SAS Access by Example

Table of Contents

Introduction

This introductory document shows you how to use SAS/Access to Oracle/OLEDB/ODBC by example.

SAS/Access is used to push data to and pull data from a variety of databases such as Oracle, SQL Server, MySQL, Microsoft Access and even Microsoft Excel, to name a few.

SAS/Access to Oracle is used solely to pull data from and push data to an Oracle database.

SAS/Access to ODBC is used to push data to and pull data from ODBC data sources. ODBC, which stands for **O**pen **D**ata**b**ase **C**onnectivity, is an older data access method used to communicate with data sources. For the most part, it has been replaced by OLEDB which is faster. This is why SAS has created SAS/Access to OLEDB. With that said, SAS/Access to ODBC may be the only SAS/Access product installed on your desktop. You can run the `proc setinit noalias;run;` command from within SAS to determine the products you have installed at your site.

Although this document focuses on Oracle databases, one method you can use to access SQL Server databases is via SAS/Access to OLEDB. In the SAS/Access to OLEDB examples below, you will have to change the connection string appropriate for your SQL Server database and initial catalog as shown in the code below:

```
libname dbSQL oledb init_string="Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security
Info=False;Initial Catalog=your-initial-catalog;Data Source=your-dbserver-name";
run;

proc sql noprint noerrorstop;
 connect to oledb as SQLDW (init_string="Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security
Info=False;Initial Catalog=your-initial-catalog;Data Source=your-dbserver-name ");

 execute( ... ) by SQLDW;

 create table table_name as
  select * from connection to SQLDW(
   SQLServer-SQL-GOES-HERE
  );

 disconnect from SQLDW;
quit;

libname dBSQL clear;
run;
```

In the examples for ODBC below, we are assuming SAS is installed on a Windows platform. Setting ODBC data sources are via the Data Sources (ODBC) Applet. If you are running SAS/Access to ODBC on a UNIX machine, you will have to install the unixODBC (or similar) software to create your data source names.

In order to access an Oracle database via SAS/Access to OLEDB, SAS/Access to ODBC and SAS/Access to Oracle, you will have to have the file `TNSNAMES.ORA` placed on your computer along with the Oracle client software. This software contains Oracle's SQL*Net which allows for communication to and from an Oracle database. The `TNSNAMES.ORA` file should be placed in the \NETWORK\ADMIN folder once the Oracle client software has been installed on your computer. This file contains a list of your company's Oracle databases along with their server locations and service names and is maintained by your company's DBA Group. Please contact your Help Desk or the Oracle Database Administrators (DBAs) for help with this installation and setup.

In several of the examples below, we show you how to create database indexes. Note that there's a lot more to know about indexes than just what's shown below. For example, instead of using a normal B-Tree index, you may want to use a BITMAP index. Also, once an index has been created on an Oracle database table, it's a good idea to run the DBMS_STATS.GATHER_TABLE_STATS procedure on that table. This will allow Oracle a fighting chance of being able to optimize your queries. Note that the ANALYZE procedure used to gather statistics for tables and indexes has been deprecated by Oracle in favor of the more robust DBMS_STATS package.

How SAS/Access Accesses Databases from within SAS

SAS/Access not only allows you to pass SQL directly to the database, but you can also create a LIBNAME statement – just as you would to a folder containing SAS datasets – to a database. Thus, you can use SAS DATA and procedure steps to access a database. But! There is a price to pay for this madness! In order for SAS to pull data from a database, SAS has to translate your DATA and procedure steps requests into SQL because databases, for the most part, only understand SQL. Thus, if you use a WHERE clause in a DATA step using a LIBNAME to a database, that WHERE clause is translated from SAS code into SQL code whenever SAS can do the translation, otherwise, no WHERE clause is passed at all. So, the question is: When can SAS do the translation? Here is a simple DATA step request using a LIBNAME `dbORACLE` to an Oracle database:

```
data MyData;
 set dbORACLE.MY_DB_TABLE(where=(CUST_KEY=12345);
run;
```

This is translated into SQL as

```
SELECT *
 FROM MY_DB_TABLE
 WHERE CUST_KEY=12345
```

If you left off the WHERE clause in the dataset option above, then you would attempt to pull back **all of the data** to your SAS session. This may fail because your computer might not have enough resources to hold all the of rows data coming back from the database. Here is another example DATA step:

```
data MyData;
 set dbORACLE.MY_DB_TABLE(where=(datepart(DATE_KEY)='01JAN2005'd));
run;
```

This translates into the following SQL:

```
SELECT *
  FROM MY_DB_TABLE
```

Hey! What happened to the WHERE clause? Since the SAS DATEPART() function is not recognized by any database, SAS removes the WHERE clause and attempts to pull back **all of the data** to your computer which THEN performs the DATEPART() function. That is, the DATEPART() function is applied *after all of the data is brought back from the database*. If the amount of data is small, this should work, but if MY_DB_TABLE contains billions of rows it might fail on your computer.

So, how do you avoid this problem? The best way is to use *Pass-Through SQL*. This will allow you to write SQL and have it passed *exactly as written* to the database. The database will then run this SQL code as if you by-passed SAS directly. This allows you to take advantage of database-specific features which are not available in SAS.

