STRATEGIC USES OF SAS® DATA STEP PROGRAMMING AND SQL PASSTHROUGH TO QUERY ORACLE® DATABASES

Henry W. Buffum, Signal Corporation

Introduction:

The SAS/Access product offers SAS users two ways to easily access data stored in Oracle databases. PROC ACCESS can be used to create views of Oracle tables which can then be accessed just like SAS datasets. This allows SAS data step programming techniques to be used to merge the Oracle data with other SAS datasets or perform subsetting and recoding tasks as needed. Or, SQL Passthrough can be used to create views of Oracle tables, which can then be read by SAS. This technique passes SQL query commands to the Oracle server, and takes advantage of the powerful Oracle RDBMS engine to perform joins, sorts, and data subsetting tasks. This technique requires a good understanding of SQL and of client-server processing, and can be much more efficient than using PROC ACCESS.

With these options offered by the SAS/Access product, the SAS programmer is faced with some questions. Which is the easiest way to deal with data in Oracle tables? Which is the most efficient, in terms of computer resources? Should SAS data step programming be abandoned in favor of SQL? This paper examines four different techniques for solving a problem involving a query of Oracle an table from SAS, and discusses their relative merits in terms of computer efficiency and ease of programming.

The code examples presented in this paper were developed to solve one problem: how to find taxonomic information in a large (250,000+ record) Oracle table for a small number of taxonomic names contained in a SAS dataset. My first program performed this task by using a PROC ACCESS view of the Oracle table, and SAS data steps to perform the merging and subsetting tasks. This program returns the correct results in about 10 minutes. My next three example programs were attempts to improve the response time. The final program only takes about one minute.

These programs were written for SAS 6.08, running under VMS. The Oracle database resides on a DEC Alpha, running OS/F. The communication between the two platforms is handled by Oracle SQL*Net for TCP/IP. While it is difficult to obtain precise benchmarking statistics in a client/server environment, it is clear from my tests that I gained significant improvements in response time by rewriting this query to maximize the use of the Oracle RDBMS engine. These same improvements in response time might not be duplicated across different computing environments and applications, but knowledge of these techniques should prove useful to SAS programmers who need to access data in Oracle databases.

The Problem: querying a taxonomic database.

The purpose of the sample programs presented in this paper is to extract current taxonomic information from an Oracle database. The input SAS dataset contains a list of benthic organisms identified by Latin name. An Oracle table exists which contains useful, up-to-date taxonomic data for hundreds of thousands of such organisms. Along with other information, this Oracle table contains alternate names that have historically been used for organisms that have been given new taxonomic names. The SAS dataset used as input for these programs may contain the older, obsolete names of some organisms. These programs were written to check the Oracle table for the most current taxonomic name for organisms that are referred to with an obsolete name in the SAS dataset. The programs must run in batch and be able to handle any number of taxonomic names at once.
The Oracle table EMAP.TAX_LATIN_NAMES contains the following attributes (as well as many others):

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LATIN_NAME</td>
<td>VARCHAR2(78)</td>
</tr>
<tr>
<td>TID</td>
<td>NUMBER(9)</td>
</tr>
<tr>
<td>SYN</td>
<td>VARCHAR2(1)</td>
</tr>
<tr>
<td>CURRENT_TID</td>
<td>NUMBER(9)</td>
</tr>
</tbody>
</table>

The LATIN_NAME attribute contains the taxonomic name for a specific taxon; TID contains the taxonomic identification number, the SYN attribute contains a flag indicating whether the taxonomic name has a synonymous (more current) taxonomic name (a value of 'S' indicates a synonym), and CURRENT_TID contains the TID for a synonymous taxonomic name if one exists.

The following data step was used to create a test input data set TAXNAMES, which is used as input for the SAS programs described in this paper.

```sas
data taxnames;
input name $ 1-30;
sciname=upper(name);
cards;
Phyllodoce maculata
Sabella microphthalma
Scoletoma acicularum
Syllides verrilli
;
```

The SAS data set here contains the scientific names of only four benthic organisms. My programs must run regardless of how many observations are in TAXNAMES.

The programming task is to find matching records in the Oracle table for each name in the SAS data set and check the SYN attribute to see if a more current name exists. If one does, the CURRENT_TID attribute must be used to locate another record in the same Oracle table and save the most current taxonomic name. Only those taxa that do have a more current name listed in the Oracle table are to be reported on.

Solution #1: Access Views and data step programming.

