Version 7 Enhancements to SAS/ACCESS ®Software

Vino Gona, SAS Institute Inc., Cary, NC   Jana Van Wyk, SAS Institute Inc., Cary, NC

Abstract

With the Version 7 release of the SAS® System, SAS/ACCESS software provides many important enhancements that allow SAS users to transparently integrate their Database Management Systems (DBMS) with their SAS business solutions.

New in Version 7, dynamic DBMS engines enable the SAS user to assign a SAS library that dynamically connects to the DBMS data server. SAS programs using the library can then directly list, read, update, delete and create DBMS tables. The creation of static access and view descriptors is no longer required. Dynamic engines are available in the Version 7 SAS/ACCESS interfaces for DB2®, ORACLE®, ODBC, INFORMIX®, SYBASE®, and INGRES®.

Additional features and performance improvements include support for table and column names up to 32 characters, support for case-sensitive DBMS object names, passing joins directly to the DBMS server for processing, updating data in a DBMS table via an SQL view of the table, more control over the use of DBMS indexes, locking and concurrency control, and updating descriptors using the ACCESS procedure.

The dynamic Data Step Interface, supported by the SAS/ACCESS interfaces to IMS and IDMS, has been enhanced in Version 7 to support the longer SAS names while still maintaining Version 6 compatibility. The SAS/ACCESS interface to IMS does not support dynamic access to the database; however, IMS descriptors now support long SAS variable names.

The IMPORT and EXPORT facility is also supported by most SAS/ACCESS interfaces. See the Reference section of this paper for more information about the IMPORT and EXPORT facility.

The ACCESS procedure, the DBLOAD procedure, PROC SQL Pass-Through, and the use of access and view descriptors are also supported in Version 7 releases of SAS/ACCESS software for Version 6 compatibility.

Most information in this paper is applicable to SAS/ACCESS software for DB2, ORACLE, ODBC, INFORMIX, SYBASE, and INGRES. This software is available on one or more of these platforms: Microsoft Windows 95, Microsoft Windows NT, OS/2®, Solaris, HP-UX, AIX, and MVS. See Appendix A for more detailed information about each SAS/ACCESS interface.

Introduction

This paper introduces some of the new features available in the Version 7 release of SAS/ACCESS software. These new features include

- **Dynamic DBMS Engines** (no access or view descriptors needed).
- **Support for all native, case-sensitive DBMS object names** up to 32 characters in length, including names which do not conform to valid SAS names.
- **Improved integration with SAS SQL views**, including embedded LIBNAME statements and updateable SQL views.
- **Performance improvements**, including passing joins between tables directly to the DBMS server and the use of DBMS indexes in queries.
- **Ability to customized when DBMS connections are issued** and options to control the kind of data locking used by the engine.
- **Interactive prompting for DBMS connection options**, like username and password.
- **An enhanced, fully-supported SQL Pass-Through Facility**
- **Ability to update descriptors** using the line mode interface to the ACCESS procedure.
- Full **Version 6 compatibility** for existing SAS programs using descriptors, the ACCESS procedure, the DBLOAD procedure, and SQL Pass-Through.
- The dynamic **Data Step Interface** for IMS and IDMS
- Additional Version 7 enhancements to SAS/ACCESS software

Examples using the SAS/ACCESS interfaces to DB2, ORACLE, and ODBC are used to illustrate the new features.

Dynamic DBMS Engines

With the Version 7 SAS System there is no longer a need to create access and view descriptors to read and manipulate data in a DBMS from within the SAS System. You can now directly use a SAS/ACCESS engine to assign a new SAS library that describes the location of the DBMS server and also defines how the SAS/ACCESS engine can connect to the DBMS server.

Connecting to the DBMS Server

One way to assign a SAS library using a SAS/ACCESS engine is to submit a LIBAME statement using options to describe how to connect to the DBMS server. When the library is assigned, a connection is made to the DBMS server, and the tables and views in the DBMS can be accessed using the library reference.

```
libname dblib oracle user=scott
         password=tiger path=orserver;
```

In the example above*, dblib* is the library reference, and *oracle* is the name of the dynamic DBMS engine in the SAS/ACCESS interface to ORACLE software. *User, password,* and *path* are the ORACLE engine LIBNAME statement options that provide the information needed to connect to the ORACLE server. By default, a connection to the DBMS is established when the LIBNAME statement is submitted.

Listing the Names of your DBMS Tables

After a SAS library is assigned, the DBMS connection can be used by the SAS System to get a list of ORACLE tables and views. This example uses the library reference *dblib* defined above with the DATASETS procedure.

```
proc datasets lib=dblib;
```

This statement causes the ORACLE engine to query the ORACLE system tables and retrieve all the ORACLE tables and views accessible to the ORACLE user named *scott*. The DATASETS procedure prints the following output to the SAS log.

```
Libref:        DBLIB
Engine:        ORACLE
Physical Name: orserver
User/Schema:   scott

#  Name        Memtype
1  DEPT        DATA
2  EMPLOYEES   DATA
3  INVENTORY   DATA
4  SALES       DATA
```

You can also use the SAS SQL dictionary tables, which are accessed through the predefined *dictionary* library reference, to list the names of the tables and views you can access from the ORACLE server. The following example shows a query against one of the SQL dictionary tables named *members* that lists the ORACLE tables accessible from the library named *dblib*.

```
proc sql;
select memname from dictionary.members
   where libname=”DBLIB”;

Member Name
-----------
DEPT
EMPLOYEES
INVENTORY
SALES
```