For example, here is an example of SAS/Access to ODBC Pass-Through SQL to create a SAS dataset from the results of an Oracle query run on a database. (We assume that an ODBC DSN has been created and is called dsnMyUserAcct).

```
proc sql;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
 create table WORK.MyData as
  select * from connection to MYDB(
   SELECT CUST_KEY,ITEM_KEY,DATE_KEY
```

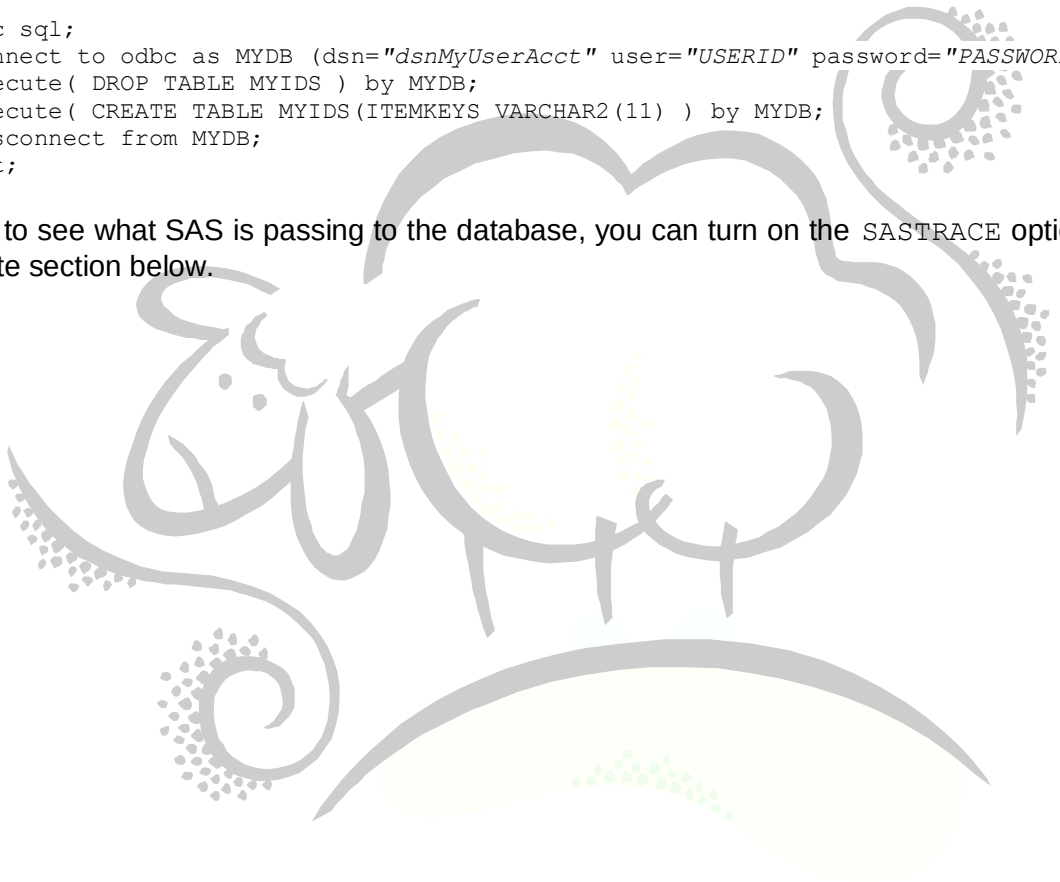
```
    FROM MY_DB_TABLE
    WHERE DATE_KEY BETWEEN DATE '2005-01-01'
                       AND DATE '2005-01-31'
  );
 disconnect from MYDB;
quit;
```

The code in red above is passed *directly* to the database without SAS getting involved. This means that no functions will be dropped and any subsetting you do will be taken into account by the Oracle database engine directly. Naturally, the resulting data is pulled back and stored into the SAS dataset MyData.

Note that you can create tables and indexes, drop tables and indexes, alter tables, and other database-specific things using the SAS EXECUTE() function. This allows you to run SQL Data Definition Language (DDL) on the database, but no rows of data will be returned, unlike the *SELECT * FROM CONNECTION TO MYDB* above. For example, you can drop a table and then create a table like this:

```
proc sql;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
 execute( DROP TABLE MYIDS ) by MYDB;
 execute( CREATE TABLE MYIDS(ITEMKEYS VARCHAR2(11) ) by MYDB;
 disconnect from MYDB;
quit;
```

If you want to see what SAS is passing to the database, you can turn on the `SASTRACE` option. This is described in a separate section below.



Using SAS/Access to Oracle to Access an Oracle Database

Here are some quick examples using SAS/Access to Oracle. Note that there are many other ways of doing the same thing. Be sure to replace the *USERID*, *PASSWORD* and *PATH* to your database below with your own username and password.

Note that the SQL code highlighted in red is passed *directly* to the database and, thus, SAS does not modify that SQL code.

1. Create a table in your user account:

```
/* Using PROC SQL */
proc sql noprint;
 connect to oracle as MYDB (user="USERID" password="PASSWORD" path="PATH" preserve_comments);

 execute(CREATE TABLE MYIDS (ITEM_KEY VARCHAR2(11))) by MYDB;

 disconnect from MYDB;
quit;

/* …or… Using SAS DATA step */
/* Assign a libname to the database */
libname dbMYDB oracle user="USERID" password="PASSWORD" path="PATH";
run;

data dbMYDB.MYIDS;
 length ITEM_KEY $ 11;
 ITEM_KEY="11111111111";output;
run;

libname dbMYDB clear;
run;
```

2. Insert data from a SAS dataset into a table in your user account:

```
/* Create your ITEMs - fake data */
data ITEMS(keep=ITEM_KEY);
 length ITEM_KEY $ 11;
 do I=1 to 1000;
  ITEM_KEY='11111111111';
  output;
 end;
run;

/* Assign a libname to the database */
libname dbMYDB oracle user="USERID" password="PASSWORD" path="PATH";
run;

/* Using PROC SQL, insert the data into the table MYIDS in your user account */
proc sql noprint;

 insert into dbMYDB.MYIDS(insertbuff=1000)
  SELECT ITEM_KEY
   FROM ITEMS;

quit;

/* …or… Using PROC DATASETS, insert the data into the table MYIDS in your user account */
proc datasets library=dbMYDB nolist;
 append base=dbMYDB.MYIDS(insertbuff=1000) data=work.ITEMS;
run;
quit;
```

```
/* Clear the libname statement */
libname dbMYDB clear;
run;
```

