view all the columns and records of any ORACLE® table in a SAS® view format. Obviously, the column names were shortened to eight characters during this viewing process; however, the

user was able to see the default informats should this table be the source in the creation of a SAS® data file.

Since numerous database users currently maintain test and production copies of tables, files, etc., the *Mode* entry area was provided to allow the user to enter either PROD or TEST. (For other operations, these choices may be retained, removed, or altered to meet acquisition needs.)

The ORACLE® Functions Box

With all required entries accomplished, the user could select any of the processing options noted in this area. A user would often perform more than one function during a SAS/AF® session, many times processing the SAS® data file, then performing various ORACLE® functions, then viewing the results as needed.

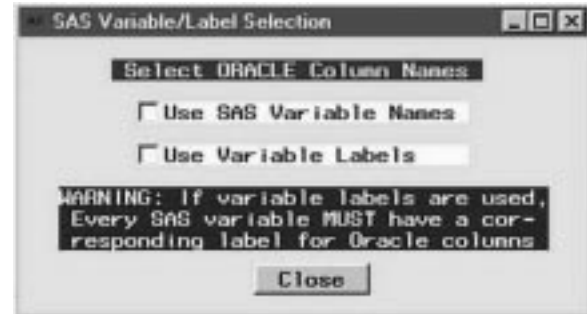
PROC DBLOAD created an ORACLE® table named USERID.SAS_LOAD from the SAS® data file whose name had been entered into that area. Default informats were forwarded from the table to the data file, and SAS® variable names were retained as the table column names. This table could be used for other functions.

To create or repopulate a SAS® data file from raw input data received from one of the labs, the *Process Infile Data* option was selected. An entry consisting of the full path and SAS® module name would be made in the *...View/Edit SAS Code...* entry area. Upon invocation, the program's code was displayed in edit mode to allow the user to make any necessary changes; e.g., the input file's name. The module was automatically saved and submitted for processing. The new/repopulated SAS® data file could be used for other functions.

Create ORACLE® Table

The *Create ORACLE Table* option performed exactly that role: a temporary table was created from the input SAS® data file via *PROC DBLOAD*, a new ORACLE® table was created, and the temp table was deleted.

This specific function would call another FRAME that allowed the user to use either the default SAS® variable names or the labels attributed to those variables as the column names for the new ORACLE® table.



This selection process was very important. If lengthy column names were required, the user had to make sure each variable had a label written exactly as the column name. For example, a column with the name ENTRY_DATE_TIME would require its input SAS® variable to have ENTRY_DATE_TIME as its label, including the uppercase format.

The SCL written to create the new table read which of the two column-name requirements was selected, and SQL language in the form of a SAS® macro was interactively written to create the new table with the correct column-name formats.

To delete the temporary table created by the *PROC DBLOAD* option or used by other options, the *Delete SAS_LOAD* option was selected.

Drop ORACLE Table gave the user the opportunity to completely delete any table for which the user had such authority. This option could be somewhat dangerous, because the wrong table, i.e., the table noted in that entry area, would be deleted. If no backup copy existed for that table, there was no manner of recovering that table and its values. As such, a message appeared in the upper right corner of the FRAME warning the user he/she needed to be aware of this unfortunate possibility.

Noted previously in this paper was that ability for the user to call dropdown lists to populate certain entry areas. The *Update Dropdown Lists* option gave the user a capability to change one or more of these lists to match the path/ownership of SAS® data files and/or ORACLE® tables. These lists were created in such a way that entries other than those found in the dropdown lists could be entered without causing an error situation. The was usually the first option selected, especially if work was being performed within a single session on SAS® data files and/or ORACLE® tables owned by different people.

In order to view and/or alter the values within the SAS® data file entered in its appropriate area, the user selected *View/Edit SAS Data File*. A FRAME very similar to the SASVIEW® product appeared. The user had the opportunity to alter values as needed and save those changes.

If the user wanted to retain all records within an ORACLE® table and insert additional records, the *Insert Table Rows* option was selected. This option made sure all column names and values remained the before and after the completion of this process. This option was important in the maintenance of the table, especially if indexes or other unique features were part of the ORACLE® table.

Delete Table Rows performed exactly that role: it simply deleted all records within an ORACLE® table while retaining the column names and properties for that table. As with the Drop ORACLE Table function, the user needed to be positive that the records were indeed to be removed from the table, and the lack of a backup could also result in similar permanent loss of data.

