Getting Started with SAS/Access for Oracle

F. Joseph Kelley, University of Georgia

Abstract. As the use of relational database software has grown, so too has the need for fast, convenient, and reliable methods of accessing and analyzing the massive amounts of data stored in them. Oracle Corporation's Oracle® database software has proven itself to be a popular, robust product and SAS Institute has provided a succession of means of reading and writing the data stored in the tables and views of the Oracle rdbms. This paper will examine the use of the product "SAS/Access for Oracle", particularly as it relates to reading data.

Introduction. There have been versions of "SAS/Access for Oracle" since the release of SAS 6.08 in the early 1990's. Typically, any significant analysis of data in an Oracle table (including a simple t-test), cannot be easily performed using SQL. The "aggregate" functions can provide simple descriptive statistics, and the "OLAP functions" (a special set of aggregate functions, also referred to as "analytic functions"), are somewhat limited in capability. SAS® software provides an extensive suite of analytical tools: regression, analysis of variance (ANOVA), time series analysis, non-linear analysis, cluster analysis, etc. Together, the two products make a formidable combination. Indeed, discussions of SAS and Oracle are common on the SAS discussion list, SAS-L, and this very presentation was prompted by one such exchange.

At its heart, SAS/Access for Oracle requires the Oracle Client software be installed and properly configured on the client machine. Any correctly-performed Oracle*Net configuration should suffice (though note that current Oracle documentation is discouraging the use of "Oracle Names"). While familiarity with PL/SQL is not required, a knowledge of SQL is of considerable use in working with any data, so this is expected (and almost all examples will make use of this). SQL may typically be replaced in SAS with a judicious combination of DATA and PROC steps, so understanding of SAS programming techniques is assumed.

The examples in this paper were developed using SAS Version 8.2, Oracle 9.2.0.4 database and Oracle 9.2.0.1 client; both client and server machines are running Windows 2000, SP4.

Accessing Oracle. There are two basic methods of reading data from Oracle tables/views:

- Using the "oracle" engine on the `libname` statement. This is a variation on the usual use of SAS Libraries. Call this method "Libname Oracle".

- Using “SQL Pass-Through” to connect directly to the Oracle instance.

SAS Libraries. (a little review to get started)
Internally-formatted SAS files are called “libraries”. For example, in the code

```
data work.Jan_2002;
    infile 'd:\sample\jan2002.data';
    input ID $ Count ;
run;
```

The Library is named WORK and has a Member named JAN_2002. Actually, WORK is a default name. The code below would have the same result:

```
data Jan_2002;
    infile 'd:\sample\jan2002.data';
    input ID $ Count ;
run;
```

JAN_2002 is simply stored in a form that is recognized by SAS (and a few other software products). This is similar in concept to an SPSS® Save file (`.sav`), an Excel™ spreadsheet (`.xls`) or an MS Word™ document (`.doc`). WORK files are deleted upon completion of a SAS session (in background mode, this would simply be the SAS program), so would represent a poor choice if the data were to be needed frequently over the course of time. It is possible to specify a more permanent location; I’ll use the name “PERM” for this, but any 8-character name will do:

```
libname perm 'e:\sasdata\2002';
data perm.Jan_2002;
      infile 'd:\sample\jan2002.data';
      input ID $ Count ;
run;
```

In this case, PERM is a directory (note only the path - `'e:\sasdata\2002'` – was specified). A check of the directory 'e:\sasdata\2002' would show the file `Jan_2002.sas7bdat` was present. At a later date, the data may be used – this time, a different Libname (but the same path) will be used:

```
libname example 'e:\sasdata\2002';
proc print  data=example.Jan_2002;
      title 'Data from January 2002';
run;
```

The Libname may include other specifications (as described in SAS documentation), one of these is an “engine” – this serves to translate data stored in one form into another. As an example, suppose some data had been created using SAS 6 some years earlier:

```
    libname example v6 'e:\sasdata\1997';
    proc print  data=example.Mar_1997;
    title 'Data from March 1997';
    run;
```

In this example, the V6 engine is used to translate the data as written by SAS Version 6.11 into a form that is recognized by SAS Version 8.2

The Oracle Engine.
The Oracle engine is specified:

```
libname library_ref oracle   user=<'> User_ID <'>
        password= <'> Password <'>
        path= <'> Oracle_Connection <'>  <other_options> ;
```

For example:

```
libname mydblib   oracle   user='SCOTT'
         password='TIGER' path='OR92Rock'  ;
```

In this case, the path points to the connection as set up in the Oracle Client software and is the SID used for the Oracle DB. The SCOTT schema is a part of one of the several sample databases provided by Oracle; these are installed by default in most cases. Note that a "schema=" may also be specified; in the above example, user SCOTT owns the SCOTT schema.