3. Read from the MYIDS table you created and print it off:

```
/* Using PROC SQL */
proc sql;
 connect to oracle as MYDB (user="USERID" password="PASSWORD" path="PATH" preserve_comments);
 select * from connection to MYDB (
  SELECT *
   FROM MYIDS
 );
 disconnect from MYDB;
quit;

/* …or… Using PROC PRINT */
/* Assign a libname to the database */
libname dbMYDB oracle user="USERID" password="PASSWORD" path="PATH";
run;

proc print data=dbMYDB.MYIDS;
run;

libname dbMYDB clear;
run;
```

4. Create a SAS data using the Oracle table MYIDS:

```
proc sql noprint;
 connect to oracle as MYDB (user="USERID" password="PASSWORD" path="PATH" preserve_comments);
  create table ThosePeskyITEMs as
   select * from connection to MYDB (
    SELECT *
     FROM MYIDS
   );
quit;
```

5. Granting the BSMITH user read access to the MYIDS table in your account:

```
proc sql;
 connect to oracle as MYDB (user="USERID" password="PASSWORD" path="PATH" preserve_comments);
  execute(GRANT SELECT ON MYIDS TO BSMITH) by MYDB;
 disconnect from MYDB;
quit;
```

6. Dropping a table when you are done:

```
/* Using PROC SQL */
proc sql noprint;
 connect to oracle as MYDB (user="USERID" password="PASSWORD" path="PATH" preserve_comments);

 execute(DROP TABLE MYIDS) by MYDB;

 disconnect from MYDB;
quit;

/* …or… using PROC DATASETS */
/* Assign a libname to the database */
```

```
libname dbMYDB oracle user="USERID" password="PASSWORD" path="PATH";
run;

proc datasets library=dbMYDB nolist;
 delete MYIDS;
run;
quit;

libname dbMYDB clear;
run;
```

7. Create a SAS dataset by reading from the table MY_DB_TABLE:

```
proc sql;
connect to oracle as MYDB (user="USERID" password="PASSWORD" path="PATH" preserve_comments);
 create table TinyTable as
  select * from connection to MYDB (
   SELECT *
    FROM MY_DB_TABLE
    WHERE ROWNUM<=10
  );
quit;
```

8. Create an index on the column ITEM in your MYIDS table:

```
proc sql;
 connect to oracle as MYDB (user="USERID" password="PASSWORD" path="PATH" preserve_comments);
  execute(CREATE INDEX IDX_ITEM ON MYIDS(ITEM_KEY)) by MYDB;
 disconnect from MYDB;
quit;
```

Using SAS/Access to OLEDB to Access an Oracle Database

Here are some quick examples using SAS/Access to OLEDB to access an Oracle database. Be sure the replace *USERID*, *PASSWORD*, *mydbserver*, and *mydbname* below with your own username, password, Oracle server name and Oracle database name.

Note that the SQL code highlighted in red is passed *directly* to the database and, thus, SAS does not get involved in these cases.

1. Create a table in your user account:

```
/* Using PROC SQL */
proc sql noprint;
 connect to oledb as MYDB (init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521");

 execute(CREATE TABLE MYIDS (ITEM VARCHAR2(11))) by MYDB;

 disconnect from MYDB;
quit;

/* …or… Using SAS DATA step */
/* Assign a libname to the database */
libname dbMYDB oledb init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521";
run;

data dbMYIDS;
 length ITEM_KEY $ 11;
 ITEM_KEY="11111111111";output;
run;

libname dbMYDB clear;
run;
```

2. Insert data from a SAS dataset into a table in your user account:

```
/* Create your ITEMs */
data ITEMS(keep=ITEM_KEY);
 length ITEM_KEY $ 11;
 do I=1 to 1000;
  ITEM_KEY='11111111111';
  output;
 end;
run;

/* Assign a libname to the database */
libname dbMYDB oledb init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521";
run;

/* Using PROC SQL, insert the data into the table MYIDS in your user account */
proc sql noprint;

 insert into dbMYIDS(insertbuff=1000)
  SELECT ITEM_KEY
   FROM ITEMS;

quit;

/* …or… Using PROC DATASETS, insert the data into the table MYIDS in your user account */
proc datasets library=dbMYDB nolist;
 append base=dbMYIDS(insertbuff=1000) data=work.ITEMS;
```

```
run;
quit;

/* Clear the libname statement */
libname dbMYDB clear;
run;
```

3. Read from the MYIDS table you created and print it off:

```
/* Using PROC SQL */
proc sql;
 connect to oledb as MYDB (init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521");
 select * from connection to MYDB (
  SELECT *
   FROM MYIDS
 );
 disconnect from MYDB;
quit;

/* …or… Using PROC PRINT */
/* Assign a libname to the database */
libname dbMYDB oledb init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521";
run;

proc print data=dbMYIDS;
run;

libname dbMYDB clear;
run;
```

4. Create a SAS data using the Oracle table MYIDS:

```
proc sql noprint;
 connect to oledb as MYDB (init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521");
  create table ThosePeskyITEMs as
   select * from connection to MYDB (
    SELECT *
     FROM MYIDS
   );
quit;
```

5. Granting the BSMTIH user read access to the MYIDS table in your account:

```
proc sql;
 connect to oledb as MYDB (init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521");
  execute(GRANT SELECT ON MYIDS TO BSMITH) by MYDB;
 disconnect from MYDB;
quit;
```

6. Dropping a table when you are done:

```
/* Using PROC SQL */
proc sql noprint;
 connect to oledb as MYDB (init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521");
```

```
 execute(DROP TABLE MYIDS) by MYDB;

 disconnect from MYDB;
quit;

/* …or… using PROC DATASETS */
/* Assign a libname to the database */
libname dbMYDB oledb init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521";
run;

proc datasets library=dbMYDB nolist;
 delete MYIDS;
run;
quit;

libname dbMYDB clear;
run;
```

