

SAS® and Oracle® PL/SQL™: Partners or Competitors?

Stanley Fogleman, Harvard Clinical Research Institute, Boston, MA

ABSTRACT

Most computer programmers tend to think in binary fashion. Either something is completely possible and should always be used or something is impossible and thus should never be considered. The question that should be asked is: How can the tasks be divided between SAS® and ORACLE® to play to their relative strengths? The paper will concentrate on the ways in which PL/SQL is very useful to SAS programmers (in the form of stored functions and procedures) and can be accessed with either SAS/ACCESS FOR ODBC® or SAS/ACCESS FOR ORACLE®.

INTRODUCTION

PL/SQL is an extension to SQL. It was developed to eliminate the need for utilizing a higher-level language such as C to make database calls in a procedural fashion. It allows both data manipulation and query statements of SQL to be included within procedural units of code¹. During processing, PL/SQL code is separated from SQL statements and the PL/SQL engine in the Oracle server processes them.

Six main benefits of PL/SQL are:

- 1) Portability among all Oracle platforms
- 2) Declaration of variables
- 3) Procedural language control structures (loops, conditional execution, error handling)
- 4) Easy maintenance
- 5) Improved data security and integrity
- 6) Improved performance

WHAT SAS AND ORACLE DO BEST

SAS is best at expressing complex data processing problems in a concise fashion. ORACLE PL/SQL is best at "encapsulating" knowledge about the database that would be difficult at best to determine from "outside" the database. In addition, PL/SQL upon compilation is stored as a binary executable which makes retrieval extremely fast. I propose to use PL/SQL procedures and functions as "helper objects" to SAS to help extend the functionality of PROC SQL.

SOUNDS NICE...WHY SHOULD I BELIEVE YOU?

With regards to portability, PL/SQL will run without modification on any supported ORACLE platform. Variables can be instantiated using the %TYPE declaration which refers to the data type stored on the database - the advantage being that if the column definition changes the code will use the new definition the next time it executes. Control structures allow you to code for complex conditions ordinarily not accessible from native SQL code. Maintenance is easy because code can be changed "off-line" without affecting other users. Validation should be made easier by breaking the code into well-defined objects which facilitates the proof of input producing desired output. Database security is enhanced by control of objects to the particular user. Performance is improved by compilation ahead of time as opposed to parsing at run time which in return reduces the number of calls to the database.

TYPES OF DATABASE OBJECTS

- 1) Functions
- 2) Stored Procedures
- 3) Database triggers
- 4) Packages.

This paper will concentrate on the first two objects, because while it is technically possible to invoke the last two from PROC SQL, the need to do so would be extremely rare.

WHY WOULD YOU WANT TO CALL A STORED PROCEDURE OR FUNCTION?

Generally speaking, the Data Base Administrator will have set up a library of functions and procedures for use with the database you are using. It may be possible to utilize some of those to return information that would otherwise require a detailed knowledge of the database schema. It also increases the efficiency of the query when used in the WHERE clause to filter the data as opposed to retrieving the data into the application and filtering there. Depending on the database staff's expertise, it may not be too difficult to have them write a few procedures or functions to perform simple tasks, particularly if you can supply them with well-defined requirements. Procedures and functions are an extension of SQL, and provide the ability to perform tasks that would be convoluted, if not impossible, with SQL by itself.

CREATING A SERIES OF FORMATS FROM A CLINICAL TRIALS DATABASE

For example, recently I was preparing to create a series of formats from a Clinical Trials Database. It was necessary to know in advance whether the format applied to a numeric or character variable. Since numeric variables

outnumbered character variables by a wide margin, it was decided to return a zero(0) for a numeric field and a one(1) for a character field. This allowed me the flexibility to signal to PROC FORMAT in advance which type of format was being created, thus avoiding errors and reruns.

The ORACLE SQL code was as simple as:

```
Select codetype(codename) as codetype from dual;
```

Note: codetype is the function, codename is the column name used as a calling argument to the function. dual is a special table on ORACLE that contains no rows, which makes it extremely handy for calling functions and procedures.