Here is a simple example of reading the data in the DEPT table:

```
      libname mydblib oracle user='SCOTT' password='TIGER'
            path='OR92Rock' ;
      run;
      proc print    data=mydblib.dept ;
      title "SCOTT: Dept";
      run;
```

Although the usual SAS DATA and PROC steps may be used (as above), more often the data will be manipulated via SQL – in this case, SAS' Proc SQL. As an example:

```
      libname mydblib oracle user='SCOTT' password='TIGER'
            path='OR92Rock' ;
      run;
      proc sql   ;
            select e.empno, e.ename, e.hiredate, d.dname
            from mydblib.emp e, mydblib.dept d
            where e.deptno=d.deptno ;
      quit; run;
      libname mydblib CLEAR; run;
```

In an interactive session, the results will be displayed in the Output window:

```
                                   Proc SQL Output


                EMPNO  ENAME                  HIREDATE  DNAME
                ________________________________________________________

                 7369  SMITH        17DEC1980:00:00:00  RESEARCH
                 7499  ALLEN        20FEB1981:00:00:00  SALES
                 7521  WARD         22FEB1981:00:00:00  SALES
                 7566  JONES        02APR1981:00:00:00  RESEARCH
                 7654  MARTIN       28SEP1981:00:00:00  SALES
                 7698  BLAKE        01MAY1981:00:00:00  SALES
                 7782  CLARK        09JUN1981:00:00:00  ACCOUNTING
                 7788  SCOTT        19APR1987:00:00:00  RESEARCH
                 7839  KING         17NOV1981:00:00:00  ACCOUNTING
                 7844  TURNER       08SEP1981:00:00:00  SALES
                 7876  ADAMS        23MAY1987:00:00:00  RESEARCH
                 7900  JAMES        03DEC1981:00:00:00  SALES
                 7902  FORD         03DEC1981:00:00:00  RESEARCH
                 7934  MILLER       23JAN1982:00:00:00  ACCOUNTING
```

The MYDBLIB connection was terminated (CLEARed) when it was no longer needed. In the examples thus far, that occurs once the data is read. In some subsequent examples this will not be the case.

SAS' Proc SQL conforms to the SQL-92 standard; the display could be modified to provide "nicer" column names. Additionally, the HIREDATE is being converted from the Oracle DATE datatype, which includes not only the date, but also the hour, minute and second. SAS uses the Datetime (output) format which matches this, but we'd rather not have the time included at all; of the several SAS Date formats available, the "`yymmdd10.`" should look good – this is done with the "sasdatefmt=" option. Substituting just the Proc SQL statements:

```
proc sql   ;
      select e.empno  "Employee ID No",
            e.ename  'Employee Last Name',
            e.hiredate 'Date Hired',
            d.dname  'Department'
      from mydblib.emp(sasdatefmt=(hiredate=yymmdd10.)) e,
            mydblib.dept d
      where e.deptno=d.deptno ;
      quit;
run;
```

Note that the SELECT statement does NOT set up column aliases. Instead the code "`d.dname  'Department'`" effectively sets up a "column label". There is a difference which will be discussed below. The results of the revised query, including the revision of the way the date is displayed and the labels are displayed below.

```
                            Proc SQL Output


              Employee  Employee
                 ID No  Last Name    Date Hired  Department
              ------------------------------------------------
                  7934  MILLER       1982-01-23  ACCOUNTING
                  7782  CLARK        1981-06-09  ACCOUNTING
                  7839  KING         1981-11-17  ACCOUNTING
                  7876  ADAMS        1987-05-23  RESEARCH
                  7369  SMITH        1980-12-17  RESEARCH
                  7902  FORD         1981-12-03  RESEARCH
                  7566  JONES        1981-04-02  RESEARCH
                  7788  SCOTT        1987-04-19  RESEARCH
                  7521  WARD         1981-02-22  SALES
                  7698  BLAKE        1981-05-01  SALES
                  7900  JAMES        1981-12-03  SALES
                  7499  ALLEN        1981-02-20  SALES
                  7654  MARTIN       1981-09-28  SALES
                  7844  TURNER       1981-09-08  SALES
```