The sample programs presented here use macros to provide the parameters needed by SAS/Access to make the connection to Oracle. This eliminates the need to enter these parameters over and over throughout the programs. The macro ORA_PARMS is used with PROC ACCESS and PROC DBLOAD. The macro SQL_CON is used with SQL Passthrough.

```sas
%macro oraParms;
user= HBUFF;
orapw=* *** ;
path= '@t:*****';
%mend;

%macro sql_con;
user= HBUFF
orapw=********
path= '@t:*****'
%mend;
```

My first solution to the programming problem used PROC ACCESS to create a view of the Oracle taxonomy table, and then two SAS data steps using MERGE and IF statements to extract the necessary data. This solution has the virtue of being easy for an experienced SAS programmer to understand and code. It took very little time to write.

The first section of the program uses PROC ACCESS to create an Access descriptor and a View descriptor for the Oracle table EMAP.TAX_LATIN_NAMES.

```sas
proc access dbms=oracle;
create tmp.tax.access;
%oraParms
table=emap.tax_latin_names;
assign=yes;
create tmp.taxv.view;
select all;
```

The ASSIGN=YES statement tells SAS to assign new names to Oracle fields that have names longer than 8 characters--this will result in the names of the fields LATIN_NAME and CURRENT_TID being shortened to their first 8 characters. Note--alternate syntax allows the SAS program to explicitly assign new variable
names. The view descriptor named TAXV (in a library name TMP) can now be accessed by SAS data steps, and points to the data in the Oracle table.

Next, data steps merge the Oracle data in the view TMP.TAXV with the SAS data set TAXNAMES, select the correct data, and get the synonym information:

data step1;
merge TAXNAMES(in=a)
   TMP.TAXV(rename=(latin_na=sciname
    current_=curr_id));
   by sciname;
   if a and syn="S";
   tid=curr_id;
proc sort; by tid;
data taxall; /*find current names */
merge step1(in=a)
   TMP.TAXV(in=b);
by tid;
if a and b;
curname=sciname;
proc print; var name syn currname;

Note that this program makes use of the Oracle RDBMS's ability to sort data quickly. It is not necessary to sort the data defined by the Access View TMP.TAXV in the SAS environment, even though the SAS code requires the data to appear in two different sort orders. The data are sorted by Oracle—simply accessing the view with a different BY statement causes Oracle to sort the data correctly.

While this program was relatively easy to code, it turned out to be prohibitively time-consuming at execution, taking as much as ten minutes to run. This solution to the problem does not use the Oracle RDBMS or the client-server configuration very efficiently. This program requires two passes through the Oracle table of 250,000+ records to find the information for four taxonomic names. Each pass through the table requires Oracle to send all the data across SQL*Net to the client (SAS) environment. A more efficient solution would have more processing performed by the Oracle RDBMS, and less data sent across the network to the SAS environment.

Solution # 2: SQL Passthrough and SAS SQL

In my first attempt to improve the response time, I re-wrote the program using PROC SQL and the SQL Passthrough facility. SQL Passthrough allows portions of SQL code to be passed to the Oracle server for processing. This results in some of the processing that would otherwise be performed by SAS on the client machine to be performed by Oracle on the server.

proc sql;
connect to oracle as db
   (%sql_con) ;
create view lax1 as
   select *
   from connection to db
   {Following SQL code is passed to Oracle}
   (select a.latin_name as name,
    b.latin_name as curname
    from emap.tax_latin_names a,
    emap.tax_latin_names b
    where a.syn='S' and
    a.current_tid = b.tid);
disconnect from db;
create view tax2 as
   select * from tax1
   where name in
      (select sciname from taxnames);
proc print data=tax2;

This program passes some SQL code to the Oracle server, and uses SAS SQL processing to perform the final query. The PROC SQL statements create two views: TAX1 and TAX2. The first is created using the SQL Passthrough facility—the SQL statements in parentheses between the CONNECTION TO DB and the DISCONNECT FROM DB tags are sent to Oracle for processing. The TAX2 view is processed entirely by SAS.

This program makes better use of the Oracle RDBMS. The SQL code processed by Oracle
joins the TAX_LATIN_NAMES table to itself to find synonyms, and only processes those records that have synonyms. The Oracle RDBMS performs this kind of query very efficiently. Having Oracle process this code reduces the amount of data that must be passed over the network to the SAS environment—only the records which have synonyms are sent. This program executed in about 5 minutes—half the time of my first program. However, this improved response time was still not adequate.

Solution #3: SQL Passthrough and the Dbload procedure.

