Access to Relational Databases Using SAS®
Frederick Pratter, Computer Science/Multimedia Studies Program, Eastern Oregon University, La Grande OR

ABSTRACT
SAS® software currently provides many of the features of a database management system, including database views and an extended superset of ANSI SQL. However, it is often impractical or just plain impossible to convert desktop or organizational databases into SAS. Consequently, SAS software provides several procedures for access to relational databases. This paper will review how to use the various SAS/ACCESS® products for linking networked workstations to remote servers. Most of the examples will use the SAS/ACCESS for Oracle, but the principles described apply equally to local databases in Microsoft Access, as well as other client/server systems such as DB2 and MySQL.

INTRODUCTION: CLIENT/SERVER DATABASE MANAGEMENT SYSTEMS
In order to understand the various SAS/ACCESS options, it is important to recognize that it was written specifically to run on client/server database systems, in which a separate database engine supplies data to the local application. In this paper, all of the examples use SAS software as the client and a relational DBMS as the server. (Since SAS was originally written to run only on the mainframe platform, it did not initially provide a separate server engine. The SAS/CONNECT®, SAS/SHARE® and most recently SAS® Integration Technologies products have been introduced to support this functionality.)

A relational DBMS (Database Management System) such as Oracle runs as a network-centric application that provides efficient database access. This efficiency results because it is not necessary to copy the entire database each time a set of records is selected. The server engine selects the desired records and only these are sent over the network to the client. The database server is generally not on same platform as the application although in principle it is possible to have the server run locally on the same machine as SAS.

As the SAS documentation indicates, before you can use SAS/ACCESS to Oracle software, the following are required, in addition to Base SAS software and the SAS/ACCESS Interface to Oracle software:
- Oracle Server (Release 8.1.7 or later)
- Oracle Client (Release 8.1.7 or later).

For the examples in this paper, Oracle 11 Server was installed on a network host running 64-bit Linux. The host name of this server is thor; since this system is running in a test network it is not necessary to supply the fully qualified domain name thor.eou.edu. The Oracle Client was installed on a Windows system running Microsoft XP Professional SP2; SAS 9.1.3 software was installed on this local Windows machine. This is a fairly typical setup, where SAS software is running on the client and connects remotely to the database server.

The most important thing, from the end-user standpoint, is that the Oracle Client must be set up and correctly configured on the local machine running SAS. You will need to contact your database administrator to determine the requirements for your particular environment. Oracle supplies a set of Configuration and Migration Tools to make this a little easier. The Oracle client software is available at no cost for educational and research purposes, but your site almost certainly has restrictions on what you are allowed to install on your desktop system. It may be that your site does things differently from the examples in this paper. Your IT support person should absolutely be consulted before attempting any of these activities, and certainly before attempting to access organizational resources.

The Oracle Net Manager is used to provide the Service Name and port on which the client will connect to the server. Unless this is done, SAS/ACCESS will not be able to find your server. In this case, the Service Name is the same as the Oracle SID. Ask your database administrator for the correct values at your site.

Here is an example of what the Oracle Net Manager configuration looks like on Windows:
You can test that the Oracle client is correctly installed on your system by opening Oracle SQL Plus from the Application Development section of the Windows Start>Programs>Oracle menu. The standard Oracle sample database comes with a few pre-loaded tables, owned by user “scott”. These are used for the examples in this paper, but of course you will have your own user name, database SID, and table names. If you are not sure, again, please talk to your database administrator to get the correct values you need for your site.
This sample illustrates that the user “scott” is signing on to the remote service “sample”. Four pre-loaded tables are available, with dummy data that can be used to test your code. If you see something like this, your Oracle client is set up properly and you can connect to the server. Now you are ready to try SAS/ACCESS.

**OVERVIEW: ACCESS TO RELATION DATABASE MANAGEMENT SYSTEMS (RDBMS)**

The complexity of the SAS/Access® software results from the necessity of sending database commands from the client to be executed by the server. The product is available in three different forms, depending on the platform. As the following table illustrates, there are more options available for Windows than for systems running UNIX operating systems.