The SAS SQL dictionary table named *COLUMNS* can be used to find out the SAS metadata associated with the ORACLE table named *EMPLOYEES*. In this query the ORACLE table dynamically queries the ORACLE server to read the meatadata associated with the ORACLE table and then maps this metadata to the corresponding SAS System metadata.

```
proc sql;
select name, format
   from  dictionary.columns
   where libname=”DBLIB” and
   memname=”EMPLOYEES”;
```

Reading from a DBMS Table

Now that you can see the names of the ORACLE tables, you can look at the data in the table named *DEPT* using the PRINT procedure.

```
proc print data=dblib.dept; run;
```

The PRINT procedure displays this output in the SAS listing.

```
Obs     DEPTNO     DNAME      LOC
1       10         ACCOUNTING NEW YORK
2       20         RESEARCH   DALLAS
3       30         SALES      CHICAGO
4       40         OPERATIONS BOSTON
```

You can use the table *DEPT* in the SAS language just like a SAS data file by referencing it as a two level SAS data file named *dblib.dept*. The next example uses the CONTENTS procedure.

```
proc contents data=dblib.dept; run;
```

This is a partial listing of the output from this example.

```
Data Set Name:       DBLIB.DEPT
Member Type:         DATA
Engine:              ORACLE

Variable Type Len Format Informat Label
DEPTNO   Num    8     3.       3.  DEPTNO
DNAME    Char  14   $14.     $14.  DNAME
LOC      Char  13   $13.     $13.  LOC
```

Once again, just like the query above of the SQL dictionary table *COLUMNS*, when the CONTENTS procedure is submitted the ORACLE engine queries the ORACLE system tables to find information about the table *DEPT*.

Updating a DBMS Table

In addition to listing table names and reading rows from DBMS tables, the ORACLE engine can directly update the rows of a table. This example uses the SQL procedure to change a row in the table named *DEPT*.

```
proc sql;
update dblib.dept set loc=”Nashville”
   where deptno=10;
```

Creating a DBMS Table

Once a SAS library has been assigned using a DBMS engine, the DBMS tables can now be referenced in a SAS program as if they were SAS data files. So DBMS tables can be created in the same way a SAS data file is created. In this example the ORACLE table *NY_DEPT* is created by using the data step.

```
data dblib.ny_dept;
set dblib.dept(keep=dname deptno);
   where loc=”NEW YORK”;
```

Note that if the table *NY_DEPT* had already existed, the data step would fail with an error message. DBMS engines will not replace a DBMS table, even though the SAS native engine will replace a SAS data file.

DBMS tables can be created using the SAS language in the same way a SAS data file is created. This example uses the SQL procedure to create a DBMS table named *APR_SALES* from the data file *SALES* and renames the variable *S1* to *SALES1*.

```
proc sql;
create table apr_sales as
  select * from sasuser.sales
       (rename=(s1=sales1));
```

Deleting a DBMS table

Another example showing how SAS/ACCESS engines in Version 7 fully integrate access to your DBMS tables from within the SAS languages is that DBMS tables can be also be deleted, or dropped, from your DBMS schema. The following SAS code shows two of the ways in which you can easily delete DBMS tables from within the SAS System. In both examples the table named *NEWSALES* is deleted.

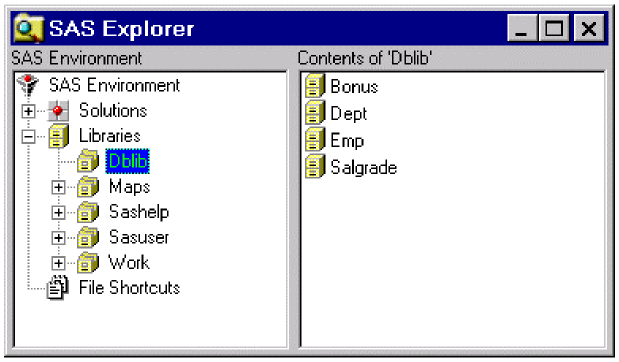
```
proc sql;
  drop table dblib.newsales; quit;

proc datasets lib=dblib nolist;
  delete newsales; run; quit;
```

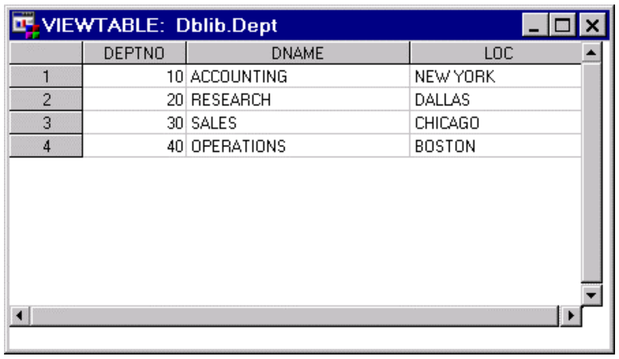
DBMS Dynamic Engines and the SAS Explorer

In Version 7 the SAS Explorer is the native Graphical User Interface (GUI) to the SAS System. The Explorer serves as a central access point from which you can easily manipulate SAS data files, like DBMS tables and views, with a graphical representation. You can create, copy, move, view, and delete DBMS tables and views, using the tree and list views of the SAS Explorer.