7. Create a SAS dataset by reading from the table MY_DB_TABLE:

```
proc sql;
connect to oledb as MYDB (init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521");
 create table TinyTable as
  select * from connection to MYDB (
   SELECT *
    FROM MY_DB_TABLE
    WHERE ROWNUM<=10
  );
quit;
```

8. Create an index on the column ITEM in your MYIDS table:

```
proc sql;
 connect to oledb as MYDB (init_string="Provider=OraOLEDB.Oracle;SERVER=mydbserver;Data
Source=mydbname;USER ID=USERID;PASSWORD=PASSWORD;PORT=1521");
  execute(CREATE INDEX IDX_ITEM ON MYIDS(ITEM_KEY)) by MYDB;
 disconnect from MYDB;
quit;
```

Using SAS/Access to ODBC to Access an Oracle Database from Base SAS on Your Desktop

Here are some quick examples using SAS/Access to ODBC to access an Oracle Database via Base SAS installed on your desktop. Be sure the replace *DSN*, *USERID* and *PASSWORD* below with your own data source name, username and password.

Note that you will need to setup an ODBC connection to your user account on the Oracle database on your desktop. You can do this yourself from the "Data Sources (ODBC)" Applet (START…Control Panel…Administrative Tools…Data Sources(ODBC)). Please contact your Help Desk if you are having problems. Let's assume that your ODBC connection to your user account is called dsnMyUserAcct. This will be used in the examples below.

Also note that the SQL code highlighted in red is passed *directly* to the database and, thus, SAS does not get involved in these cases.

1. Create a table in your user account:

```
/* Using PROC SQL */
proc sql noprint;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);

 execute(CREATE TABLE MYIDS (ITEM_KEY VARCHAR2(11))) by MYDB;

 disconnect from MYDB;
quit;

/* …or… Using SAS DATA step */
/* Assign a libname to the database */
libname dbMYDB odbc (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
run;

data dbMYIDS;
 length ITEM_KEY $ 11;
 ITEM_KEY="11111111111";output;
run;

libname dbMYDB clear;
run;
```

2. Insert data from a SAS dataset into a table in your user account:

```
/* Create your ITEMs */
data ITEMS(keep=ITEM_KEY);
 length ITEM_KEY $ 11;
 do I=1 to 1000;
  ITEM_KEY='11111111111';
  output;
 end;
run;

/* Assign a libname to the database */
libname dbMYDB odbc (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
run;

/* Using PROC SQL, insert the data into the table MYIDS in your user account */
proc sql noprint;

 insert into dbMYIDS(insertbuff=1000)
  SELECT ITEM_KEY
   FROM ITEMS;

quit;
```

```
/* …or… Using PROC DATASETS, insert the data into the table MYIDS in your user account */
proc datasets library=dbMYDB nolist;
 append base=dbMYIDS(insertbuff=1000) data=work.ITEMS;
run;
quit;

/* Clear the libname statement */
libname dbMYDB clear;
run;
```

3. Read from the MYIDS table you created and print it off:

```
/* Using PROC SQL */
proc sql;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
 select * from connection to MYDB (
  SELECT *
   FROM MYIDS
 );
 disconnect from MYDB;
quit;

/* …or… Using PROC PRINT */
/* Assign a libname to the database */
libname dbMYDB odbc (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
run;

proc print data=dbMYIDS;
run;

libname dbMYDB clear;
run;
```

4. Create a SAS data using the Oracle table MYIDS:

```
proc sql noprint;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
  create table ThosePeskyITEMs as
   select * from connection to MYDB (
    SELECT *
     FROM MYIDS
   );
quit;
```

5. Granting the BSMITH user access to the MYIDS table in your account:

```
proc sql;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
  execute(GRANT SELECT ON MYIDS TO BSMITH) by MYDB;
 disconnect from MYDB;
quit;
```

6. Dropping a table when you are done:

```
/* Using PROC SQL */
proc sql noprint;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);

 execute(DROP TABLE MYIDS) by MYDB;
```

```
 disconnect from MYDB;
quit;

/* …or… using PROC DATASETS */
/* Assign a libname to the database */
libname dbMYDB odbc (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
run;

proc datasets library=dbMYDB nolist;
 delete MYIDS;
run;
quit;

libname dbMYDB clear;
run;
```

7. Create a SAS dataset by reading from the table MY_DB_TABLE:

```
proc sql;
connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
 create table TinyTable as
  select * from connection to MYDB (
   SELECT *
    FROM MY_DB_TABLE
    WHERE ROWNUM<=10
  );
quit;
```

8. Create an index on the column ITEM in your MYIDS table:

```
proc sql;
 connect to odbc as MYDB (dsn="dsnMyUserAcct" user="USERID" password="PASSWORD" autocommit=yes);
  execute(CREATE INDEX IDX_ITEM ON MYIDS(ITEM_KEY)) by MYDB;
 disconnect from MYDB;
quit;
```

Using the SASTRACE Option to Determine SQL Passed to the Database

If you would like to see what SQL code SAS is passing to the database – which is a good idea when you are developing your code – try turning on the `SASTRACE` option. This will place notes in your SAS Log file containing exactly what SAS passed to the database.

```
options sastrace=",,,d" sastracelog=saslog nostsuffix;
run;
```