USE OF PROCEDURES FOR STANDARDIZED REPORTS

Another example would be the use of procedures for a set of standardized reports, where parameters would drive the selection criteria. The advantage would be that the procedure could be validated and reused as a standalone object, where a typical report process involves selection, calculation, formatting and printing. This enforces modularity which is a key underlying component of good system design. If a particular component of a modularized system fails, it should be easier to diagnose and solve the problem. Of necessity, a stored procedure or function "has knowledge of" the schema it is querying, so error messages you get in the course of writing the macro should be much more instructive and meaningful than the somewhat general error message from SAS PROC SQL "ERROR IN SQL STATEMENT" - this is not a good start to a Monday morning. Also, a stored procedure has already been optimized, so the rows should be returned much faster.

RETRIEVING FIELDS STORED AS TEXT

Yet a third example was a clinical trials database where a particular trial had the case report form page number stored as a decimal (1.1, 1.2. etc.) as a text field. The pass through query that had been written was failing to retrieve all of the relevant pages. The solution was to use the ORACLE "TO_NUMBER" function to convert the page number to a number on the ORACLE side before comparing it to the pages I was seeking. So the resulting SQL read something like:

```
Select * from  
DEMOG where TO_NUMBER(PAGENO) = 1.5;
```

Again, this was relatively simple to accomplish on the ORACLE side - the alternative of diagnosing exactly how the where clause was being parsed gave me nightmares.

CO-EXISTENCE OF PL/SQL AND SAS

A recent posting to SAS-L by Philip Crane illustrated how easily SAS and PL/SQL coexist. Someone wanted to know how to avoid hard coding the dates for a report that had to be run on the first of every month to show monthly figures. A combination of SAS functions, macro variables, and ORACLE system provided functions do the trick nicely. (Note: ORACLE supplied functions are indeed written in PL/SQL).

To get the dates to load the macro variables, a data step is run:

```
1 data _null_;  
2   strt_dt = intnx("month", today(), 0, "b");  
3   end_dt  = intnx("month", today(), 0, "e");  
4   call symput("strt_dt",put(strt_dt,date9.));  
5   call symput("end_dt", put(end_dt, date9.));  
6   call symput("strt_ym",put(strt_dt,yyymm6.));  
7 run;  
8  
9 %put &strt_dt &end_dt &strt_ym;  
10
```

Then a PROC SQL is run using a combination of SAS macro variables and ORACLE supplied functions:

```
11 proc sql;  
12 connect to oracle (.....);  
13   create table table&strt_ym as  
14     select * from connection to oracle  
15       (select id, name  
16         from mastertable  
17         where trunc(date) between  
18           between to_date("&strt_dt",'ddmonyyyy')  
19             and to_date("&end_dt" ,'ddmonyyyy')  
20       );  
21 disconnect from oracle;  
22 quit;
```

This approach utilizes the relative strengths of each language nicely. SAS certainly has a greater quantity and quality of date functions. Calculating the first and last days of the current month is certainly possible in PL/SQL, but not at all straightforward. Likewise, although it is easy to format a SAS date, it is somewhat more difficult to pass that value to ORACLE in a format that it understands (recall that ORACLE dates are analogous to SAS datetimes). The TO_DATE function takes a string value (in this case a SAS macro variable) and converts it to an ORACLE date value. This approach also has the advantage of making coding errors obvious and easy to detect.

WHICH ORACLE DATABASE OBJECTS ARE MOST USEFUL?

There are two database objects on ORACLE that are most useful to a SAS programmer, and those are the stored function and the stored procedure. Generally, you use a procedure to perform an action and a function to compute a value. Note: depending on your local environment, your Oracle DBA may have chosen to store a collection of procedures and functions in an object called a package. The syntax, while not terribly difficult, is outside the scope of this paper. Likewise, on very rare occasions you might need to execute a database trigger (perform some action after an update to a database). This is likewise outside the scope of this paper.

WHAT IS A STORED FUNCTION?

A stored function takes one or more arguments and must return a single value. A function can be stored in the database as a schema object for repeated execution. A function is called as part of an expression.

WHAT IS A STORED PROCEDURE?

A stored procedure is a named program that executes some predefined statements and then returns control to whatever called it. After creating a procedure, you can invoke it by name from other programs.² A procedure can take three types of parameter, aptly named (IN, OUT and IN OUT). The latter passes a value from the calling environment into the procedure and a possibly different value from the procedure back to the calling program).