Clicking on a predefined DBMS library reference in the SAS Explorer's tree view causes the Explorer to dynamically query the DBMS server for a list of DBMS tables. This list is presented in the list view in the right-hand side of the SAS Explorer. After you click on the library named *DBLIB*, the windows on your screen will look like this.



Double-clicking on the table name in the list view in the right-hand side of the SAS Explorer will cause the DBMS engine to query the DBMS server, read, and display the data in the DBMS table.

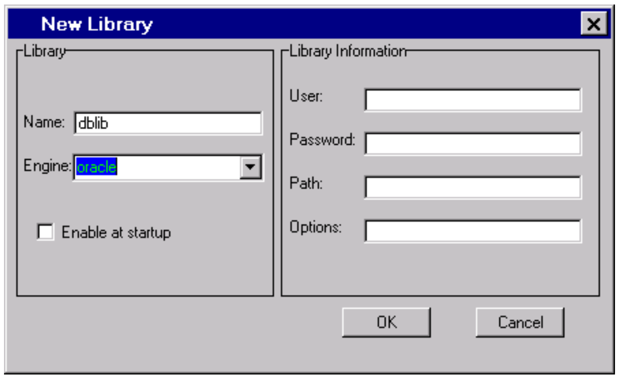

VIEWTABLE: Dblib.Dept

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

Assigning a DBMS Engine Library in the SAS Explorer

There are several ways to create a new SAS library from within the SAS Explorer windowing environment. You can use the *File* pull-down and select *New* and then *Library*, or you can click on *Libraries*, position the cursor on any library name, press the right mouse button, and then select *New->Library* from the dialog box.

After you click on the *OK* push button to create a new library, the New Library dialog window, as shown below, will appear. You can then type in your new library reference, the name of the engine you wish to use, and any option values that are needed by the engine when it assigns this library. The left-hand side of the New Library dialog drives the contents of the right hand side of the dialog. The right-hand side of the New Library dialog displays the names of the options that are relevant for the engine that has been selected in the left-hand side. So as you change the *ENGINE* field value, the right-hand side of the window will refresh with the list of appropriate options needed by that particular engine.



Name: dblib
Engine: oracle
☐ Enable at startup

Note the checkbox in the window named 'Enable at startup'. If this box is checked when the library is assigned, this library assignment will automatically be made every time you invoke the SAS System.

SAS and DBMS Naming Conventions

In Version 7 of the SAS System, the SAS naming rules for member and variable names have changed. The SAS System now allows member and variable names up to 32 characters in length. Valid characters are a-z, A-Z, 0-9, and the underscore character. The first character must be either an alphabetic character or an underscore. Also, SAS variable names are now case-sensitive. This means that when the SAS data file is created the exact case of the variable names is preserved. However, when the variable name is used in a SAS program the variable name can be typed in upper, lower, or mixed case.

These case-sensitive long names in SAS version 7 are very important and convenient to SAS/ACCESS software users since most Database Management Systems have similar rules for their DBMS object names. In Version 6 of SAS/ACCESS software, DBMS object names longer than eight characters had to be truncated and renamed for uniqueness. This restriction is lifted for most Version 7 SAS/ACCESS software (see Appendix A for the exceptions).

The default behavior of each SAS/ACCESS engine with regard to DBMS object names and case sensitivity is chosen to reflect the standards set by the DBMS product itself. For example, the SAS/ACCESS to DB2 engine operating in the MVS environment normalizes DB2 names to upper case by default but allows the user to override this default behavior.

This example creates the DB2 table named *PHONE_NUMBERS* with the columns *HOME_PHONE* and *WORK_PHONE*. Note that the names are uppercased-normalized by default.

```
libname mylib db2 ssid=user1;

data mylib.phone_numbers;
input home_phone $ work_phone $;
345-3456 555-4040
111-3333 222-4444
;
```

The CONTENTS procedure prints the following output to the SAS log.

```
Data Set Name: MYLIB.PHONE_NUMBERS
Member Type:   DATA
Engine:        DB2

# Variable   Type Len Pos Label
1 HOME_PHONE Char   8   0 HOME_PHONE
2 WORK_PHONE Char   8   8 WORK_PHONE
```

You can override the default behavior and create any valid DBMS table or column name by using library options supported by the dynamic DBMS engine. For example, the DB2 engine supports options named *PRESERVE_TAB_NAMES* and *PRESERVE_COL_NAMES*. When set to the value *YES,* these options preserve the case sensitivity of SAS member and variable names when a DBMS table is created and also preserve the case of table and column names when a DBMS table is read.

This example creates the DB2 table *PhoneNumbers* with the columns *HomePhone* and *WorkPhone*.

```
libname mylib db2 ssid=user1
preserve_tab_names=yes
preserve_col_names=yes;

data mylib.PhoneNumbers;
input HomePhone $ WorkPhone $;
345-3456 555-4040
111-3333 222-4444
;
```

The CONTENTS procedure output shows this information, which is the metatdata describing the table *PhoneNumbers*, as it is retrieved from the DB2 system tables.

```
Data Set Name: MYLIB.PhoneNumbers
Member Type:   DATA
Engine:        DB2

# Variable  Type Len Pos Label
1 HomePhone Char   8   0 HomePhone
2 WorkPhone Char   8   8 WorkPhone
```

Notice that the table and column names preserve the exact case used when the table was created in the SAS language using the data step.