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Available Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unix</td>
<td>Unix</td>
<td>SAS/Access to &lt;RDBMS&gt;</td>
</tr>
<tr>
<td>Windows</td>
<td>Unix</td>
<td>SAS/Access to &lt;RDBMS&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS/Access to ODBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS/Access to OLEDB</td>
</tr>
<tr>
<td>Windows</td>
<td>Windows</td>
<td>SAS/Access to &lt;RDBMS&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS/Access to ODBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS/Access to OLEDB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAS/Access to PC File Formats</td>
</tr>
</tbody>
</table>

Table 1. SAS/ACCESS Interfaces by Platform

The only option available for a user at a UNIX client workstation is to use one of the database specific products. Each of these interfaces includes four procedures:

- PROC ACCESS – Used to download DBMS data into SAS
- PROC DBLOAD – Load SAS data into DBMS
- PROC CV2VIEW – Convert SAS/ACCESS views to SAS SQL views
- PROC SQL Pass-Through – Execute native SQL (Structured Query Language)
The ACCESS procedure (as distinct from the SAS/ACCESS product) as originally designed was extremely cumbersome to use. I once described it as “probably the worst designed SAS product of the decade” (Pratt, 1993). This was because in Version 6 of the SAS System® it was necessary first to create an Access descriptor to describe the data in a single DBMS table and then create a second View descriptor to define a subset of the DBMS data described by the Access descriptor.

The following program shows how this works for a remote Oracle database table:

```sas
proc access dbms=oracle;
create work.test.access;
user="scott";
orapw="tiger";
table="scott.emp";
path="sample";
assign=yes;
list all;
create work.test.view;
select EMPNO ENAME JOB;
subset where DEPTNO = 20;
proc print data=test;
id EMPNO;
```

This program first creates an "Access descriptor" called `test` in the work directory. The user ID, password, and server are all just as for the Oracle SQL Plus client. Oracle field names are converted to SAS format, and the contents of the database are listed. Note that the table "emp", owner "scott", has to be specified.

Then a "View descriptor", also called `test` is created containing some of the rows and columns from the remote table EMP. The resulting database view can then be used to read (or update) the database, according to the Oracle permissions available to the specified user.

Note that the syntax for PROC ACCESS is very specific; for example, even though the access and view descriptors are created in the temporary WORK library, it is still necessary to specify the three level names `work.test.access`, and `work.test.view`.

In Version 7 and higher, it is no longer necessary to go through this cumbersome process-- one no longer needs to create access and view descriptors. As the SAS/ACCESS Reference documentation notes:

The ACCESS procedure is still supported for the database systems and environments on which it was available in Version 6. However, it is no longer the recommended method for accessing relational DBMS data. It is recommended that you access your DBMS data more directly, using the LIBNAME statement or Pass-Through Facility. (SAS Institute Inc. 2007a)

As the following code illustrates, by using a “dynamic libname engine” SAS can treat the remote database as if it were a SAS dataset.

```sas
libname oralib oracle
user="scott" password="tiger" path="sample";
libname mydblib oracle dbprompt=yes;
```

This program produces exactly the same result as the preceding one, without having to leap through quite so many hoops. Note that although the Oracle table names are case sensitive, the procedure converts them into SAS data sets, which are not. If you want to preserve the case of a table name, be sure to specify the option `preserve_tab_names=yes` on the libname statement.

It may be that you do not want to hard code the database connection parameters into the program code. (I know I don’t.) In that case, try the following:

```sas
libname oralib oracle
```

4
You should see a database prompt, as follows. Note that you don’t get the prompt until you actually try to connect, that is, in this example when the PRINT procedure attempts to access the Oracle table. Note that you only have to supply the credentials the first time you connect.

```sas
proc print data=mydblib.emp;
    id empno;
    var ename job;
    where deptno = 20;
run;
```

**Figure 3. Using the libname statement interactively**

PROC CV2VIEW

It may be that you have some old SAS/ACCESS views that you would like to convert to PROC SQL views. SAS/ACCESS provides a utility procedure CV2VIEW that carries out this conversion. The syntax of this procedure is somewhat complex; see the Reference documentation for examples that you can customize for your site.
PROC DBLOAD