Using Dates, Pt 1. As shown above, the display of the date value (in HIREDATE) may be altered by assigning an alternative SAS date format with the variable (column); this is done with the option "sasdatefmt=". There is an extensive selection of such formats; they are described in SAS documentation. Suppose only persons hired after December 31, 1985 had been needed? This would require the use of a "SAS Date Constant". These constants take the general form: 'DDMonYYYY'**d** e.g. `'31dec1985'd`. Here is how the previous code might be modified to select only those hired after Dec. 31, 1985:

```
proc sql   ;
      select e.empno  "Employee ID No",
            e.ename  'Employee Last Name',
            e.hiredate 'Date Hired',
            d.dname  'Department'
   ▶  from mydblib.emp(sasdatefmt=(hiredate=yymmdd10.)) e,
            mydblib.dept d
      where e.deptno=d.deptno  AND
   ▶        hiredate > '31dec1985'd;
quit;
run;
```

It must be kept in mind that the SAS date constant is used because with "Libname Oracle" access, SAS is processing the data. "HIREDATE" is a column in an Oracle table and is stored as an Oracle DATE datatype; SAS can read and (correctly) process this type. This example will be revisited shortly.

```
                       Using a SAS Date Constant

              Employee  Employee
                 ID No  Last Name    Date Hired  Department
              ————————————————————————————————————————————————

                  7788  SCOTT        1987-04-19  RESEARCH
                  7876  ADAMS        1987-05-23  RESEARCH
```

Creating Tables. Now suppose the results of a query were to be passed to a SAS procedure. That is, the data were to be used outside the PROC SQL environment. In such a case, a Table should be created. In SAS terminology, this table (in SQL) would be treated as a SAS (Library) File (often called a "SAS dataset" – this terminology is avoided here because of possible confusion with other types of files associated with SAS). In SQL this is done via the "CREATE TABLE" statement. In Proc SQL, CREATE TABLE takes the results of a SELECT and writes them to the table (library file) specified. As an example:

```
proc sql    _method _tree ;
    create table EmployeeInfo as
        select * from (
            select e.empno as EmployeeID ,
                  e.ename as EmployeeLastName ,
                  e.hiredate ,
                  e.sal as Salary ,
                  d.dname  as Department ,
                  d.deptno as DeptNO
            from mydblib.emp(sasdatefmt=(hiredate=yymmdd10.)) e,
                  mydblib.dept d
            where e.deptno=d.deptno
        );
quit;
run;
proc print   data=EmployeeInfo  noobs; run;
```

In fact, this uses a type of nested query: the Oracle tables DEPT and EMP (actually, in SQLPlus, they would be addressed as SCOTT.DEPT and SCOTT.EMP – but the connection as SCOTT removes a need to specify the schema) are queried and the results of this are passed to the outer SELECT which takes everything found. The results are saved in EmployeeInfo. A Proc PRINT of EmployeeInfo shows (portion):

```
                                  EmployeeInfo Data



    Employee    Employee
```

```
        ID       LastName         HIREDATE        Salary    Department         DEPTNO

      7934       MILLER         1982-01-23       1300.00    ACCOUNTING             10
      7782       CLARK          1981-06-09       2450.00    ACCOUNTING             10
      7839       KING           1981-11-17       5000.00    ACCOUNTING             10
      7876       ADAMS          1987-05-23       1100.00    RESEARCH               20
      7369       SMITH          1980-12-17        800.00    RESEARCH               20
      7902       FORD           1981-12-03       3000.00    RESEARCH               20
      7566       JONES          1981-04-02       2975.00    RESEARCH               20
      7788       SCOTT          1987-04-19       3000.00    RESEARCH               20
      7521       WARD           1981-02-22       1250.00    SALES                  30
      7698       BLAKE          1981-05-01       2850.00    SALES                  30
      7900       JAMES          1981-12-03        950.00    SALES                  30
      7499       ALLEN          1981-02-20       1600.00    SALES                  30
      7654       MARTIN         1981-09-28       1250.00    SALES                  30
      7844       TURNER         1981-09-08       1500.00    SALES                  30
```

Column and Table Aliases. Column Aliases are commonly used to provide more descriptive names for columns than those used in the columns themselves. In terms of a SAS Library file, column names correspond to variable names. When using the Oracle engine ("Libname Oracle") access, SAS will not honor column alias use *unless a table is being created*. It will use "column labeling" – this does not conform to the ANSI SQL standard, and is subtly different. "Column Labels", on the other hand, are not honored when creating a table.

- "select e.empno as EmployeeID" -- Unless part of a subquery in a "Create Table", the column name will be displayed as EMPNO.
- "select e.empno 'Employee ID No' " – If used in a subquery to a "Create Table", this label will NOT be used – again, the column name will be EMPNO.