Names with Special (non-alphanumeric) Characters

In addition to these new naming rules in the SAS language, the version 7 release of the SAS System supports DBMS object names that contain special characters, like the blank character. Most DBMS systems support the use of special characters in table and column names. Using table and column names with special characters in a SAS program will result in a syntax error, unless the names are used in the SQL procedure with the SAS SQL language.

To enable the support for names with special characters, the SQL procedure can be invoked with the new option *DQUOTE=ANSI*. When this option is used, table and column names specified in the SQL statements can be double-quoted to preserve any special characters. For example, this SQL program creates the table *"Table Name"*" with one column named *"My Column".*

```
proc sql dquote=ansi;
create table mylib."Table Name"
  ("My Column" integer);
insert into mylib."Table Name"
  ("My Column") values(100);
```

Note that this table cannot be referenced outside of the SQL procedure unless the table is enclosed in an SQL view, as this SAS code demonstrates.

```
proc sql dquote=ansi;
create view sasuser.db2view as
  select "My Column" as mycol
  from mylib."Table Name"; quit;

proc print data=sasuser.db2view;run;
```

When this SQL view named *db2view* is used in the SAS System, the DB2 column "My Column" is referenced as *mycol*. The PRINT procedure will use the view to display the data in the DB2 table named "*Table Name*". This output from the CONTENTS procedure shows the metadata of the view as stored in the SAS System.

```
proc contents data=sasuser.db2view;run;

Data Set Name: SASUSER.DB2VIEW
Member Type:   VIEW
Variables:     1
Engine:        SQLVIEW

#    Variable    Type    Len    Pos   Label
1    mycol       Num       8      0   My Column
```

SQL Views with Embedded LIBNAME Statements

Another enhancement in the version 7 release of the SAS System is the support of embedded LIBNAME statements within the definition of an SQL view. For SAS/ACCESS users this means that all the information needed to connect to the DBMS server and retrieve the DBMS data can be stored within the definition of an SQL view. The LIBNAME statements embedded in a view in this fashion will be assigned by PROC SQL any time the view is read and will be deassigned as soon as the view has been processed.

This type of SQL views are like the SAS/ACCESS view descriptors in that view descriptors also contain the DBMS connection information, in addition to the DBMS table metadata. The advantage of this type of view is that they can be referenced and used without dependence on any predefined SAS library reference.

This example creates the SQL view *MYVIEW,* which is a view of an ORACLE table *DEPT*. In this example, the ODBC engine is used to connect to an ORACLE data server using an ODBC

ORACLE driver. The ORACLE server is defined to ODBC with the name *orsrv*. Before storing the password in the SQL view, the SQL procedure encrypts the password.

```
proc sql;
create view sasuser.myview as
  select dname from dblib.dept
    using libname dblib odbc user=scott
          pw=tiger datasrc=orsrv;
```

When using the SQL view named *myview* in a SAS program, the library *dblib* is assigned using the ODBC engine. This causes a connection to ORACLE, via the ODBC driver, using the username and password information provided in the SAS SQL view. After the view is closed, the library is deassigned and the DBMS connection to ORACLE is ended.

There can be more than one LIBNAME statement embedded in a view. Here is an example with multiple libraries assigned from within an SQL view.

```
proc sql;
create view sasuser.myview as
select dname
  from odblib.dept t1, oralib.dept2 t2
  where t1.deptno = t2.deptno
  using
   libname odblib odbc user=scott pw=tiger
              datasrc=orsrv,
   libname oralib oracle user=myid pw=mypw
              path=orserver;
```

Updating DBMS Data using SQL Views

In Version 6 of the SAS System, all PROC SQL views were read-only, i.e., they could only be used to read data. In the Version 7 release of the SAS System, SQL views can be used to update the underlying data that the view references. However, for a PROC SQL view to be qualified as updateable, the following rules apply.

- The SQL view must be based on only one DBMS table or view. SQL views referring to more than one table or view can not be used to update data.
- The SQL view can not refer to DBMS data via the SQL Pass-Through facility.
- The SQL view may contain derived columns; however, the derived columns can not be updated.

In this example, a row in the ORACLE table *DEPT* is updated using an SQL view with a library defined by the ODBC dynamic engine.

```
proc sql;
create view sasuser.myview as
  select * from mylib.dept
    using libname mylib odbc user=scott
          pw=tiger datasrc=orsrv;
```

The SQL view can now be used to update the ORACLE table.

```
proc sql;
update sasuser.myview
  set loc=”Nashville” where deptno=10;
```

Passing Joins to the DBMS for Better Performance

In SAS versions prior to Version 7, an SQL query involving one or more DBMS tables (view descriptors in Version 6) was processed by the SQL engine as if the DBMS tables were individual SAS data files. The SQL procedure fetched all the rows from each individual DBMS table and then performed the join processing within the SAS software. This algorithm performed poorly, especially if each of the tables is large, and the SAS software and the DBMS data server communicate over a network.

A better way to perform this join is to let the DBMS server perform the join and return only the results of the join to the client, i.e., the SAS software. This is exactly what happens with the Version 7 release of the SAS System and will provide a major performance enhancement for many of your programs that perform joins across tables in a single DBMS.