If a value change was to be performed to a column residing in the ORACLE® table, the user selected the *Update ORACLE Table* option. A FRAME appeared that cued the user to enter (1) the column for which the value was to be changed and (2) the new value itself. Although available, this option was actually seldom selected. The ORACLE® DBA preferred to retain this capability for himself...apparently some type of power issue.

PROC COMPARE was one of the high points to this SAS/AF® application, because the user was provided the capability to view within one FRAME the observations alone containing differences from two SAS® data files or differences found in the observations of a SAS® data file and the equivalent records from an ORACLE® table displayed in SAS® data file format. The output could be viewed in either browse or edit mode. The normal PROC COMPARE does not provide this capability; therefore, this double view was often selected by users.

The FRAME used for this PROC COMPARE allowed the user to not only select the two SAS® data files (or one data file and an ORACLE® table), but also a limited list of variables/columns for value comparison. It also provided a SORT selection to assure the variables/columns matched each other on a record-to-record basis. This FRAME is not displayed in this document.

In this clinical environment, the ORACLE® database structure had been designed to require an _UPDATE table to be populated and validated before loading records to a _DATA table. The majority of table work within this SAS/AF® application therefore applied to such an _UPDATE table.

SQLPlus™, an ORACLE® software product that supports ORACLE® version 7.0, had been loaded on the server. This product required very specific language in order to invoke database commands. To allow the creation of .SQL files containing this language, an option called *Create _UPDATE SQL Script* was selected. The user manipulated whether default SAS® variable names or the labels to those variables were the column names, and SQL commands were written to the .SQL file to assure that the function was fully successful.

View ORACLE Table provided the capability to view the records of a table in SASVIEW® display format. Similar in nature to the PROC COMPARE option, the column names were shortened to eight characters, and informats/formats equivalents to those of the table columns were selected. Veteran SAS® users often selected this display process instead of the regular display capability of ORACLE®.

The Macros

One major component of this SAS/AF® application was the EDITLOAD.SCL within which macros were created and submitted for processing using SUBMIT, SUBMIT CONTINUE, and ENDSUBMIT statements built around programs that:

- Created SYMPUT and read SYMGET values which were then “plugged” into DATA statements and PROC SQL language.
- Interactively submitted the SAS® modules and SQL programs for processing by SAS® and ORACLE®.
- Using the results of such processing, performed all the functions previously noted, especially those functions submitted by EXECUTE.

An excellent representation of these macros is the one which read SYMGET values and created the SQL that created new ORACLE® tables.

For purposes of demonstration, let us assume a SAS® data file, UREA.SD2, has already been created in the default SASUSER file and populated

with lab data. From the initial entry FRAME, we have a userid (SASMANN) and a password (SHOWMETHEMONEY) to allow access to ORACLE®. The new table will be named UREA_DATA. We want to perform the procedure in TEST mode. The table owner is PFI (this is not the logon userid), and it is necessary to update the dropdown lists.

The entry areas and entered values are:

SAS Data File: UREA

ORACLE Table Name: PFI.UREA_DATA

(Note the period in the table name.)

Mode: TEST

In the functions box, we select the radio dial beside *Update Dropdown Lists*. The SCL processes this request within both DOS and WINDOWS. At DOS prompts, we type “exit <return>” to finish the reading of DIR or PROC SQL procedures. The results are read into SAS® data files.

We read the SASLOG™ by selecting that option in a dialog box named *Code Processing Finished*, then return to the main FRAME.

We select the *Create ORACLE Table* radio dial. Using all the values previously noted, the following section of SCL is called to build a SAS® macro. We select *SAS Variable Labels* as the column names for the new table.

The actual SCL language and macro are displayed. SYMGET and other transferred values are shown in the macro in UPPERCASE.