Table Aliases are often used in JOINs to reduce the visual clutter that may be associated with full table specifications; they also allow the same table to be referenced with different aliases in more complicated JOINs. In SQLPlus this table specification is of the form: <schema.>table . In SAS, this would be <libref.>table. The column(variable) specification is <table.>column-name. With lengthy table names, this becomes tedious and error-prone. The SQL standard allows for a "shorthand" method: a table alias. There are two commonly-accepted methods of assigning a table alias:

- FROM table_name AS table_alias
- FROM table_name table_alias

As far as SAS is concerned, either is acceptable. In the "Libname Oracle" method of accessing Oracle tables, how this is coded does not matter because SAS processes the SQL statements. However, Oracle disallows the use of AS in the creation of a table alias; this has no effect on the method shown in the preceding examples. It would in some that will follow and for this reason, use of 'AS' is not encouraged. In much of SAS' documentation on Proc SQL, 'AS' is frequently used when creating a table alias; it is for this reason the author has added these cautionary notes.

VIEWS. The discussion above has focused on tables, but the use of Views is similar, whether creating them in SAS or reading them from the Oracle DB.

The following was executed in SQLPlus to create the View "VW_EMPLOYEE"

```
SQL> create view VW_Employee as
  2  select emp.empno  as EmployeeID,
  3      emp.ename  as EmployeeLastName ,
  4      dept.dname  as Department
  5  from emp inner join
  6      dept
  7* on emp.deptno=dept.deptno
```

Here is the SAS code to read the View VW_Employee:

```
proc sql   ;
      select *
      from mydblib.vw_employee ;
quit;
run;
```

Only a portion of the results are displayed:

```
                                Proc SQL Output                               1
                             SELECT on VIEW: Results

                      EMPLOYEEID  EMPLOYEELASTNAME  DEPARTMENT
                      ____________________________________________
                            7369  SMITH             RESEARCH
                            7499  ALLEN             SALES
```

a JOIN used to connect the View (VW_Employee) with the Table(Emp):

```
proc sql   ;
      select vw_employee.EmployeeID,
            vw_employee.EmployeeLastName ,
            vw_employee.Department,
            emp.sal as salary
      from mydblib.vw_employee inner join
            mydblib.emp
      on vw_employee.employeeid=emp.empno ;
quit;
```

A portion of the output is reproduced:

```
                                   Proc SQL Output                             1
                               SELECT on VIEW Results

                EMPLOYEEID  EMPLOYEELASTNAME  DEPARTMENT             SAL
                -------------------------------------------------------
                      7369  SMITH             RESEARCH            800.00
                      7499  ALLEN             SALES              1600.00
                      7521  WARD              SALES              1250.00
                      7566  JONES             RESEARCH           2975.00
```

It should be noted that SAS supports the ANSI Standards for JOIN as well as the older forms.

Proc SQL may create Views that are used both within Proc SQL and by other DATA and PROC steps. The entire program is included.

```
title  "Proc SQL Output";
title2 "SELECT on VIEW: Results";
libname mydblib oracle user='SCOTT' password='TIGER' path='OR92Rock';
run;
proc sql   _method _tree ;
      create view v_employee as
      select * from (
            select v_e.EmployeeID,
                  v_e.EmployeeLastName ,
                  v_e.Department,
                  e.sal as salary,
                  e.deptno as DeptCode
            from mydblib.vw_employee v_e inner join
                  mydblib.emp e
            on v_e.employeeid=e.empno
      );
quit;
run;
proc print data=v_employee;
run;
libname mydblib CLEAR;
run;
proc print data=v_employee;
run;
```

Again, only a portion is shown:

```
                                   Proc SQL Output                             1
                               SELECT on VIEW: Results

      Obs    EMPLOYEEID    EMPLOYEELASTNAME    DEPARTMENT          SALARY    DEPTCODE

        1       7369          SMITH            RESEARCH            800.00       20
        2       7499          ALLEN            SALES              1600.00       30
        3       7521          WARD             SALES              1250.00       30
```

This is pretty much what has already been seen. Recall, however, that Proc PRINT was executed twice: once before the connection to the database was severed (CLEARed) and once after. Looking at the log:

```
151
152   run;                            « At this point, PROC SQL has completed
153   proc print data=v_employee;    « Now, PRINT the VIEW V_EMPLOYEE
154   run;

NOTE: There were 14 observations read from the data set WORK.V_EMPLOYEE.
NOTE: PROCEDURE PRINT used:           « No Problems
      real time           0.09 seconds
      cpu time            0.03 seconds


155   libname mydblib CLEAR;          « Now CLEAR the Libref MYDBLIB
NOTE: Libref MYDBLIB has been deassigned.
156   run;
157   proc print data=v_employee;    « Print the VIEW V_EMPLOYEE again
ERROR: Libname MYDBLIB is not assigned.«There is no longer a connection to Oracle; the
ERROR: Libname MYDBLIB is not assigned. View exists, but the data cannot be accessed.
ERROR: SQL View WORK.V_EMPLOYEE could not be processed because at least one of the
       data sets, or views, referenced directly (or indirectly) by it could not be
       located, or opened successfully.  « This Error message provides more detail
158   run;

NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE PRINT used:
      real time           0.01 seconds
      cpu time            0.01 seconds
```