Example of stored procedure:

```
create or replace procedure MY_PROCEDURE (NEW_EMPLOYEE IN varchar2)
AS
    BEGIN
        insert into EMPLOYEES
            (name, job_title, phone_num)
        values
            (NEW_EMPLOYEE, null, null);
    END;
```

And this is the code to execute:

```
SQL> execute my_procedure ('STANLEY FOGLEMAN');
```

So, what it does is it inserts a new record into the EMPLOYEES table with the values as following:

Column "NAME" will get value "STANLEY FOGLEMAN"

Column "JOB_TITLE" will get null value

Column "PHONE_NUM" will get null value

"NEW_EMPLOYEE" is the IN parameter declared for the procedure.

Note: the user executing the procedure has to have an "INSERT" privilege to the table "EMPLOYEES".

HOW DO I FIND OUT WHAT PROCEDURES AND FUNCTIONS ARE AVAILABLE TO ME?

A simple PROC SQL query will tell you which objects are available to you. Bear in mind that you need to be granted execute access to be able to utilize these.

```
Proc sql noprint;
connect to oracle (.....);
Create table all_procedures_functions as
Select * from connection to oracle
(select object_name, object_type from user_objects where object_type in
('PROCEDURE', 'FUNCTION'))
```

```
Disconnect from oracle;  
quit;
```

Note: user_objects will only give you the objects that "belong" to the schema you are connected to the database with.

All_objects table will let you see all of the objects in the database. You would need read access permission to all_objects to

find out what is available and executer permission for a procedure or function if you want to use it.

Of course, you will still need the advice of your local DBA as to which functions and procedures would be most pertinent.

WHAT ABOUT THE DIFFERENCE BETWEEN ORACLE STATIC AND DYNAMIC SQL?

Static SQL is known at compile time - the compiler "knows" the data types, inputs and output. Additionally, the security constraints can be checked at compile time. With dynamic SQL, none of the above is known to the compiler and must be determined at run time. The trade-off is that static code is easier to code and maintain, but the dynamic code can be used more generically. For instance, if you were to code a "rowcount" utility statically, you would need to "hard-code" a table name - not very useful - with a dynamic call, you could specify the table name at program execution time.³

WHAT ABOUT DYNAMIC CODE?

Suppose you are lucky enough to have specifications in a "normalized" format, such as an excel spreadsheet. Using SAS to generate either a SAS program or a PL/SQL program is relatively straightforward, in part due to the simplicity of the FILE filename statement. Using PL/SQL to generate a SAS program or another PL/SQL program is not straightforward, because creating an external flat file requires calling a system procedure. However, it is important to note that in PL/SQL the DMBS_SQL package provides the ability to perform DDL (data definition language) statements e.g. (DROP TABLE, CREATE TABLE) and to create DML (data manipulation language) routines e.g., (INSERT ,UPDATES, DELETES) and DCL statements(GRANT, REVOKE) that are not available in static PL/SQL routines.⁴

CONCLUSION

ORACLE Stored Procedures and functions are an invaluable addition to any SAS programmers skill set and require no additional licenses to utilize. Dividing the functionality between SAS code and PL/SQL code helps to enforce modularity. By modularizing a program, you accomplish two things: 1)you make the program easier to modify and maintain 2)you increase the efficiency of the program by reducing the number of rows returned in some cases with preliminary processing already performed.

REFERENCES

- 1) Pataballa and Nathan, Oracle 9i - Programming with PL/SQL- Student Guide (Redwood City:Oracle,2001), p.1-3.
- 2) Pribyl and Feuerstein, Learning Oracle PL/SQL (Sebastopol: O'Reilly, 2002) p.29.
- 3) Kyte, Thomas. Website:asktom.oracle.com (faq section).
- 4) Trezzo, Brown, Niemiec, Oracle PL/SQL Tips and Techniques (Berkeley:Osborne, 1999) pp.615-616.

ACKNOWLEDGMENTS

Thanks to Michael L. Davis for suggesting this paper and Samuel Kasparov for reviewing the paper and making suggestions and providing the stored procedure example.

CONTACT INFORMATION

(In case a reader wants to get in touch with you, please put your contact information at the end of the paper.)

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

Stanley Fogleman
Harvard Clinical Research Institute
930 Commonwealth Ave West
Boston MA 02215
Web: <http://www.hcri.harvard.edu>



SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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