CHAPTER 2

SAS Names and Support for DBMS Names

Introduction

Beginning in Version 7 of SAS software, SAS naming conventions have been enhanced to allow longer names for SAS data sets and SAS variables. The conventions also allow case-sensitive or mixed case names for SAS data sets and variables.

The following SAS language elements can now be up to 32 characters in length:

- members of SAS libraries, including SAS data sets, data views, catalogs, catalog entries, and indexes
- variables in a SAS data set
- macros and macro variables.

The following SAS language elements remain unchanged with a maximum length of 8 characters:

- librefs and filerefs
- SAS engine names and passwords
- names of SAS/ACCESS access descriptors and view descriptors (in order to maintain compatibility with Version 6 names)
- variable names in SAS/ACCESS access descriptors and view descriptors.

For a complete description of the new SAS naming conventions, see the *SAS Language Reference: Dictionary* .

The following *libref.dataset* shows the longer name for the data set MYDB.TEMP_EMPLOYEES_QTR4_2000. Likewise, a variable name can be longer and defined in mixed case, such as **Q4HireDates**.

When you specify mixed case or case-sensitive names in SAS code, SAS displays the names as you have specified them. In this example, the SAS variables, **Flight** and **dates**, are defined in mixed case:

```
options nodate linesize=64;

data test;
```

```
  input Flight $3. +3 dates date9.;
  format dates date9.;
datalines;
114   01MAR2000
202   01MAR2000
204   01MAR2000
;

proc print data=test (keep=FLIGHT DATES);
run;
```

Output 2.1 Mixed Case Names Displayed in Output

```
           SAS System

     Obs    Flight        dates

      1      114      01MAR2000
      2      202      01MAR2000
      3      204      01MAR2000
```

When the TEST data set is output, the *variable names are stored as they are defined*, instead of automatically being stored in uppercase. However, when SAS processes the names, it can process them as FLIGHT and DATES.

Note: Because of the way that SAS processes names, it recognizes variable names regardless of the case in which they were created. For example, if you were to use PROC DATASETS to rename the **`Flight`** variable, the procedure would recognize **`Flight`** even if you input it as **`flight`** or **`FLIGHT`**. However, the new variable name is stored as the mixed-case name **`All_flights`**. △

```
proc datasets library=work memtype=data;
   modify test;
   rename flight=All_flights;
run;
```

Output 2.2 SAS Log Showing a Renamed Variable

```
20   proc datasets library=work memtype=data;
                     -----Directory-----

          Libref:            WORK
          Engine:            V8
          Physical Name:     /tmp/SAS_xxxxxxxxabc
          File Name:         /tmp/SAS_xxxxxxxxefg
          Inode Number:      84111
          Access Permission: rwxr-xr-x
          Owner Name:        marie
          File Size (bytes): 1024


                             File
         #  Name  Memtype    size  Last modified
         ---------------------------------------------
         1  TEST  DATA      16384   11MAY1999:18:38:31
21   modify test;
22   rename flight=All_flights;
NOTE: Renaming variable flight to All_flights.
23   run;
```

SAS Name Literals

A SAS *name literal* is a name token that is expressed as a quoted string, followed by the letter **n**. Name literals enable you to use special characters or blanks that are not otherwise allowed in SAS names when you specify a SAS data set or variable. Name literals are especially useful for expressing database column and tables names that contain special characters.

Name literals are subject to certain restrictions:

- You can use a name literal only for SAS variable and data set names, statement labels, and DBMS column and table names.
- You can use name literals only in a DATA step or in the SQL procedure.
- If a name literal contains any characters that are not allowed when VALIDVARNAME=V7, then you must set the system option to VALIDVARNAME=ANY. For details on using the VALIDVARNAME= system option, see "VALIDVARNAME" on page 63.

Examples of name literals are

- **data mydblib.'My Staff Table'n; ... run;**
- **data Budget_for_1999;**
  input '$ Amount Budgeted'n 'Amount Spent'n ...run;

SAS/ACCESS LIBNAME and PROC SQL Options

Using the SAS/ACCESS LIBNAME statement and PROC SQL options, SAS software can handle table and column names in DBMSs that are case-sensitive or *non-standard for SAS*. Non-standard names includes those with blank spaces or special characters (such as @, #, %) that are not allowed in SAS names. The following list briefly describes these options. See your DBMS chapter for information about how SAS processes your DBMS-specific names.