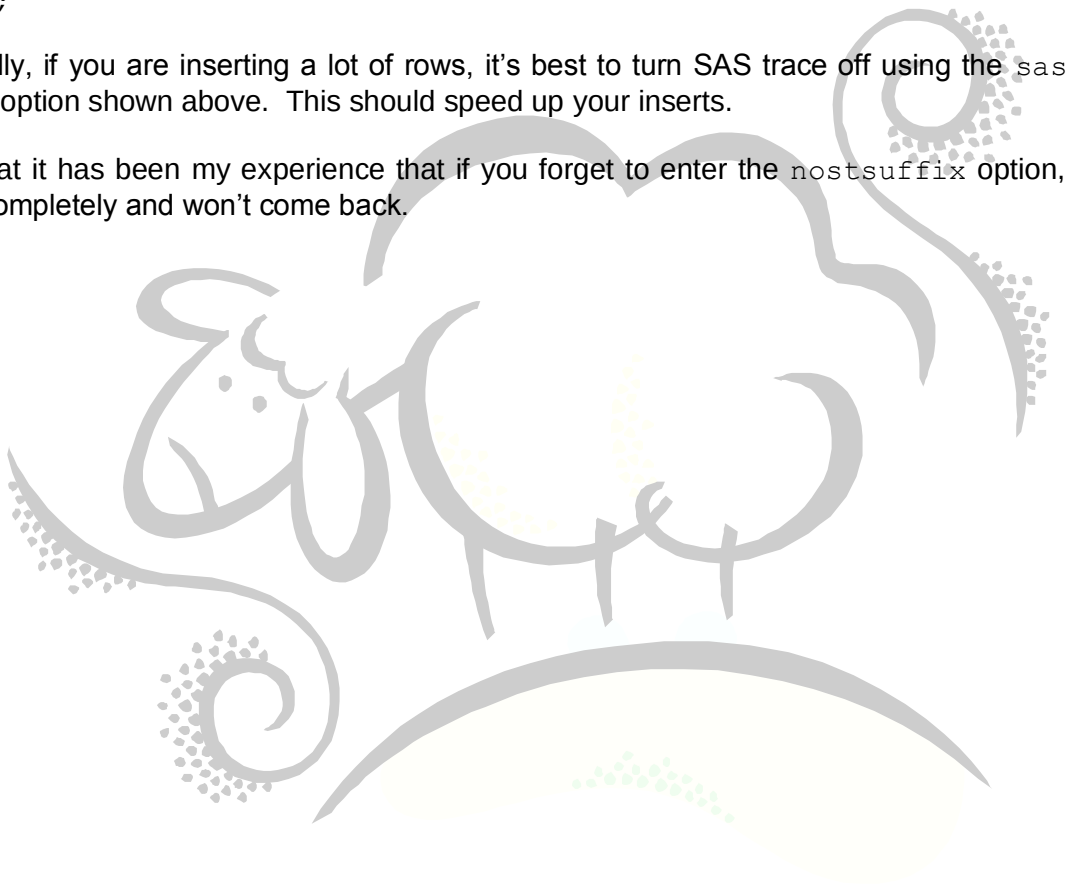
This will place the SAS tracing information in your SAS Log (`sastracelog=saslog`).

You can turn this off by using this:

```
options sastrace=none;
run;
```

Generally, if you are inserting a lot of rows, it's best to turn SAS trace off using the `sastrace=none` system option shown above. This should speed up your inserts.

Note that it has been my experience that if you forget to enter the `nostsuffix` option, your session stops completely and won't come back.



Using the INSERTBUFF=, READBUFF=, DBCOMMIT= and BUFFSIZE= Options

If you find that reading from or writing to a database table seems to be very slow, you may want to check out the INSERTBUFF=, READBUFF=, DBCOMMIT= and BUFFSIZE= options.

The options INSERTBUFF=, READBUFF= and DBCOMMIT= are available on the LIBNAME statement, whereas the BUFFSIZE= option is allowed in the connection parentheses on the PROC SQL CONNECT TO line.

INSERTBUFF specifies the number of rows to insert into a database table as a chunk. In many cases, this value defaults to 1, so it's a good idea to increase it.

READBUFF specifies the number of rows to read from a database table as a chunk.

DBCOMMIT specifies the number of rows to insert into a database table before forcing a COMMIT. Note that if the value of DBCOMMIT is lower than INSERTBUFF, then the value of DBCOMMIT overrides INSERTBUFF. I usually make them the same value.

BUFFSIZE is a general buffer size with a maximum of 32767. Below I specify 30000.

By default, INSERTBUFF, READBUFF and DBCOMMIT are set low. By changing these to higher values, you will be able achieve faster reads from and writes to the database. In the examples below, I've coded in my values based on my own trial-and-error. You should perform your own tests to see what values achieve the greatest performance in your environment. Note that setting these values huge is not the answer! There's limited memory on your computer and this plays a big factor in which values you will ultimately choose.

For example, here is the LIBNAME statement:

```
libname dbORA oracle user="USERNAME" pass="PASSWORD" path="PATH"
                     insertbuff=100000 readbuff=100000 dbcommit=100000;
run;
```

And here is the PROC SQL CONNECT TO line:

```
proc sql noprint noerrorstop;
 connect to oracle as ORADW (user="USERNAME" pass="PASSWORD" path="PATH"
buffsize=30000);
```

Using SAS/Access to OLEDB to Significantly Decrease Load Times into a SQL Server Database

You can significantly decrease your SAS dataset load times into a SQL Server database table by using the bulkload option on the SAS LIBNAME statement. Here is the new LIBNAME statement -- pointed to the *my-init-cat* database:

```
libname dbTOPAMAX oledb provider="SQLOLEDB.1" properties=('Integrated
Security'='SSPI' 'Persist Security Info'='False' 'Initial Catalog'='my_init-cat'
'Data Source'='my-db-datasrc') dbcommit=30000 insert_sql=yes bulkload=yes
oledb_services=no;
run;
```

You can play with the DBCOMMIT option, which indicates how many records to commit to the database at one time. Above, I specified dbcommit=30000, which will commit to the database after every 30000 rows of data. Naturally, you will have to