For example, assume two large DBMS tables named *TABLE1* and *TABLE2* have a column named *DEPTNO,* and you want to retrieve the rows from an inner join of these tables where the *DEPTNO* value in *TABLE1* is equal to the *DEPTNO* value in *TABLE2*.

```
proc sql;
select tab1.deptno, dname from
  dblib.table1 tab1,
  dblib.table2 tab2
  where tab1.deptno = tab2.deptno
   using libname dblib oracle user=scott
       password=tiger path=myserver;
```

In version 7, this join between two tables within the same library will be detected by the SQL procedure and passed by the ORACLE engine directly to the DBMS server. The ORACLE engine will pass this query directly to ORACLE.

```
select tab1.deptno, dname
  from table1 tab1, table2 tab2
  where tab1.deptno = tab2.deptno;
```

The DBMS processes the inner join between the two tables and only the result rows are passed back to the SAS System. Both inner and outer joins between two or more DBMS tables are supported in this new enhancement.

Passing Where Clauses to the DBMS for Better Performance

Sometimes you may have a situation where you want to perform a join between a large DBMS table and a relatively small SAS data file. This code illustrates this functionality.

```
proc sql;
select tab1.deptno, loc from
  dblib.mytable tab1,
  sasuser.sasds tab2
  where tab1.deptno = tab2.deptno;
```

In this example, PROC SQL will retrieve all the rows from the DBMS table named *MYTABLE* and then apply the where clause in the SAS SQL procedure processing. This processing could be both cpu-intensive and I/O-intensive, if *MYTABLE* is large.

With version 7 SAS/ACCESS software there is an option that will enable the SQL procedure to pass a where clause to the DBMS so that only the rows that match the where clause are retrieved from the DBMS table. This can be a huge performance boost, especially in a networked environment.

```
libname dblib oracle user=scott pw=tiger;
proc sql;
select tab1.deptno, loc from
    dblib.table1(dbkey=deptno) tab1,
    sasuser.sasds tab2
    where tab1.deptno=tab2.deptno;
```

In this example, the *DBKEY* option tells the SQL procedure to pass the where clause in a form similar to "where deptno=<hostvariable>" to the DBMS engine. The engine then passes this optimized query with the where clause to the DBMS server. The <hostvariable> is substituted, one at a time, with the values of the *DEPTNO* variables from the observations of the smaller SAS data file *SASDS*.

Thus, only the rows that match the where clause are retrieved from the DBMS server. Without this option, PROC SQL retrieves all the rows from *TABLE1*. With the *DBKEY* option as specified in the example above, the SQL statement created by the ORACLE engine and passed to the DBMS look something like this:

```
select deptno, loc
  from table1
  where deptno=:hostvariable;
```

The <hostvariable> above has the value of the *DEPTNO* variable from the SAS data file *SASDS*. The number of selects issued will be equal to the number of rows in *SASDS*. A general rule of thumb to use is that the SAS data file should have relatively fewer rows compared to the DBMS table with which it is being joined.

The *DBKEY* option can also be used in a SAS data step to improve the performance when performing a data join using the *KEY*= option of the *SET* statement. This data step creates a new data file which joins the data file *KEYVALUES* with the DBMS table *MYTABLE* using the variable *DEPTNO* in the where clause issued by the DBMS engine.

```
data sasuser.new;
  set sasuser.keyvalues;
  set dblib.mytable(dbkey=deptno)
        key=dbkey;
```

[In addition to the *DBKEY* option, SAS/ACCESS software also allows the option *DBINDEX*=YES, *DBINDEX*=NO, and *DBINDEX*=<index name>. You can use the *DBINDEX* option if you know that the DBMS table you are using has one or more indexes using the column(s) on which the join is being performed. You can use the *DBINDEX*=<index name> option if you know the name of the index, or use *DBINDEX*=YES if you do not know the index name.]

Options to control DBMS Connections

Since the overhead of executing a connection or attach to a DBMS server can be very resource intensive and expensive, SAS/ACCESS engines now provide options with which you can control when the connection is made to your DBMS server. You can also control when a DBMS connection should be shared between more than one library. In general, a connection to a DBMS begins one transaction, so all statements issued in this connection are done within the context of the active transaction.

By default, a SAS/ACCESS engine executes a DBMS connection whenever a library is assigned, e.g., every time a LIBNAME statement is issued. Every reference to a table in this library that reads data from a table will share the read-only connection. However, if the SAS program attempts to update a DBMS table using this library reference, another separate connection is issued and all the updates occur in the new connection. These rules are followed so that there is one connection for all *READ-ONLY* transactions and separate, distinct connections for each *UPDATE* transaction.

The following LIBNAME statement, using the DB2 engine under the MVS system, executes a connection, which is a DB2 Call Attach facility command, to the DB2 DBMS server.

```
libname dblib db2 authid=user1;
```

If you want to assign more than one SAS library to your DBMS server, and if you will *not* be updating the DBMS tables, SAS/ACCESS engines provide an option to allow you to optimize the way the engine performs connections. Your SAS libraries can share a single *READ-ONLY* connection to the DBMS, if your library definitions use the *CONNECTION* option. This next example shows you how to use the *CONNECTION* option to control your connection together with the *ACCESS* option to specify your read-only data access requirements.

```
libname dblib1 db2 authid=user1
connection=globalread access=readonly;

libname dblib2 db2 authid=user1
connection=globalread access=readonly;
```

You can also delay *when* the connection to the DBMS occurs. If you do not want the connection to occur when the library is assigned, you can use the *DEFER* option when defining the library. If you indicate *DEFER*=YES, the connection to the DBMS will be issued by the engine the first time a DBMS table is used in the SAS program.

```
libname dblib1 db2 authid=user1 defer=yes;
```

You may want to use *DEFER=YES* if you are assigning DBMS engine libraries in an AUTOEXEC SAS program. The processing of the AUTOEXEC will be faster if you use *DEFER=YES* so that the DBMS connection is not made every time SAS is invoked.