The DBLOAD procedure works in the opposite direction from PROC ACCESS. It is used to copy data into a DBMS from a SAS dataset. This PROC is useful for bulk loads, e.g., copying entire SAS datasets into Oracle, but caveat programmer: there are two important “features” that must be noted:

- The default load limit is 5000 records; in order to load larger tables, specify limit=0
- The PROC will abend if table exists; it can only be used to create new tables, unless the APPEND option is specified as a procedure option.

The following example illustrates creating a new Oracle table from SAS using DBLoad:

```sas
data emp2;
   retain empno 1234 ename="PRATTER" job="SAS GURU"
   mgr=7839 hiredate="01JAN08"d sal=5000 deptno=99;
   output;
run;

proc dbload dbms=oracle data=emp2;
   orapw="tiger"; user="scott"; path="sample";
   table="EMP2";
   load;
run;
```

The syntax is fairly close to that of PROC ACCESS. A SAS file called “emp2” is copied to Oracle as a table called “EMP2”. Note the “load” statement—if this does not appear the table will not be created. One important gotcha is that Oracle running on UNIX has trouble with lower case table names. You may find yourself creating a table you cannot access. Be sure to code the Oracle table name for the new table as all upper case. If you want to use the SAS variable labels as the Oracle field names, rather than the variable names, include a label statement as one of the procedure arguments.

Here is the resulting table, in SQL Plus. Note that the SAS datetime field has been correctly converted into an Oracle date field:

```
SQL> select * from emp2;
EMPNO ENAME JOB MGR HIREDATE SAL DEPTNO
------- ----- ------ ------- ------- ------
1234 PRATTER SAS GURU 7839 01-JAN-08 5000 99
```

Figure 4. Create new table with DBLOAD
SQL PASS-THROUGH

The least complicated way to manage remote database tables is with PROC SQL. This alternative offers a relatively straightforward interface for experienced SQL users. The main advantage is that SQL statements can be "passed through" to the database engine, where they execute in native mode. Except for bulk loads, where it is still probably better to use DBLOAD, PROC SQL has the advantage of simplicity and familiarity. It is important to note however that there are DBMS specific differences from system to system. Again, the best advice is to consult the documentation that comes with the specific version of the SAS/ACCESS product, and be sure to coordinate with your DB administrator before accessing her tables remotely!

The following example illustrates how to use PROC SQL as an alternative to PROC ACCESS:

```
proc sql;
  connect to oracle (user=scott orapw=tiger path=sample);
  create table EMP as
    select * from connection to oracle
    (select * from EMP);
  disconnect from oracle;
quit;
```

Three SQL statements are required: *Connect* and *disconnect* attach to the database and detach respectively. The SQL *select* statement has two parts: the parenthesized expression (select * from EMP) is "pass-through SQL". This code is sent to the Oracle database server to return the data from table "emp".

The outer *select * from connection to oracle* returns the result to SAS. Finally, the *create table* clause cause the results to be saved as the temporary dataset work.EMP. If this clause were omitted, PROC SQL would simply display the table in the output window (the default behavior for a "select").

UNIX SERVER/WINDOWS CLIENT

In addition to the SAS/ACCESS Interface to DBMS, if the database is on a UNIX server and SAS is running on a Windows client, it is possible use SAS/ACCESS to ODBC. The main advantage of this approach, as opposed to the DBMS specific products, is that it allows a Windows client to access a wide variety of database engines, not just the one it is licensed for.

ACCESS TO ODBC

The one drawback is that before using this product it is necessary to set up an ODBC data source that points to the database. The user needs to install the correct driver, for example, Microsoft ODBC for Oracle, and then go to Administrative Tools from Start>Programs and click on "Data Sources (ODBC)" to set up an ODBC DSN.