Creating a Comma-Delimited List using PROC SQL's SEPARATED BY Clause

Many SQL WHERE clauses make use of the IN() functionality. In order to use the IN() functionality, you must provide a comma-delimited list of values. This can be done using the trusty DATA _NULL_ statement, or a quicker way of accomplishing the same thing, as shown below, using PROC SQL:

```
DATA myDataSet;
 myVariable="A";OUTPUT;
 myVariable="B";OUTPUT;
 myVariable="C";OUTPUT;
RUN;

PROC SQL NOPRINT;
 SELECT myVariable
  INTO :myLIST separated by ","
  FROM myDataSet;
RUN;
%PUT ===>&myLIST.<===;
```

The results will be

```
===>A,B,C<===
```

If quotes are desired around the resulting values, then the following will work:

```
DATA myDataSet;
 myVariable="A";OUTPUT;
 myVariable="B";OUTPUT;
 myVariable="C";OUTPUT;
RUN;

PROC SQL NOPRINT;
 SELECT "'" || trim(left(myVariable)) || "'"
  INTO :myLIST separated by ","
  FROM myDataSet;
RUN;
%PUT ===>&myLIST.<===;
```

The result will be

```
===>'A','B','C'<===
```

This can be very useful for pass-through SQL to Oracle. If the number of selection criteria values is relatively small – there is an upper bound to the amount of IN-clause data that can be used with pass-through to Oracle – rather than joining tables, use an IN-clause if possible.

One way to subset data at the database using the technique shown above refers is shown in the following example:

```
DATA MYITEMS;
 LENGTH ITEM_KEY $ 11;
 ITEM_KEY="11111111111";OUTPUT;
 ITEM_KEY="22222222222";OUTPUT;
 ITEM_KEY="33333333333";OUTPUT;
RUN;

PROC SQL NOPRINT;
```

```
 SELECT DISTINCT "'" || TRIM(LEFT(ITEM_KEY)) || "'"
  INTO :MYITEMS SEPARATED BY ","
  FROM MYITEMS;

 CONNECT TO ORACLE AS MYDB (USER="USERNAME" PASS="PASSWORD" PATH="PATH");
 CREATE TABLE MY_SUBSET_DATA as
  SELECT *
   FROM CONNECTION TO MYDB
   (
    SELECT *
     FROM MY_DB_TABLE
     WHERE ITEM_KEY IN (&MYITEMS.)
   )
 ;
 DISCONNECT FROM MYDB;
QUIT;
```

And here is part of the SAS Log showing the SQL query passed to the database (when SASTRACE has been turned on):

```
DEBUG: PREPARE SQL statement:  78 1394917872 no_name 0 SQL
    SELECT * FROM MY_DB_TABLE WHERE ITEM_KEY IN
   ('11111111111','22222222222','33333333333') 79 1394917872 no_name 0 SQL
```

Notice that all of the ITEM_KEYs are passed directly to Oracle subsetting the data at the database rather than pulling all of the data back to your SAS session! As mentioned above, there is an upper-limit to the amount of data you can pass-through to Oracle in this manner. If you have a lot of data, it might be better to create a table and insert data into it rather than using the IN()-functionality.

HMMM….WHERE DID I PUT THAT TABLE?

Occasionally, you will forget what tables you've created or what indexes you've created on those tables. You can use the Oracle system table `ALL_TABLES` to search for tables that you've created, `ALL_TAB_COLUMNS` to find out the column names for a specific table, `ALL_INDEXES` to search for indexes created on a specific table, and `ALL_IND_COLUMNS` to search for indexes on the columns in a table. Below are the layouts for those tables (I have highlighted the most important fields):

ALL_TABLES

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
OWNER                                     NOT NULL VARCHAR2(30)
TABLE_NAME                                NOT NULL VARCHAR2(30)
TABLESPACE_NAME                                    VARCHAR2(30)
CLUSTER_NAME                                       VARCHAR2(30)
IOT_NAME                                           VARCHAR2(30)
PCT_FREE                                           NUMBER
PCT_USED                                           NUMBER
INI_TRANS                                          NUMBER
MAX_TRANS                                          NUMBER
INITIAL_EXTENT                                     NUMBER
NEXT_EXTENT                                        NUMBER
MIN_EXTENTS                                        NUMBER
MAX_EXTENTS                                        NUMBER
PCT_INCREASE                                       NUMBER
FREELISTS                                          NUMBER
FREELIST_GROUPS                                    NUMBER
LOGGING                                            VARCHAR2(3)
BACKED_UP                                          VARCHAR2(1)
NUM_ROWS                                           NUMBER
BLOCKS                                             NUMBER
EMPTY_BLOCKS                                       NUMBER
AVG_SPACE                                          NUMBER
CHAIN_CNT                                          NUMBER
AVG_ROW_LEN                                        NUMBER
AVG_SPACE_FREELIST_BLOCKS                          NUMBER
NUM_FREELIST_BLOCKS                                NUMBER
DEGREE                                             VARCHAR2(10)
INSTANCES                                          VARCHAR2(10)
CACHE                                              VARCHAR2(5)
TABLE_LOCK                                         VARCHAR2(8)
SAMPLE_SIZE                                        NUMBER
LAST_ANALYZED                                      DATE
PARTITIONED                                        VARCHAR2(3)
IOT_TYPE                                           VARCHAR2(12)
TEMPORARY                                          VARCHAR2(1)
SECONDARY                                          VARCHAR2(1)
NESTED                                             VARCHAR2(3)
BUFFER_POOL                                        VARCHAR2(7)
ROW_MOVEMENT                                       VARCHAR2(8)
GLOBAL_STATS                                       VARCHAR2(3)
USER_STATS                                         VARCHAR2(3)
DURATION                                           VARCHAR2(15)
SKIP_CORRUPT                                       VARCHAR2(8)
MONITORING                                         VARCHAR2(3)
CLUSTER_OWNER                                      VARCHAR2(30)
DEPENDENCIES                                       VARCHAR2(8)
```

ALL_TAB_COLUMNS

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
OWNER                                     NOT NULL VARCHAR2(30)
```