PRESERVE_COL_NAMES=YES | NO

is an option on the SAS/ACCESS LIBNAME statement. If you specify YES, this option preserves spaces, special characters, and mixed case in DBMS column names . The default value for this option is DBMS-specific. See for more information about this option.

Specify the alias PRESERVE_NAMES=YES | NO, if you plan to specify both the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options in your LIBNAME statement. Using this alias saves you some time when coding.

You can use the DATA step to read data from multiple data sets, in this example, two DBMS tables. This example merges data from the two DBMS tables, STAFF and SUPERV, and writes it to the SAS data set WORK.COMBINED.

```
options linesize=120 nodate;

libname mydblib oracle user=karin password=haggis
  path='airhrdata' schema=airport
  preserve_col_names=yes;

data combined;
  merge mydblib.staff mydblib.superv(in=super
    rename=(supid=idnum));
  by idnum;
  if super;
run;

proc print data=combined (outobs=10);
  title "Supervisor Information";
run;
```

Note: The PRESERVE_COL_NAMES=YES LIBNAME option retains the case of the column names from the DBMS–in this example, uppercase—when creating the corresponding SAS variable names. △

Partial output for this example is shown:

Output 2.3 Reading Data from Multiple DBMS Tables

```
                              Supervisor Information

     OBS  IDNUM  LNAME          FNAME       CITY     STATE     HPHONE     JOBCAT

       1  1106   MARSHBURN      JASPER   STAMFORD     CT    203/781-1457    PT
       2  1118   DENNIS         ROGER    NEW YORK     NY    718/383-1122    PT
       3  1126   KIMANI         ANNE     NEW YORK     NY    212/586-1229    TA
       4  1352   RIVERS         SIMON    NEW YORK     NY    718/383-3345    NA
       5  1385   RAYNOR         MILTON   BRIDGEPORT   CT    203/675-2846    ME
       6  1401   ALVAREZ        CARLOS   PATERSON     NJ    201/732-8787    TA
       7  1405   DACKO          JASON    PATERSON     NJ    201/732-2323    SC
       8  1417   NEWKIRK        WILLIAM  PATERSON     NJ    201/732-6611    NA
       9  1420   ROUSE          JEREMY   PATERSON     NJ    201/732-9834    ME
      10  1431   YOUNG          DEBORAH  STAMFORD     CT    203/781-2987    FA
```

In the following example, you use the ORACLE PAYROLL table to create a new ORACLE table, PAY1, and then print it. Both the PRESERVE_COL_NAMES=YES and the PROC SQL DQUOTE=ANSI options are used to preserve the case and non-standard characters in the column names. Notice that you do not need to quote the column aliases in order to preserve the mixed case. You only need *double* quotes when the column name has non-standard characters or blanks.

By default, the SAS/ACCESS engine for ORACLE uses the database's rules for setting the case of table and column names. Therefore, even though the new ORACLE table name, **pay1**, is created in lowercase in this example, ORACLE stores the name in uppercase as **PAY1**. How table and column names are stored is DBMS-specific; see your DBMS chapter or DBMS documentation for more information.

```
options linesize=120 pagesize=60 nodate;

libname mydblib oracle user=yao password=cary path='ora8_servr'
  schema=hrdept preserve_col_names=yes;

proc sql dquote=ansi;
create table mydblib.pay1 as
  select idnum as "ID #", sex, jobcode, salary,
   birth as BirthDate, hired as HiredDate
       from mydblib.payroll
    order by birth;
```

```
title "Payroll Table with Revised Column Names";
select * from mydblib.pay1;
quit;
```

Recall from the description of how SAS processes columns on page 10, that SAS recognizes a column name, regardless of how it was created. Therefore, in this example, SAS recognizes the **jobcode** column name, whether you specify it in your SAS code as lowercase, mixed case, or uppercase. In the ORACLE PAYROLL table, the SEX, JOBCODE, and SALARY columns were created in uppercase, and therefore, they retain this case in the new table, PAY1, unless you rename them.