Locking and Concurrency Control

Version 7 SAS/ACCESS engines support options that allow you to control some of the row, page, or table locking performed by the DBMS engine as it executes your SAS programs. For example, the ORACLE engine has the options *READLOCK_TYPE, UPDATELOCK_TYPE, READ_ISOLATION_LEVEL, UPDATE_ISOLATION_LEVEL* and *LOCKWAIT* options. By default, the ORACLE engine does not lock any data when reading rows from ORACLE tables. However, this behavior can be overridden by using the locking options. For example, if you want to lock the data pages of a table while the SAS system is reading the data and prevent other processes from updating the table, you can use the *READLOCK_TYPE* option.

```
proc sort
data=dblib.dbtable(readlock_type=table)
out=sasuser.mytable; by mycol;
```

In this example the ORACLE engine obtains a TABLE SHARE lock on the table so that the table can not be updated by other processes while your SAS program is reading it.

Locking options may be given as libname options also as shown in the example below.

```
libname dblib oracle user=scott
password=tiger path=orserver
updatelock_type=row;
```

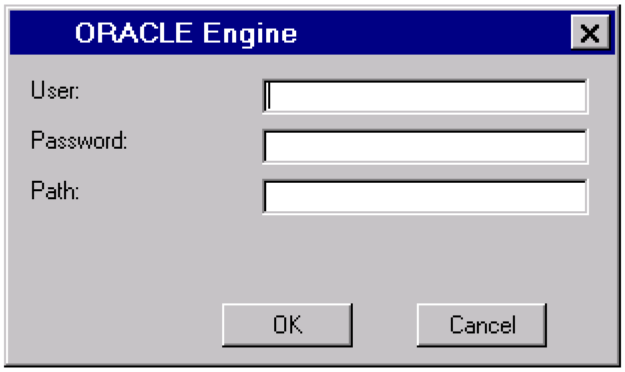
ORACLE will acquire row level locks on rows read for update on tables using this library reference.

Dynamic Prompting for DBMS Connection Options

In Version 6 of SAS/ACCESS software, users stored their DBMS connection information, like username and password, in SAS programs or (encrypted) in the contents of view and access descriptors. In version 7 of SAS/ACCESS software, a dynamic way of specifying DBMS connection information is supported. When a dynamic DBMS engine is used to assign a SAS library and the option *DBPROMPT=YES* is specified in the library definition, a dialog window appears and will prompt the user to interactively enter DBMS connection options, such as a username and password.

```
libname lib1 oracle dbprompt=yes defer=no;
```

When this LIBNAME statement is issued, a prompt window similar to the one shown below appears. The window will contain connection options that are valid for the DBMS engine being used.



This new feature of DBMS engines provides several advantages. It provides a more secure way to allow use of DBMS account passwords. The passwords do not have to be stored in a SAS program or in a descriptor file. Also, when a password, or even a username, changes, the SAS program itself does not have to change. Another advantage is that the same SAS program can be used by any username and password combination that is specified dynamically in the prompt dialog during the SAS job execution. It may also be appropriate to use the interactive use of connection options when moving from a test-only DBMS server to a production server. In this case, only the server name needs to be changed when the prompt dialog window appears.

Enhanced SQL Pass-Through Facility

The SQL Pass-Through facility continues to be a very important feature in the SAS/ACCESS software in the Version 7 release. This powerful feature provides a way for you to directly pass SQL statements to your DBMS when using special syntax in the SAS SQL procedure. This component is usually only supported by the SAS/ACCESS interfaces to relational databases. New in Version 7, case-sensitive SAS names up to 32 characters in length are supported when your DBMS names are mapped to internal SAS names for use within the SAS System.

If you wish to use the SQL Pass-Through views you created in Version 6 and continue to have your DBMS names automatically conform to Version 6 naming conventions, this is supported using the VALIDVARNAME global system option, as shown in the next example.

Suppose this SQL view was created in Version 6 using SAS/ACCESS Pass-Through facility.

```
proc sql;
connect to oracle
 (user=scott password=tiger path=orserv);
create view sasuser.mypass as
  select * from connection to oracle
  (select “employee_name” from emptable );
disconnect from oracle; quit;
```

If you use this SQL view in Version 7, the long, case-sensitive DBMS name is mapped to the internal SAS name as shown here in the PRINT procedure output.

```
proc print data=sasuser.mypass;run;

Obs        employee_name
1          Jones, Mary
2          Smith, Howard
```

However, if you want to use this SQL Pass-Through view in “version 6 compatibility mode” and use SAS names that were valid in Version 6, you can specify the options statement, as shown in this example.

```
options validvarname=v6;
proc print data=sasuser.mypass;

Obs        EMPLOYEE
1          Jones, Mary
2          Smith, Howard
```

Note that the column name was uppercased and truncated to eight characters when it was assigned to a SAS variable name. This is consistent with Version 6 naming conventions.