TABLE_NAME	**NOT NULL**	**VARCHAR2(30)**
COLUMN_NAME	**NOT NULL**	**VARCHAR2(30)**
DATA_TYPE		**VARCHAR2(106)**
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(30)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)
COLUMN_ID		NUMBER
DEFAULT_LENGTH		NUMBER
DATA_DEFAULT		LONG
NUM_DISTINCT		NUMBER
LOW_VALUE		RAW(32)
HIGH_VALUE		RAW(32)
DENSITY		NUMBER
NUM_NULLS		NUMBER
NUM_BUCKETS		NUMBER
LAST_ANALYZED		DATE
SAMPLE_SIZE		NUMBER
CHARACTER_SET_NAME		VARCHAR2(44)
CHAR_COL_DECL_LENGTH		NUMBER
GLOBAL_STATS		VARCHAR2(3)
USER_STATS		VARCHAR2(3)
AVG_COL_LEN		NUMBER
CHAR_LENGTH		NUMBER
CHAR_USED		VARCHAR2(1)
V80_FMT_IMAGE		VARCHAR2(3)
DATA_UPGRADED		VARCHAR2(3)

ALL_INDEXES

Name	Null?	Type
OWNER	**NOT NULL**	**VARCHAR2(30)**
INDEX_NAME	**NOT NULL**	**VARCHAR2(30)**
INDEX_TYPE		VARCHAR2(27)
TABLE_OWNER	**NOT NULL**	**VARCHAR2(30)**
TABLE_NAME	**NOT NULL**	**VARCHAR2(30)**
TABLE_TYPE		CHAR(5)
UNIQUENESS		VARCHAR2(9)
COMPRESSION		VARCHAR2(8)
PREFIX_LENGTH		NUMBER
TABLESPACE_NAME		VARCHAR2(30)
INI_TRANS		NUMBER
MAX_TRANS		NUMBER
INITIAL_EXTENT		NUMBER
NEXT_EXTENT		NUMBER
MIN_EXTENTS		NUMBER
MAX_EXTENTS		NUMBER
PCT_INCREASE		NUMBER
PCT_THRESHOLD		NUMBER
INCLUDE_COLUMN		NUMBER
FREELISTS		NUMBER
FREELIST_GROUPS		NUMBER
PCT_FREE		NUMBER
LOGGING		VARCHAR2(3)
BLEVEL		NUMBER
LEAF_BLOCKS		NUMBER
DISTINCT_KEYS		NUMBER
AVG_LEAF_BLOCKS_PER_KEY		NUMBER
AVG_DATA_BLOCKS_PER_KEY		NUMBER
CLUSTERING_FACTOR		NUMBER
STATUS		VARCHAR2(8)
NUM_ROWS		NUMBER
SAMPLE_SIZE		NUMBER
LAST_ANALYZED		DATE

```
DEGREE                                            VARCHAR2(40)
INSTANCES                                         VARCHAR2(40)
PARTITIONED                                       VARCHAR2(3)
TEMPORARY                                         VARCHAR2(1)
GENERATED                                         VARCHAR2(1)
SECONDARY                                         VARCHAR2(1)
BUFFER_POOL                                       VARCHAR2(7)
USER_STATS                                        VARCHAR2(3)
DURATION                                          VARCHAR2(15)
PCT_DIRECT_ACCESS                                 NUMBER
ITYP_OWNER                                        VARCHAR2(30)
ITYP_NAME                                         VARCHAR2(30)
PARAMETERS                                        VARCHAR2(1000)
GLOBAL_STATS                                      VARCHAR2(3)
DOMIDX_STATUS                                     VARCHAR2(12)
DOMIDX_OPSTATUS                                   VARCHAR2(6)
FUNCIDX_STATUS                                    VARCHAR2(8)
JOIN_INDEX                                        VARCHAR2(3)
```

ALL_IND_COLUMNS

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
INDEX_OWNER                               NOT NULL VARCHAR2(30)
INDEX_NAME                                NOT NULL VARCHAR2(30)
TABLE_OWNER                               NOT NULL VARCHAR2(30)
TABLE_NAME                                NOT NULL VARCHAR2(30)
COLUMN_NAME                                        VARCHAR2(4000)
COLUMN_POSITION                           NOT NULL NUMBER
COLUMN_LENGTH                             NOT NULL NUMBER
CHAR_LENGTH                                        NUMBER
DESCEND                                            VARCHAR2(4)
```

For example, to determine all of the indexes on the columns for a specific table owned by BSMITH, you can use ALL_IND_COLUMNS, like this:

```
SELECT TABLE_NAME,INDEX_NAME,SUBSTR(COLUMN_NAME,1,50) "COLUMN_NAME"
 FROM ALL_IND_COLUMNS
 WHERE TABLE_OWNER='BSMITH'
 ORDER BY 1,2,3
```

The results of this query for the BSMITH user are:

```
TABLE_NAME                     INDEX_NAME                     COLUMN_NAME
------------------------------ ------------------------------ -------------------------
ITEM_INFO                       IDX_ITEM                        ITEM
ITEM_INFO                       IDX_ITEM_STATUS                 ITEM
ITEM_INFO                       IDX_ITEM_STATUS                 STATUS
ITEM_INFO                       IDX_STATUS                      STATUS
```

This means there are three indexes on the table `ITEM_INFO`: `IDX_NDX`, `IDX_ITEM_STATUS`, and `IDX_STATUS`. The index `IDX_ITEM_STATUS` is a composite index involving two columns, `ITEM` and `STATUS`. The remaining two indexes are simple indexes involving only one column each. Since the column COLUMN_NAME in ALL_IND_COLUMNS is defined as a VARCHAR2(4000), I substringed COLUMN_NAME to a reasonable length of 50.

If you just want a list of the tables you have created in your own schema, try this:

```
SELECT TABLE_NAME
 FROM ALL_TABLES
 WHERE OWNER='BSMITH'
```

The results for the BSMITH user are:

