

Accessing Oracle Tables Using Base SAS/SAS Access

David Thul, Blue Cross Blue Shield of Minnesota

INTRODUCTION:

One of the things that continues to impress me about The SAS System (SAS) is its ability to access so many different data sources so easily. It is true that SAS has not met a data source it doesn't like. In my twelve years of programming in SAS, I have hit against at least a dozen different data sources. I must admit I don't like them equally. One of my favorite sources is Oracle. One reason I feel this way is that the link between Oracle and SAS is so flexible and versatile. At this point, I must point out that while SAS Access is necessary to link with an Oracle database, the intricacies of that product are beyond the scope of this paper. It is assumed that Access is installed at your location. I will be presenting some examples of how to use SAS/Access in your Base SAS programs but I will not go into any detail about the Access product itself.

Note: While this paper uses Oracle as a default example, these methodologies will work on most RDBMS platforms.

USING SAS TO READ ORACLE DATA.

There are two methodologies that SAS has implemented for you to read data from Oracle tables. They are Explicit and Implicit pass-through.

EXPLICIT PASS-THROUGH:

This method is very commonly used on the IBM mainframe to read against DB2 databases. It requires you to use Proc SQL to connect to the database. For those of you who are familiar with this, the transition to reading Oracle data is quite simple. The DB2 code looks like this:

```
Proc SQL;
  Connect to DB2 (SSID=DB2Q);
  Create table sastable
  As select * from connection to DB2
  (Select col1, col2, col3
   from db2table
   where conditions
   [group by]
   [ order by]);
  Disconnect from DB2;
Quit;
```

Typically, your bind and password information for this connection will be found in the JCL of your job.

The code for Oracle is almost identical:

```
Proc SQL;
  Connect to Oracle
  (user=clm_user password=pw path="PHID" schema="clm_owner");
  Create table sastable
  As select * from connection to oracle
  (Select col1, col2, col3
   from oracletable
   where conditions
   [group by]
   [ order by]);
  Disconnect from Oracle;
Quit;
```

Notice that the connection information (user, password, etc.) is found within the proc and does not rely on JCL to bind. Typically, with an Oracle connection, you are working on a distributed environment (ie. UNIX) so there is no JCL.

One of the advantages of Explicit Pass-through is that once you are within the parentheses, you are actually utilizing the native RDBMS's sql. I've found that I like some of the functions that Oracle provides and this allows you to use them. The problem is that there are times where I may want to incorporate some SAS functions into my resultset as well. In that case, I can put the SAS function within the outer select (outside of the parentheses) like this:

```
Beginning of proc sql
Create table sastable
As select sasfunction(column derived within parantheses) as newfield,
* from connection to oracle
(RDBMS sql);
Disconnect from Oracle;
Quit;
```

The resultset created by this select statement will be the same as the one above except a new field, newfield, will be added that was created using SAS functionality.

IMPLICIT PASS-THROUGH (LIBNAME):

While the above examples show flexibility in terms of being able to use SAS functions as well as the native RDBMS's functions, the explicit pass-through methodology is limited to proc sql. Since we are using SAS, wouldn't it be nice to be able to use the tried and true data step and procs that we have all come to know and love? The good news is that we can. If you put the following statement into your program, you will be able to "trick" SAS into thinking that your RDBMS' table is a SAS file. Let me explain by example:

```
libname orcl oracle user=clm_user password=pw path="PHID" schema="clm_owner";
```

This works like a libname statement you have probably used in the past to point to SAS data. Notice the addition of the keyword "oracle." That is what tells SAS that you will be reading an Oracle file. The keywords after the keyword "oracle" correspond to the connection statement embedded in the explicit pass-through example above. The difference is that this connection lasts for the duration of your SAS program instead of terminating at the end of the proc SQL. Now you can utilize the tried and true SAS procs to manipulate an Oracle table. The disadvantage to this methodology is that you do not have the ability to use the native RDBMS' SQL.

To emulate the explicit pass-through example above with a data step, do the following:

```
Data Sastable;
Set orcl.table (keep=col1 col2 col3);
Where conditions;
Run;

Proc sort data=Sastable;
Run;
```

As you can see, a lack of SQL skills does not stop you from accessing Oracle tables via SAS. Use your imagination and you will find many uses for this powerful tool.

CAUTION:

This is a good time to introduce a caution. When the SAS compiler finds an implicit pass-through, it has to convert the SQL or SAS code to the native RDBMS's SQL. This happens behind-the-scenes but does take something from the efficiency of the program. This efficiency trade-off is typically quite small and worth the advantages of this methodology. However, if you are using SAS functions in a where clause that don't exist in the native RDBMS (like datepart, for example), SAS will have no way to know how to translate the command to the native SQL. When this happens, the where clause is ignored resulting in possible unexpected results and full table scans.

USING SAS TO CREATE ORACLE DATA.

Another advantage to the implicit pass-through methodology is that, with the proper authority you can create Oracle tables from SAS as well. You could just as easily reverse the above example to:

```
Data orcl.table;
Set sastable (keep=col1 col2 col3);
Where conditions;
Run;
```

You could then sort if you choose.

CAUTION:

As well as allowing you to sort an RDBMS table you may have just created, SAS will allow you to sort the source data too. That data is most likely used by a broader audience who may be expecting it to be in a certain sort order. Fortunately, your DBA will probably have database-level protection to prevent you from doing this. Even so, you will

likely get an error from SAS so if you are going to introduce source data to your SAS program via a sort, it is best to use the out= operator to create another table.

```
Proc sort data=orcl.table out=sortedorcl;  
By coll;  
Run;
```

CONCLUSION:

Whether you are a seasoned SAS professional who would like to work with Oracle or other RDBMS data but don't want to learn SQL or you are seasoned at both, SAS/Access will allow you to work with the data in a very seamless and simple manner.

CONTACT INFORMATION

David Thul
Blue Cross/Blue Shield of Minnesota
(651) 662-4901
dthul@attbi.com

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.