A partial output from the example is shown:

Output 2.4 DBMS Table Created with Non-Standard and Standard Column Names

```
                 Payroll Table with Revised Column Names

ID #  SEX  JOBCODE    SALARY             BirthDate            HiredDate
--------------------------------------------------------------------------
1118  M    PT3         11379    16JAN1944:00:00:00    18DEC1980:00:00:00
1065  M    ME2         35090    26JAN1944:00:00:00    07JAN1987:00:00:00
1409  M    ME3         41551    19APR1950:00:00:00    22OCT1981:00:00:00
1401  M    TA3         38822    13DEC1950:00:00:00    17NOV1985:00:00:00
1890  M    PT2         91908    20JUL1951:00:00:00    25NOV1979:00:00:00
1777  M    PT3          9630    23SEP1951:00:00:00    21JUN1981:00:00:00
1404  M    PT2         91376    24FEB1953:00:00:00    01JAN1980:00:00:00
```

PRESERVE_TAB_NAMES=YES | NO

is an option on the SAS/ACCESS LIBNAME statement. If you specify YES, this option preserves blank spaces, special characters, and mixed case in DBMS table names. The default value for this option is DBMS-specific. See for more information about this option.

In the following example, you use PROC PRINT to print the DBMS table, PAYROLL. Because the DBMS table was created in uppercase and you set the PRESERVE_TAB_NAMES=YES option, you must specify the table name in uppercase in your code. A partial output follows the example.

```
options nodate linesize=64;
libname mydblib oracle user=yao password=cary path='ora8_servr'
preserve_tab_names=yes;

proc print data=mydblib.PAYROLL;
  title 'PAYROLL Table';
run;
```

Output 2.5 DBMS Table with a Case-Sensitive Name

```
                                  PAYROLL Table

Obs    IDNUM    SEX    JOBCODE    SALARY                 BIRTH                 HIRED

  1    1919      M       TA2       34376      12SEP1960:00:00:00      04JUN1987:00:00:00
  2    1653      F       ME2       35108      15OCT1964:00:00:00      09AUG1990:00:00:00
  3    1400      M       ME1       29769      05NOV1967:00:00:00      16OCT1990:00:00:00
  4    1350      F       FA3       32886      31AUG1965:00:00:00      29JUL1990:00:00:00
  5    1401      M       TA3       38822      13DEC1950:00:00:00      17NOV1985:00:00:00
```

If you had omitted the PRESERVE_TAB_NAMES= option or set it to NO in this example, you could have specified the DBMS table name in lowercase.

In the next example, you create a PROC SQL view based on a DBMS table that you created in the previous example on page 12. Because you set PRESERVE_TAB_NAMES=YES in the following example, the name of the **PAY1** table is case-sensitive. When you also use the PRESERVE_COL_NAMES=YES option, you can rename the columns as well.

```
options nodate linesize=64;

libname mydblib oracle user=yao password=cary path='ora8_servr'
preserve_tab_names=yes preserve_col_names=yes;

proc sql dquote=ansi outobs=5;
create view work.jobcodes as
  select "ID #" as EmpID, sex, salary
    from mydblib.PAY1
    where Jobcode in ('TA2','TA3');

proc print data=work.jobcodes;
title 'By Jobcode TA2 or TA3';
run;
```

To simplify your coding, you could have also used the alias PRESERVE_NAMES=YES instead of listing both of the options on the LIBNAME statement.

A partial output for the example is shown:

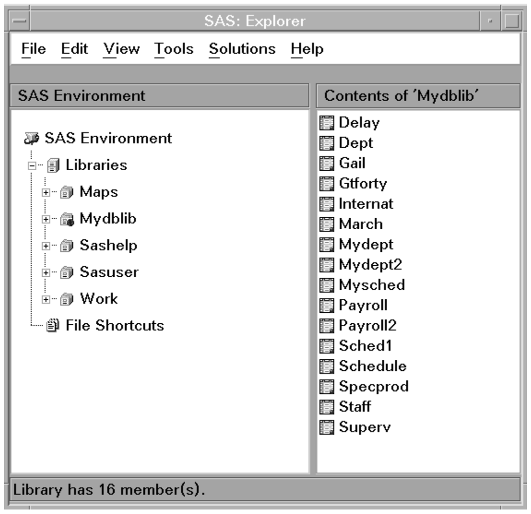
Output 2.6 PROC SQL View Created from a Case-Sensitive DBMS Table Name

```
           By Jobcode TA2 or TA3

                 Emp
          Obs    ID       SEX    SALARY

            1    1401      M      38822
            2    1639      F      40260
            3    1480      F      39583
            4    1017      M      40858
            5    1876      M      39675
```