```
TABLE_NAME
------------------------------
ITEM_INFO
TESTTAB1
```

Or, equivalently,

```
SELECT TABLE_NAME
 FROM USER_TABLES
```

The table USER_TABLES is the same as ALL_TABLES except it subsets the data based on your own schema (the one you are logged into at the time).

To get all of the columns for these tables, try this:

```
SELECT TABLE_NAME,COLUMN_NAME
 FROM ALL_TAB_COLUMNS
 WHERE OWNER='BSMITH'
 ORDER BY 1,2
```

The results for the BSMITH user are:

```
TABLE_NAME                     COLUMN_NAME
------------------------------ -------------
ITEM_INFO                      ITEM
ITEM_INFO                      ITEM_DESC
ITEM_INFO                      STATUS
TESTTAB1                       COL1
```

Finally, to get a list of only the indexes on a table (without the columns involved), try this:

```
SELECT TABLE_NAME,INDEX_NAME
 FROM ALL_INDEXES
 WHERE TABLE_OWNER='BSMITH'
 ORDER BY 1,2
```

```
TABLE_NAME                     INDEX_NAME
------------------------------ --------------------
ITEM_INFO                       IDX_ITEM
ITEM_INFO                       IDX_ITEM_STATUS
ITEM_INFO                       IDX_STATUS
```

SAS/Access and Oracle's Bulk Load Feature

As you have seen, you can insert data into an Oracle table using the INSERT statement in PROC SQL. Depending on your site, you can use Oracle's SQL*Loader to load data into your table much faster than an INSERT Statement.

First, you must be using at least Oracle 9i and the Oracle SQL*Net client software must be at least version 9 on your server or desktop computer. Please check with your local SAS or Oracle administrator for more information.

If the requirements above have been met, you can use SQL*Loader. Here is an example:

```
OPTIONS linesize=150 nocenter nodate sastrace=',,,d' sastraceloc=saslog;
RUN;

DATA ITEMS(KEEP=ITEM_KEY);
 LENGTH ITEM_KEY $ 11;
 DO I=1 TO 1000;
  ITEM_KEY='11111111111';
  OUTPUT;
 END;
RUN;

LIBNAME dbMYDB ORACLE USER="USERNAME" PASSWORD="PASSWORD" PATH="PATH";
RUN;

PROC SQL NOPRINT;

 INSERT INTO dbMYDB.MYITEMS(BULKLOAD=YES,BL_LOAD_METHOD=APPEND,BL_DIRECT_PATH=YES)
  SELECT ITEM_KEY
   FROM WORK.ITEMS;

QUIT;

LIBNAME dbMYDB CLEAR;
RUN;
```

Note that if your site did not meet the requirements, your SAS Log will have the following Oracle SQL*Loader error message in it:

```
ORA-02352: Direct path connection must be homogeneous
```

Despite the note in the SAS Log telling you that the data was loaded into your table, the error message above indicates that your data was **not** loaded.

Creating Excel Spreadsheets and Access Databases using SAS/Access to OLEDB

Note: You MUST create a blank Access database BEFORE using the method outlined below for Access databases.

You can create a new Excel spreadsheet or read from an existing Excel spreadsheet using SAS/Access to OLEDB in this manner.

This option may only work on 32-bit machines. The Microsoft Jet database driver is not available on 64-bit machines…although this may change in the future.

```
*-------------------------------------------------------------------------*;
* Libname to read from/write to a Microsoft Excel Spreadsheet.            *;
* If your sheet does NOT contain column names, change hdr=yes to hdr=no.  *;
*-------------------------------------------------------------------------*;
libname olexls oledb provider='Microsoft.Jet.OLEDB.4.0'
 properties=('data source'='LOCATION-OF-YOUR-EXCEL-WORKBOOK\WORKBOOK-NAME.xls')
 provider_string='Excel 8.0;hdr=yes';
run;

proc print data=olexls.'YOUR-SHEET-NAME'n;
run;

libname olexls clear;
run;

*-------------------------------------------------------------------------*;
* Libname to an Access Database.                                          *;
* LOCATION-OF-ACCESS-DATABASE can be an UNC or directory location.        *;
*-------------------------------------------------------------------------*;
libname olemdb oledb provider="Microsoft.Jet.OLEDB.4.0"
 properties=('data source'="LOCATION-OF-ACCESS-DATABASE\ACCESS-DATABASE.mdb");
run;

data SAS-DATASET-NAME;
 set olemdb.'ACCESS-TABLE-NAME'n;
run;

libname olemdb clear;
run;
```

Note that you can use Pass-Through SQL to read from and write to an Access database as well. Here is an example:

```
libname dbPROD01 oledb init_string="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
C:\MyAccessDB.mdb" INSERTBUFF=50000 DBCOMMIT=50000;
run;

proc sql noprint noerrorstop;
 connect to oledb as PROD01 (init_string="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\MyAccessDB.mdb");

 execute(DROP TABLE MYTABLE) by PROD01;

 execute(CREATE TABLE MYTABLE(COL1 BYTE,
                              COL2 BYTE,
                              COL3 TEXT(255),
                              COL4 SINGLE)) by PROD01;
```

```
 insert into dbPROD01.MYTABLE
  select COL1,COL2,COL3,COL4
   from work.MYTABLE;

 execute(CREATE INDEX IX_MYTAB ON MYTABLE(COL1,COL2)) by PROD01;

 disconnect from PROD01;
quit;
```

Support sheepsqueezers.com

If you found this information helpful, please consider supporting sheepsqueezers.com. There are several ways to support our site:

- ☐ Buy me a cup of coffee by clicking on the following link and donate to my PayPal account: Buy Me A Cup Of Coffee?.
- ☐ Visit my Amazon.com Wish list at the following link and purchase an item: http://amzn.com/w/3OBK1K4EIWIR6

Please let me know if this document was useful by e-mailing me at comments@sheepsqueezers.com.