![Figure 5. Configuring ODBC for Oracle](image-url)
To add a new data source, just click on add and select the appropriate driver; in this case it is “Oracle in Oraclent11g_home1”. You should then be prompted for the connection parameters, as follows:

![Oracle ODBC Driver Configuration](image)

Click on “Test Connection” to make sure that you can access the database. This only needs to be done once for each client workstation, after which the database will be accessible using the following SQL code:

```sql
proc sql;
    connect to odbc (dsn=sample uid=scott pwd=tiger);
    create table EMP as
        select * from connection to odbc
        (select * from emp);
    disconnect from odbc;
quit;
```

As the comparison to the previous example clearly illustrates, SQL pass-through works the same way in the DBMS specific products and in SAS/ACCESS to ODBC. The only difference is that the “connect” statement references the “DSN” (data source name) created using the ODBC administrator.

As with the SAS/ACCESS Interface to Oracle, you can also specify a **libname** reference:

```sql
libname odbclib odbc dsn="sample" user="scott" password="tiger";
```

One can also use DBLOAD with ODBC to create an Oracle table from a Windows SAS dataset:

```sql
proc dbload dbms=odbc data=mylib.dept;
    dsn="oracle";
    uid="scott";
    pwd="tiger";
    table="emp2";
    limit=0;
    load;
run;
```
ACCESS TO OLEDB

ODBC is an old technology; Microsoft recommends using the newer OLEDB interface instead. In most cases OLEDB gives better performance than ODBC, and does not requires creating a data source name on the client workstation. The only trick is that you need to know the name of the Oracle provider; in this case it is "MSDAORA.1".

The syntax for the OLEDB libname statement should be quite familiar by now:

```sas
libname oradb oledb
  init_string="Provider=MSDAORA.1;Password=tiger;User ID=scott;
  Data Source=sample;Persist Security Info=True";
proc print data=oradb.emp;
  id empno;
run;
```

You can also connect to an OLEDB resource using the SQL pass-through facility, as follows:

```sas
proc sql;
  connect to oledb
    (init_string="Provider=MSDAORA.1;Password=tiger;User ID=scott;
    Data Source=sample;Persist Security Info=True");
  create table EMP as
    select * from connection to oledb
    (select * from EMP);
quit;
```

ACCESS TO PC FILE FORMATS

In contrast to the other two products described, SAS/ACCESS to PC File Formats is not used to access client/server databases. Instead, it can be used on the Windows platform to read and write database, delimited or spreadsheet files.

The easiest way to access the PC File Formats Interface is from the SAS Display manager File menu Import and Export Data commands, which run wizards that can be used to read and write PC files. There is an option in the Wizard to save the resulting SAS statements to a file. Here is an example of the result, which imports the employees table from the Microsoft Access Northwind sample database.

```sas
PROC IMPORT OUT= WORK.emp
  DATATABLE= "Employees"
  DBMS=ACCESS REPLACE;
  DATABASE="C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb";
  SCANMEMO=YES;
  USEDATE=NO;
  SCANTIME=YES;
RUN;
```

It is important to recognize that Microsoft Access, while relational, is not a true DBMS, since there is no database engine to connect to. All this code does is to make a copy of one file on the computer (the MS Access table) in another file format (SAS).

You can of course also connect to MS Access using ODBC. The following libname statement connects to the same table, without converting it, assuming that an ODBC data source has been set up with the name "Northwind" to point to the MS Access sample database.

```sas
libname odbclib odbc dsn="Northwind";
```

CONCLUSION

SAS software offers a number of choices for RDBMS access, depending on client/server platform. The dynamic libname engines for Oracle, DB2, MySQL, ODBC, and OLEDB available in Version 9.1 are a great improvement over the clumsy access and view descriptors of the early releases. SAS continues to enhance libname support for external data sources of all sorts, and this, along with the SQL Pass-through facility allows users to access a wide variety of organizational tables.
REFERENCES


CONTACT INFORMATION

Frederick Pratter
Computer Science/Multimedia Studies Program
Eastern Oregon University
La Grande OR
fpratter@eou.edu
http://cs.eou.edu/CSMM/fpratter

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 registered trademarks or trademarks of their respective companies.