The SCL

OCREATE:

method;

```
dsntab=symget('dbd');
orctabl=symget('dbt');
mode=symget('dbx');
user=symget('dbu');
pw=symget('dbp');
```

submit continue;

```
options source macrogen symbolgen ls=80;
run;
```

```
/* Create ORACLE DB table */
```

```
libname &user clear;
libname slibc clear;
libname &user 'c:\sasuser';
libname slibc 'c:\sas';
filename smacc
'c:\sas\sasmacro\&dsntab.sas';
```

```
proc dbload dbms=oracle data=&user.&dsntab;
  user=&user;
  password=&pw;
  table=SAS_LOAD;
  path="@&mode";
  limit=0;
  commit=0;
  list;
  load;
run;
quit;
```

```
data temp1;
  set &user.&dsntab;
run;
proc contents data=temp1 noprint position
out=temp2(keep=name label npos
  type);
run;
data temp3;
  set temp2;
  length tabconts tabname $ 40 npos 4;
  tabconts=label;
  tabname=name;
  drop label name;
run;
proc sort data=temp3 out=slibc.temp;
  by npos tabname tabconts;
run;
data _null_;
  file smacc notitle noprint;
  put @1 '%macro &dsntab;' /
      'proc sql noprint;' /
      'connect to oracle (path="@&mode"
user=&user orapw=&pw);' /
      'execute (create table &orctabl as
select';
run;
data _null_;
  do obsnum=1 to (last-1);
    length tablong $ 75;
    set slibc.temp point=obsnum nobs=last
end=eof;
    tablong=trim(tabname) || ' as ' ||
trim(tabconts); /* long columns */
    file smacc notitle noprint mod;
    put @1 tablong +(-1) ',';
```

```

end;
stop;
run;
data _null_;
  set slibc.temp end=eof;
  by npos tabname tabconts;
  tablong=trim(tabname) || ' as ' ||
trim(tabconts); /* long columns */
  if eof then do;
    file smacc notitle noprint mod;
    put @1 tablong +(-1) ' from
&user.SAS_LOAD) by oracle;' /
        'disconnect from oracle;' /
        'quit;' /
        '%mend &dsntab;';
  end;
run;
proc datasets library=slibc;
  delete temp;
quit; run;
proc datasets;
  delete temp1 temp2 temp3;
quit; run;

%&dsntab;

filename smacc clear; run;

proc sql noprint;
connect to oracle (path="@&mode" user=&user
orapw=&pw);
execute (drop table &user.SAS_LOAD) by
oracle;
disconnect from oracle;
quit; run;

endsubmit;

call display('continue.frame');

endmethod;

```

The Macro

```

%macro UREA;

proc sql noprint;

connect to oracle (path="@TEST"
user=SASMANN orapw=SHOWMETHEMONEY);

execute (create table PFI.UREA_DATA as
select
FIRSTVAR as FIRST_UREA_VARIABLE,
SECNDVAR as SECOND_UREA_VARIABLE,
THIRDVAR as THIRD_UREA_VARIABLE,
from SASMANN.SAS_LOAD) by oracle;

```

```
disconnect from oracle;
```

```
quit;
```

```
%mend UREA;
```

The reader can see that the PROC DBLOAD reads values from the SAS® data file into the temporary table named SASMANN.SAS_LOAD using default variable names.

With these variable names in hand, the macro selects each variable as the variable label for loading into the ORACLE® table PFI.UREA_DATA.

Once the creation process has successfully ended, the SASMANN.SAS_LOAD table is dropped. This makes sure the next time this table is needed, the correct variable and values will be used; otherwise, it might be possible for the insertion of incorrect variables/values into a table.

Conclusion

Since the previously-mentioned download includes all sources for the FRAMEs and SCL, it is not necessary to provided additional examples about this SAS/AF® application. Once FRAMEs are matched with their sources, much of the language and usage will be self-explanatory.

Although this application may be used for many database products such as DB2, FOXPRO, etc., there are great possibilities that the actual SQL statements included in the SCL will require modification. There are no copyrights on any of the files, catalogs, etc., included in the download .EXE file; therefore, the user may alter the code as needed. I do request that if you pass on this code to others, you provide the appropriate attribution.

About the Author

Paul Ehresmann is a private contractor who is a SAS® Quality Partner™ and SAS® master consultant. He has specialized in SAS® programming, statistical analysis, and applications development for over 20 years within various government and commercial industries. He has undergraduate and graduate degrees from Wittenberg University in Ohio and Indiana University.



References

SAS Institute Inc., (1989), *SAS/AF® Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc., (1994), *SAS® Screen Control Language: Reference, Version 6, Second Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc., (1994), *SAS/ACCESS® Software for Relational Databases: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.