When creating a VIEW, SAS writes only a structure for the library file: it has populated the Program Data Vector (PDV) with the variable (column) names, and the metadata about them, but it does not save the data itself – it relies upon the connection for that. Should the connection to Oracle no longer be viable, even the library header information is lost; nothing usable will remain of the VIEW. A subsequent reconnect to the database will restore the View, so long as it is done within the same session (without a qualifier, the View will have been written as WORK.V_EMPLOYEE). As was the case with data library files (e.g. PERM.JAN_2002), Views may also be written to permanent storage.

SQL Pass-Through.

The "libname Oracle" method of accessing Oracle tables is straightforward and may be used in almost all circumstances. Although most of the examples above used it within the context of Proc SQL, one of the very earliest examples used it in exactly the same way it might have been done when reading a member of *any* SAS Library file. Using this method, Oracle tables and views may be directly accessed by SAS DATA and PROC steps very much as if they were native-format

SAS Library files. Proc SQL can provide a very helpful method of filtering the data, but it isn't strictly necessary. This is not the case with "SQL Pass-Through", as the very name suggests. In this, Oracle, not SAS, processes the SQL statements and returns the results to your SAS session. The general format is:

```
proc sql <options> ;
CONNECT TO ORACLE <AS alias> (USER=ORACLE-user-name
      PASSWORD=ORACLE-password
      PATH="ORACLE-path-designation"
      BUFFSIZE=number-of-rows
      PRESERVE_COMMENTS);

      [sql statements]

DISCONNECT FROM ORACLE;
quit;
```

As an example:

```
proc sql   ;
CONNECT to oracle as oradb
      (user='SCOTT' password='TIGER' path='OR92Rock');
%put &sqlxmsg;
      SELECT *
      FROM connection to oradb
                  (SELECT table_name,
                  tablespace_name
                  FROM user_tables) ;
%put &sqlxmsg;
DISCONNECT from oradb;
quit;
run;
```

A few notes on the example code:

- Use of an ALIAS (in this case "oradb"), is optional, but becomes necessary with multiple connections.
- In older documentation, there are examples of "connect to ORARDB". "ORARDB" is obsolete for all releases of Oracle since the release of Oracle 8i (Oracle 8.1.5); use "ORACLE" to specify the database software (unless it is pre-8i).
- The statement "`%put &sqlxmsg;`" simply echoes any Oracle return codes. As a check, replace "table_name" with "table_names".
- The options shown for both "connect to" and "from connection" are only a subset – see SAS documentation for a more exhaustive list.
- The "table" USER_TABLES is not a table at all; it is one of the static data dictionary views.

It is possible to execute additional SQL statements, depending upon roles granted; this is done via the "EXECUTE" statement. This topic, however, exceeds the scope of this paper. Unfortunately, it is not possible to access the Oracle metadata about objects via the "`desc[ribe] [schema.]object-name`" statement (e.g. "desc scott.emp"), however, the same information is available (albeit in a more cumbersome fashion) via SQL (shown as an SQL query by user 'SCOTT', this may be adapted to SAS):

```
  SQL> select owner, table_name, column_name, data_type,
    2> data_length, data_precision, data_scale
    3> from all_tab_cols
    4> where table_name= 'EMP';
```

Much of what has already been seen in "libname oracle" remains the same in SQL Pass-Through. As an example of writing a query to a table:

```
proc sql   ;
    connect to oracle as oradb  (user='SCOTT' password='TIGER'
       path='OR92Rock');
%put &sqlxmsg;
    create table work.pt_example as
        select *
        from connection to oradb
            (select e.empno as EmployeeID ,
                e.ename as LastName ,
                e.hiredate ,
                e.sal as Salary ,
                d.dname  as Department ,
                d.deptno as DeptNO
            from emp e   inner join   dept d
            on e.deptno=d.deptno
        );
%put &sqlxmsg;
run;
    disconnect from oradb;
quit;
run;
proc print   data=pt_example;
title "Using SQL Pass-Through";
run;
```