Line Mode Updating of Descriptors

In Version 7 of SAS/ACCESS software, the ACCESS procedure supports updating access and view descriptors via line mode syntax. The *UPDATE* statement is provided in the ACCESS procedure for modifications such as re-specifying a password or DBMS server name that may have changed since the descriptors were created.

```
proc access dbms=oracle;
  modify sasuser.ortable.view;
  orapw=”mine”;
  path=”NEW_SERVER”; run;
```

This ACCESS procedure code modifies an existing view descriptor to update the *ORAPW* and *PATH* options with new values.

SAS Version 6 Compatibility

The Version 7 release of SAS/ACCESS software provides complete compatibility for any existing SAS programs that were developed in the Version 6 SAS environment.

Any existing programs using the ACCESS procedure or the DBLOAD procedure will continue to run in Version 7. However, the interactive, full-screen interface to the ACCESS and DBLOAD procedures is not supported in Version 6.

All existing SQL Pass-Through views created in Version 6 and all existing access and view descriptors created with Version 6 are fully supported by the Version 7 release of SAS/ACCESS software.

Data Step Interface for IMS and IDMS

The Data Step Interface component of the IMS and IDMS engines provides you with a powerful tool for dynamically reading and updating your IMS or IDMS data using special SAS data step functions. New in Version 7, the Data Step Interfaces support SAS variable names up to 32 characters in length.

This example shows how the *INFILE* statement in the data step is used to define the IMS data to read. When the data step is submitted, the IMS data is read into the SAS data file *SASUSER.CUSTLIST*.

```
data sasuser.custlist;
  infile acctsam dli status=st pcbno=2;
  input @1 soc_sec_number $char11.
  input @12 customer_name $char40.;
  if st ^= ‘ ‘ then do;
    put_all_; abort; end;
```

The resulting data file can be used in any SAS program, like this PRINT procedure.

```
proc print data=sasuser.custlist;
  var customer_name;
  title2 ‘Customer List’; run;
```

Defining the “Scope” of a SAS Library when using a DBMS engine

Each dynamic DBMS engine supports library definition options to restrict or qualify the “scope”, or “schema”, of the tables that can be used via the library reference. For example, the DB2 engine supports the *AUTHID* and *LOCATION* library options, and the ORACLE engine supports *SCHEMA* and *DBLINK* library options. This example uses the ORACLE engine.

```
libname dblib oracle user=scott
password=tiger schema=ourgroup;

proc datasets lib=dblib;

#  Name         Memtype
1  OURTABLE     DATA
2  OURVIEW      DATA
```

The *dblib* library reference is scoped to the ORACLE schema named *ourgroup*. The DATASETS procedure lists only the tables and views that are accessible to the *ourgroup* schema. Any reference to a table using the library reference dblib will be passed to the ORACLE server as a qualified table name. For example, if a SAS program reads a table by specifying the SAS reference *dblib.ourtable,* the engine passes this query to the server.

```
select * from “ourgroup.ourtable”;
```

DBMS Connect and Disconnect Exits

There is a DBMS engine library definition option that can be defined to execute a DBMS command or stored procedure after a successful connection to the DBMS. Similarly, an option can be specified so that a DBMS command is executed in the connection immediately prior to the disconnection from the DBMS server.

```
libname dblib oracle
       user=scott password=tiger
       dbinitcmd=”EXEC MY_PROCEDURE”;
```

When this library is assigned, the engine connects to the DBMS and the stored procedure MY_PROCEDURE is executed. The stored procedure or command can not return any rows of data, and if the command fails, the library assignment fails with the DBMS error message displayed in the SAS log and assigned to the macro variables SYSDBRC and SYSDBMSG.

Character Variables up to 32,768 characters

Version 7 SAS allows character variables with a length up to 32,768 characters (32K bytes). Therefore SAS/ACCESS engines can now retrieve (or insert) SAS character variables with a length of up to 32,768 characters from (or into) DBMS columns of this size. Prior to Version 7 of the SAS System, the size limit of character variables was 200 characters.

SYSDBRC & SYSDBMSG SAS Macro Variables

SAS/ACCESS software now supports two new global system macro variables. The macro variables are SYSDBRC and SYSDBMSG and their values indicate the status of the last SAS/ACCESS engine statement executed. The value of SYSDBRC is 0 and the value of SYSDBMSG is a blank string, if the statement completed successfully. If an error was encountered, SYSDBRC has the DBMS error number or error code, and SYSDBMSG has the DBMS error message.

The macro variables supported in Version 6 for SQL Pass-Through facility are also supported in Version 7. The SQLXRC and SQLXMSG macro variables are available in SQL Pass-Through only.

```
libname dblib oracle user=bad pw=wrong;
```

This library assignment fails with an error because the connection to the DBMS fails due to invalid username and password. When the ORACLE engine attempts to connect to the ORACLE server, the server returns an error message that is retrieved by the engine. The engine updates the values of the global system macro variables *SYSDBRC* and *SYSDBMSG* with the values shown below.

```
%put &sysdbrc;
```

```
-1017
%put &sysdbmsg;
ORACLE: ORA-01017: invalid
username/password; logon denied.
```

These new SAS macro variables will be especially useful in SCL programs when the exact DBMS error code and error text can be retrieved via the macros and displayed to the user.

Summary

With Version 7 of the SAS System, accessing data in DBMS systems from the SAS System is easier than ever before. SAS users whose business and information solutions require the use of a variety of data sources from within the SAS system can now use the powerful dynamic DBMS engines provided by SAS/ACCESS software. The tools supported by SAS/ACCESS allow you to easily integrate your SAS system solutions with virtually any external data source. The Version 7 release of SAS/ACCESS software integrates with the SAS language to provide powerful and easy-to-use SAS system techniques to seamlessly read and update data in a variety of DBMS sources.