The SAS Explorer window has replaced the Access window in Version 7 and later. In the next example, you submit the SAS/ACCESS LIBNAME statement with the PRESERVE_TAB_NAMES=NO option and then open the SAS Explorer window. The resulting window lists the ORACLE tables and views that are referenced by the **Mydblib** libref. Notice that 16 members are listed and that all of the member names are in the case (initial capitalization) that is set by the Explorer window.

```
libname mydblib oracle user=jyoti pass=tiger
  preserve_tab_names=no;
```

Display 2.1 SAS Explorer Window Listing DBMS Tables and Views



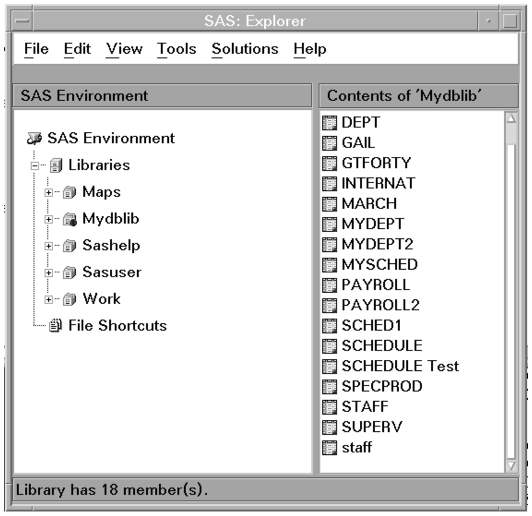
Delay
Dept
Gail
Gtforty
Internat
March
Mydept
Mydept2
Mysched
Payroll
Payroll2
Sched1
Schedule
Specprod
Staff
Superv

Library has 16 member(s).

In the next example, you submit the SAS/ACCESS LIBNAME statement with the PRESERVE_TAB_NAMES=YES and then open the SAS Explorer window. This time, you see a different listing of the ORACLE tables and views referenced by the **`Mydblib`** libref. Notice that there are 18 members listed, including one that is in lowercase and one that has a name separated by a blank space. Because of the LIBNAME option, SAS displays the tables names in the exact case in which they were created.

```
libname mydblib oracle user=jyoti pass=tiger
  preserve_tab_names=yes;
```

Display 2.2 SAS Explorer Window Listing Case-Sensitive DBMS Tables and Views



DEPT
GAIL
GTFORTY
INTERNAT
MARCH
MYDEPT
MYDEPT2
MYSCHED
PAYROLL
PAYROLL2
SCHED1
SCHEDULE
SCHEDULE Test
SPECPROD
STAFF
SUPERV
staff

Library has 18 member(s).

DQUOTE=ANSI | SAS

is a PROC SQL option. This option specifies whether PROC SQL treats values within *double* quotes as a character string or as a column name or table name.

When you specify DQUOTE=ANSI, your SAS code can refer to DBMS names that contain characters and spaces that are not allowed by SAS naming conventions.

In the next example, you create a DBMS table that is specified in double quotes and has a blank in its name, **`International Delays`**. Both of the preserve-names LIBNAME options are specified using the options' alias, PRESERVE_NAMES=YES.

```
options linesize=64 nodate;

libname mydblib oracle user=orjan pass=mypw path='airdata'
   schema=airport preserve_names=yes;

proc sql dquote=ansi;
  create table mydblib."International Delays" as
    select int.flight as "FLIGHT NUMBER", int.dates,
            del.orig as ORIGIN,
           int.dest as DESTINATION, del.delay
    from mydblib.INTERNAT as int, mydblib.DELAY as del
    where int.dest=del.dest and int.dest='LON';
quit;

proc sql dquote=ansi outobs=10;
title "International Delays";
  select * from mydblib."International Delays";
```

Notice that you use single-quotes to specify the data value for London (**`int.dest='LON'`**) in the WHERE clause. Because of the preserve-name LIBNAME options, using double-quotes would cause SAS to interpret this data value as a column name.