Only a portion of the results are displayed:

```
                             Using SQL Pass-Through

 Obs EMPLOYEEID LASTNAME                HIREDATE    SALARY DEPARTMENT     DEPTNO

   1     7369    SMITH       17DEC1980:00:00:00    800.00 RESEARCH         20
   2     7499    ALLEN       20FEB1981:00:00:00   1600.00 SALES            30
```

Using Dates, Pt 2. An earlier example showed how to use an alternative format to display dates and demonstrated how to select records based on a comparison to a date value. These used SAS methods; with SQL Pass-Through, Oracle methods must be employed. Specifically, to change the display of "HIREDATE", an Oracle date display format must be associated with the column. For the Date comparison, it is necessary to convert a "human-readable" date into an Oracle DATE datatype. These operations are conceptually similar to the methods employed with "libname oracle", but the specific syntax is very different. These operations will use functions provided by Oracle for datatype conversions:

- TO_CHAR to_char(date,'format') – displays a date value using 'format'
- TO_DATE to_date(string,'format') – converts a string to a date using 'format'

'format' is optional in this – the default is 'DD-MON-YYYY' (e.g. 23-OCT-1993). The ANSI SQL standard uses a 'CAST'; the Oracle-specific functions are more powerful.

Here is the modified Proc SQL code:

```
proc sql   ;
    connect to oracle as oradb  (user='SCOTT' password='TIGER'
        path='OR92Rock');
%put &sqlxmsg;
    create table work.pt_example  as
        select  *
        from connection to oradb
            (select e.empno as EmployeeID ,
                e.ename as LastName ,
      1 ►       TO_CHAR(e.hiredate,'dd-Mon-yyyy') as DateHired ,
                e.sal as Salary ,
                d.dname  as Department ,
                d.deptno as DeptNO
            from emp e   inner join   dept d
            on e.deptno=d.deptno
      2 ►   where hiredate >= TO_DATE('31-DEC-1985')
        );
%put &sqlxmsg;
run;
    disconnect from oradb;
quit;
run;
```

The revised code is indicated with "►". In the first, note that a non-default format is specified. Use of "Mon" will produce a mixed-case name. Also note the use of a column alias – without this, the column will appear as some variant of "TO_CHAR_E_HIREDATE…". In the second, note the *original* column name is used in the comparison – not the alias. This statement selects all whose

HIREDATE is greater-than-or-equal-to the date '31-DEC-1985'. The (not surprising) results:

```
                              Using SQL Pass-Through

  Obs   EMPLOYEEID  LASTNAME    DATEHIRED       SALARY  DEPARTMENT      DEPTNO

   1       7788     SCOTT       19-Apr-1987    3000.00  RESEARCH          20
   2       7876     ADAMS       23-May-1987    1100.00  RESEARCH          20
```

As might be expected, Views act in a way similar to what has already been seem: as long as the connection to the Oracle DB is maintained, they will work. They will fail if the connection fails.

SAS literature is encouraging the use of "libname oracle" for access, but SQL Pass-Through allows many already-existing SQL programs to continue to be used. The queries used in these examples are trivial: rewriting the far more sophisticated queries that may be encountered in production environments – queries that may be making use of SQL:1999 or SQL:2003 syntax – would be a daunting task.

SQL:1999. The newest SQL standards (SQL:1999 and now SQL:2003) superseded the SQL-92 standard. The newer standard incorporates a number of features that are quite useful in practice. For example, ROLLUP:

```
create table EmployeeInfo as
      select *
          from connection to oradb
              (select d.dname  as Department ,
                  count(e.empno) as NumEmployees,
                  e.ename as LastName,
                  TO_CHAR(sum(e.sal), '$99,990.99') as TotalSalary
               from emp e,
                  dept d
               where e.deptno=d.deptno
               group by rollup (d.dname, e.ename)
              );
```

This will summarize total employees per department, as well as total salary per department and provide grand totals for each:

```
                                          Proc SQL Output

          Obs     DEPARTMENT          NUMEMPLOYEES     LASTNAME       TOTALSALARY

            1     SALES                          1     WARD             $1,250.00
            2     SALES                          1     ALLEN            $1,600.00
            3     SALES                          1     BLAKE            $2,850.00
            4     SALES                          1     JAMES              $950.00
            5     SALES                          1     MARTIN           $1,250.00
            6     SALES                          1     TURNER           $1,500.00
            7     SALES                          6                      $9,400.00
            8     RESEARCH                       1     FORD             $3,000.00
            9     RESEARCH                       1     ADAMS            $1,100.00
           10     RESEARCH                       1     JONES            $2,975.00
           11     RESEARCH                       1     SCOTT            $3,000.00
           12     RESEARCH                       1     SMITH              $800.00
           13     RESEARCH                       5                     $10,875.00
           14     ACCOUNTING                     1     KING             $5,000.00
           15     ACCOUNTING                     1     CLARK            $2,450.00
           16     ACCOUNTING                     1     MILLER           $1,300.00
           17     ACCOUNTING                     3                      $8,750.00
           18                                   14                     $29,025.00
```

This was added in SQL:1999 and is not available in Proc SQL (though a similar report could be generated by the MEANS, TABULATE or REPORT Proceedures).