It became clear to me that the way to improve the efficiency of this program was to reduce the amount of data that Oracle had to process and send over the network to SAS. If it were possible to send Oracle a WHERE clause that would cause it to process only those taxonomic names which I needed, it could do the job very quickly. I experimented with hard-coding the taxonomic names into the SQL code passed to Oracle and found this to be true. I needed a way to pass the taxonomic names in my data set to the Oracle automatically. Unfortunately, the Oracle server can not access data in SAS datasets. However, it is possible to copy a SAS dataset to an Oracle table, using PROC DBlOAD.

I have found that PROC DBlOAD is generally not the best way to create permanent Oracle tables (Buffum and Keith, 1994). This SAS procedure creates Oracle tables, but sets certain parameters (particularly the storage parameters) to default values that may not be appropriate in a permanent database. However, this procedure is very useful for creating a temporary Oracle table from a SAS data set that can be used to process a SQL query.

To use PROC DBlOAD, the SAS programmer must have CREATE TABLE and DROP TABLE privileges for the Oracle database. The following code first uses PROC DBlOAD to drop any pre-existing version of the table TEMP, and then creates a new table TEMP by copying the SAS data set TAXNAMES to a table of that name.

```sql
/* clear table name in Oracle */
proc dbload dbms=oracle;
  %ore_parms;
  sql DROP TABLE TEMP;
/* send names to Oracle table */
proc dbload dbms=oracle data=taxnames;
  %ore_parms;
  table temp;
  load;
  run;
```

Next, SQL Passthrough is used to create a view of the taxonomy database. The SQL code is very similar to that used in solution #2. However, a WHERE clause is used to select only the taxonomic names which are found in the table TEMP, thus significantly reducing the amount of data processed by Oracle.

```sql
proc sql;
   connect to oracle as db (%sql_con);
   create view tax as
   select *
   from connection to db
   (select a.latin_name as name,
    b.latin_name as curname
    from emap.tax_latin_names a,
    emap.tax_latin_names b
    where a.current_tid=b.tid and
    a.latin_name in
    (select rtrim(sciname)
    from TEMP)
    order by a.latin_name);
   disconnect from db;
```

This program uses the client-server configuration very efficiently. The Oracle RDBMS processes the SQL query and passes the least amount of data possible over the network to the SAS environment. This program ran in under one minute.

Solution #4: SQL Passthrough and SAS Macro language programming.

The fourth and last solution to this programming
problem uses SQL Passthrough in a manner very similar to the previous program. A WHERE clause is passed to the Oracle RDBMS to select only the records required. However, this program does not require using PROC DBLOAD. Therefore, the SAS user does not need to have CREATE TABLE or DROP TABLE privileges for the Oracle database for this program to run. Instead, it uses the SAS Macro language to pass all of the taxonomic names in the SAS data set to Oracle.

The program utilizes a commonly used macro language technique to load the taxonomic names into an array of macro variables. (Michel and Michel, 1994; Whitlock, 1994; Klenz, 1991) Next, the macro NAMES is defined, which can then retrieve these names for the SQL Passthrough code that will be executed by Oracle.

```
data _null_; set taxnames; call symput("VAR"],[left(_N_), "trim(sciname)])"," run; %macro names; %do i=1 %10 &vcount; &var&j, /* comma needed for SQL */ %end; /* write final dummy name with no comma */ 'dummy' %mend; /* Next, SQL Passthrough is code is used to */ /* process the query */ proc sql; connect to oracle as db (%sqL_con); create view tax as select * from connection to db (select _latin_name as name, b._latin_name as curname from emap.tax_latin_names a, emap.tax_latin_names b where a.current_tid=b.tid and a._latin_name in (%names )); disconnect from db;```

This SQL code is very similar to that used in the previous solutions. However, in this case the WHERE clause that selects the needed records invokes the macro NAMES to supply the taxonomic names needed.

This program uses the client-server configuration just as efficiently as the previous example, and takes about the same amount of time to run. Its real advantage is that it does not require creating and dropping tables in the Oracle database.

**Conclusion**

SAS programmers who must access data in an Oracle database should have a variety of techniques at their disposal. A good knowledge of the SAS/Access product, SQL programming, data step programming, and macro language programming are all important. For many applications, the simplest and most familiar techniques will get the job done. But applications that require the most efficient use of computing resources call for more sophisticated techniques. Using the SQL Passthrough facility to allow the Oracle server to process queries efficiently is essential in these cases.

**References**


Trademarks

SAS, SAS/AF and SAS/ACCESS are registered trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Oracle is a registered trademark of Oracle Corporation.

Contact Author:

Harry Buffum
Signal Corporation
27 Tarzwell Drive
Narragansett, RI 02882

(401) 782-3183