Acknowledgements

SAS and SAS/ACCESS are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. R indicates USA registration. DB2 and OS/2, are registered trademarks or trademarks of International Business Machines Corporation. ORACLE is a registered trademark or trademark of Oracle Corporation.

Other brand and product names are registered trademarks or trademarks of their respective companies.

References

For information about accessing external data from your PC:

Sanders, Roger E., "Accessing Data from your PC using Version 7 of the SAS System", Proceedings of the Twenty-Third Annual SAS Users Group International Conference. Cary, NC: SAS Institute Inc., 1998.

For information about general SAS I/O Enhancements:

Beatrous, Steve & Clifford, Billy, "Sometimes You Do Get What You Want: SAS I/O Enhancements in Version 7", Proceedings of the Twenty-Third Annual SAS Users Group International Conference. Cary, NC: SAS Institute Inc., 1998

For information about SAS/ACCESS software for Version 6:

SAS Institute Inc., *SAS/ACCESS Software for Relational Databases: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc., 1994, 196 pp.

Appendix A.

SAS/ACCESS interfaces and their components.

These are the individual features available in SAS/ACCESS software products. SAS/ACCESS interfaces contain a subset of this list of features.

- The engine: **Dynamic Engine**. SAS libraries can be dynamically assigned and metadata about tables is dynamically returned to the SAS System. Tables can be listed, read, updated, deleted, and created.
- The engine: the **SQL Pass-Through** facility. The engine supports the SQL procedure statements CONNECT, EXECUTE, SELECT, and DISCONNECT. The native DBMS SQL language is used to directly query the DBMS server.
- The engine: the **Data Step Interface.** This engine component provides access to the DBMS by surfacing the DBMS data to the SAS program via data step functions.
- The **DBLOAD procedure** creates, and appends to, DBMS tables. Any SAS data file can be used as the input data to the DBLOAD procedure. This procedure supports line mode syntax.
- The engine: the **view descriptor** interface. The ACCESS procedure is used to create static descriptors that contain metadata about a table. The descriptor is used to read and update data in a table. Only SAS names up to 8 characters are supported.
- The **ACCESS procedure** creates access and view descriptors that are customized views of the data in a DBMS table that SAS programs use to read and update the tables. This procedure supports line mode syntax.
- The **IMPORT and EXPORT** facility. This engine component provides a point-and-click style interface to import external file formats into SAS data files and to export SAS data files to an external file format.

SAS/ACCESS interface to DB2

The interface supports the Dynamic Engine, the SQL Pass-Through facility, the ACCESS procedure, the view descriptor interface, and the DBLOAD procedure. The SAS/ACCESS interface to DB2 is supported on MVS.

SAS/ACCESS interface to DB2 for Common Servers

The interface supports the Dynamic Engine, the SQL Pass-Through facility, and the DBLOAD procedure. The SAS/ACCESS interface to DB2 for Common Servers is supported on Microsoft Windows 95, Microsoft Windows NT, OS/2, Solaris, HP-UX, and AIX.

SAS/ACCESS interface to ODBC

The interface supports the Dynamic Engine, the SQL Pass-Through facility, the IMPORT/EXPORT facility, and the DBLOAD procedure. SAS/ACCESS to ODBC is available on Windows 95, Windows NT, OS/2, Solaris, HP-UX, and AIX.

SAS/ACCESS interface to ORACLE

The interface supports the Dynamic Engine, the SQL Pass-Through facility, the ACCESS procedure, the view descriptor interface, and the DBLOAD procedure. SAS/ACCESS to ORACLE is supported on Windows 95, Windows NT, OS/2, Solaris, HP-UX, AIX, and MVS.

SAS/ACCESS interface to INFORMIX

The interface supports the Dynamic Engine and the SQL Pass-Through facility. SAS/ACCESS to INFORMIX is available on Solaris, HP-UX, and AIX.

SAS/ACCESS interface to INGRES

The interface supports the Dynamic Engine, the SQL Pass-Through facility, the ACCESS procedure, the view descriptor interface, and the DBLOAD procedure. SAS/ACCESS to INGRES is supported on Solaris, HP-UX, and AIX.

SAS/ACCESS interface to SYBASE

The interface supports the Dynamic Engine, the SQL Pass-Through facility, the ACCESS procedure, the view descriptor interface, and the DBLOAD procedure. SAS/ACCESS to SYBASE is available on Windows 95, Windows NT, OS/2, Solaris, HP-UX, and AIX.

SAS/ACCESS interface to IMS

The interface supports the ACCESS procedure, the view descriptor interface, and the Data Step Interface. SAS/ACCESS to IMS is available on MVS.

SAS/ACCESS interface to IDMS

The interface supports the ACCESS procedure, the view descriptor interface, and the Data Step Interface. SAS/ACCESS to IDMS is available on MVS.

Check with your SAS representative for more product functionality and availability details.

Author Information

Vino Gona, SAS Institute Inc., 100 SAS Campus Drive, Cary, NC 27513, (919) 677-8000, email: sasvxg@unx.sas.com

Jana Van Wyk, SAS Institute Inc., 100 SAS Campus Drive, Cary, NC 27513, (919) 677-8000, email: sasjvw@unx.sas.com