With the new standard, a new selection of "aggregate" functions were added: the "analytic" functions (sometimes called the "OLAP functions"). Many of the Aggregate functions (e.g. SUM, MIN, COUNT, etc) are available in Proc SQL. Some are available with different names ('STD' instead of 'STDDEV' for the standard deviation), and many of the SAS functions seen in the DATA step are available (these far exceed the SQL-92 standard). Nevertheless, the OLAP functions afford some real analytic capabilities to the database itself. This is based on an example found in *Oracle 9i:SQL Reference, Release 2 (9.2)*, October 2002, Part No. A96540-02 (pp 6-131 – 6-132) (http://download-east.oracle.com/docs/cd/B10501_01/server.920/a96540.pdf). It has been adapted for this paper. Note that instead of SCOTT/TIGER this is SH/TIGER. Several sample databases are installed, these additional schemas are described in **References**. This is a relatively simple example and assumes a linear relationship between "prod_list_price" and "quantity_sold", such that prod_list_price is the independent variable (x) and quantity_sold is the dependent variable (y). In statistical models this is typically shown as

$$y=b_0+b_1x$$

This is just the equation of a line.

slope	$= b_1$	r2	= coefficient of (linear) determination (r^2)	avgx	= average of X
intercept	$= b_0$	count	= number of non-null pairs	avgy	= average of Y

All values are computed on non-null pairs

```
proc sql;
connect to oracle as oradb
      (user='SH' password='TIGER' path='OR92Rock');
%put &sqlxmsg;
    create table work.olap_example as
        SELECT  *
        from connection to oradb
            (SELECT  s.channel_id,
            REGR_SLOPE(s.quantity_sold, p.prod_list_price) SLOPE ,
            REGR_INTERCEPT(s.quantity_sold, p.prod_list_price)INTCPT,
            REGR_R2(s.quantity_sold, p.prod_list_price)  RSQR ,
            REGR_COUNT(s.quantity_sold, p.prod_list_price)  COUNT ,
            REGR_AVGX(s.quantity_sold, p.prod_list_price)  AVGLISTP ,
            REGR_AVGY(s.quantity_sold, p.prod_list_price)  AVGQSOLD
            FROM  sales s, products p
            WHERE s.prod_id=p.prod_id AND
                  p.prod_category='Men'  AND
                  s.time_id=to_DATE('10-OCT-2000')
            GROUP BY s.channel_id);

%put &sqlxmsg;
      disconnect from oradb;
quit;
run;
proc print   data=work.olap_example  noobs;
title "Data: WORK.OLAP_Example";
title2 "prod_category='Men'" ;
run;
```

Running this query, we see:

```
                          Data: WORK.OLAP_Example
                            prod_category='Men'

   CHANNEL_
      ID         SLOPE        INTCPT       RSQR       COUNT     AVGLISTP     AVGQSOLD

      C        -0.068369     16.6278    0.051343      20       65.4950      12.1500
      I         0.019710     14.8114    0.001631      46       51.4804      15.8261
      P        -0.012474     12.8545    0.017040      30       81.8700      11.8333
      S         0.006156     13.9919    0.000898      83       69.8133      14.4217
      T        -0.004113      5.2272    0.008132      27       82.2444       4.8889
```

Although SAS' Proc REG and Proc GLM provide considerably more robust means of analysis (I don't believe there is any way to perform an analysis of variance – or even a t-test), the Analytic functions can provide some initial information. The Oracle Analytic functions are described in the SQL Reference.

Conclusion.

Use of SAS/Access for Oracle provides a fairly straightforward way for analysts both to read Oracle tables and to analyze the data in them. Two methods of this access were presented:

- "libname oracle" - which allows an Oracle table to be viewed (and treated) as if it were a member of a SAS Library file. In this way, it may be directly accessed by both DATA and PROC steps, as well as Proc SQL.
- SQL Pass-Through - which allows the RDBMS to process the SQL statements instead of SAS. This method allows existing SQL queries to be run with only minimal modification. Further, it allows access to newer features of SQL not provided by the current level of SQL supported by SAS. Finally, although most users will not need this, it is also possible to use "hints" with this method.

This paper was intended to provide an introduction together with some simple, working examples. The sample schemas used are installed by default, and if present, can provide an excellent vehicle for testing this connectivity. Use of SAS and Oracle together is common in many industries, especially financial and pharmaceutical; this product is integral to that.

Author Contact.

Your comments and questions are welcome and encouraged!

F. Joseph Kelley
Enterprise IT Services
University of Georgia
Athens, GA 30602-1911
USA

jkelley@uga.edu

References.

Principal SAS Documentation:
SAS OnlineDoc®, Version 8, Copyright (c) 1999 SAS Institute Inc., Cary, NC
http://v8doc.sas.com/sashtml/
This is beginning to show its age, several procedures and keywords are (effectively) obsolete. The "SAS Online Help" is less comprehensive, but more current.
SAS OnlineDoc® 9.1.3, Copyright (c) 2002-2004 SAS Institute Inc., Cary, NC
http://support.sas.com/onlinedoc/913/docMainpage.jsp
SAS now updates the documentation with each release; this is the most current.

Principal Oracle Documentation:
Oracle Technology Network provides substantial documentation for all Oracle products. All are Copyright (c) the Oracle Corporation, Redwood Shores, CA. Several URLs will lead you to this:
http://www.oracle.com/technology/documentation/index.html
This link will connect you to documentation for Oracle 10g R1 and R2 as well as 9i R1 and R2. It is the main link.
The mass of online documentation for Oracle 9i R2 is at
http://www.oracle.com/pls/db92/db92.homepage

The interested reader may wish to examine:
Sample Schemas
Oracle 9i: Sample Schemas, Release 2 (9.2), March 2002, Part No. A96539-01
http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96539.pdf
SQL Reference
Oracle 9i: SQL Reference, Release 2 (9.2), October 2002, Part No. A96540-02
http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96540.pdf

Additional References.
In addition to "SAS/Access for Oracle", there are other SAS/Access products. The article below provides an overview of "SAS/Access for PC File Formats" and "SAS/Access to ODBC".
Kelley, F. Joseph, "New ways to access your data with SAS", *UCNS Computer Review*, Fall 2001
http://www.eits.uga.edu/tti/Computer_Review/Fall2001/SAS-data.html

An earlier paper on SAS/Access for ODBC was written by David Riba. The version of SAS used in this paper was 6.12, but the concepts have changed very little. This paper was presented at both SUGI and SESUG.
Riba, S. David, and Elizabeth Riba, "ODBC: Windows to the Outside World", *Proceedings of the SouthEast SAS Users Group Conference, 1996*
http://www.jadetek.com/pubs.htm

The paper has focused upon *reading* from Oracle. The author has assumed this is the most common activity. However, for those who will want to write to an Oracle table, the following paper was presented at SUGI29. A revised version was presented at SESUG-04 in Nashville. Her information about the use of BULKLOAD should be of use to those who will be writing large tables.
Levin, Lois, "Methods of Storing SAS® Data into Oracle Tables", *Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference*
Copyright © 2004 by SAS Institute Inc., Cary, NC
http://www2.sas.com/proceedings/sugi29/106-29.pdf

This paper has been silent on the matter of SQL optimization. Performance tuning in Oracle requires cost-based optimization, and an understanding of how even small changes in SQL code can result in (much) more or *less* efficient queries. It also requires a good deal of familiarity with the data and its organization. In Oracle, it is possible to use "hints" to alter the SQL processing. Hints may be passed in SQL Pass-Through by specifying "Preserve_Comments" in the CONNECT statement (see syntax specification above). Russ Lavery has a paper that provides information on how the SAS SQL optimizer may be investigated and more efficient queries structured. This would *not* apply to SQL Pass-Through as Oracle, not SAS, processes the SQL statements. This paper was first presented at the NorthEast SAS Users Group (NESUG) Conference November, 2004. An enhanced version was subsequently presented at the International SAS Users Group (SUGI) Conference in April, 2005, and again at SESUG-05.

Lavery, Russ, "The SQL Optimizer Project: _Method and _Tree in V9.1".
Proceedings of the Thirtieth Annual SAS® Users Group International Conference. Cary, NC: SAS Institute Inc.
Copyright © 2005 by SAS Institute Inc., Cary, NC
http://www2.sas.com/proceedings/sugi30/101-30.pdf

A rich source of information on SAS and its interaction with Oracle (and other RDMDS) is the "SAS Discussion List" – SAS-L (sas-l@listserv.uga.edu). The original impetus for this paper came from several discussions which had occurred in the past year on this list and it remains an excellent resource.

Acknowledgements.

The author would like to thank the frequent contributors to SAS-L who have commented on the topic.

Trademarks.