Output 2.7 DBMS Table with Non-Standard Column Names

```
                     International Delays

FLIGHT
NUMBER                 DATES  ORIGIN  DESTINATION     DELAY
-----------------------------------------------------------
219       01MAR1998:00:00:00  LGA     LON                18
219       02MAR1998:00:00:00  LGA     LON                18
219       03MAR1998:00:00:00  LGA     LON                18
219       04MAR1998:00:00:00  LGA     LON                18
219       05MAR1998:00:00:00  LGA     LON                18
219       06MAR1998:00:00:00  LGA     LON                18
219       07MAR1998:00:00:00  LGA     LON                18
219       01MAR1998:00:00:00  LGA     LON                18
219       02MAR1998:00:00:00  LGA     LON                18
219       03MAR1998:00:00:00  LGA     LON                18
```

See Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software," on page 65 and the SQL Procedure chapter in the *SAS Procedures Guide* for more information about the DQUOTE= option.

In the next example, you query the DBMS table and use a label to change the **`FLIGHT NUMBER`** column name to a standard SAS name, **`Flight_Number`**. A label—enclosed in single quotes—changes the name only in the output. Because this column name and the table name (**`International Delays`**) each have a space in their names, you have to enclose the names in double-quotes. A partial output follows the example.

```
options linesize=64 nodate;

libname mydblib oracle user=orjan pass=mypw path='airdata'
   schema=airport preserve_names=yes;

proc sql dquote=ansi outobs=5;
title "Query from International Delays";
  select "FLIGHT NUMBER" label='Flight_Number', dates, delay
    from mydblib."International Delays";
```

Output 2.8 Query Renaming a Non-Standard Column to a Standard SAS Name

```
   Query from International Delays

Flight_
Number                DATES     DELAY
-------------------------------------
219       01MAR1998:00:00:00       18
219       02MAR1998:00:00:00       18
219       03MAR1998:00:00:00       18
219       04MAR1998:00:00:00       18
219       05MAR1998:00:00:00       18
```

See the SAS naming conventions and "VALIDVARNAME" on page 63 for more information.

DBMS Column Names to SAS Variable Names

The SAS system option VALIDVARNAME=V7 is the default value for your SAS session unless you set this option to a different value. (VALIDVARNAME=V7 applies to Version 7 and later of SAS software.) Therefore, to change the standard SAS rules for names, you must set one of the following options: VALIDVARNAME=ANY, PRESERVE_TAB_NAMES=YES, PRESERVE_COL_NAMES=YES, or PROC SQL DQUOTE=ANSI.

If the aforementioned options are not set, the following rules apply when you map DBMS column names to SAS variable names:

- Characters that are not standard in SAS names (such as @ and #) that appear in DBMS column names are changed to underscores in SAS variable names. For example, the DBMS column name **MY$DEPT** becomes SAS variable name **MY_DEPT**.
- SAS makes DBMS column names into unique SAS variable names by appending a number (starting with 0) to the variable name when they are changed to conform with SAS rules. For example, DBMS column names **MY$DEPT**, **My$Dept**, and **my$dept** become SAS variable names **MY_DEPT**, **MY_Dept0**, and **MY_DEPT1**.

The following two tables describe how SAS processes DBMS names when it is retrieving DBMS data. This information applies generally to the DBMS names; see your DBMS chapter for possible exceptions. See "Naming Examples" on page 19 for examples that illustrate the different kinds of naming actions and defaults.

Table 2.1 DBMS Column to SAS Variable Names When Reading DBMS Data

If your DBMS *column* name is a...	...and you want this SAS variable name...	...Then use these LIBNAME, PROC SQL, or System Options [1]
Case-sensitive DBMS column name, such as **Flight**	Default SAS variable name (uppercase), such as **FLIGHT**	**preserve_col_names=no**
DBMS column name with characters that are not valid in SAS names, such as **My$Flight**	Default SAS variable name (uppercase) where an underscore replaces the invalid characters, such as **MY_FLIGHT**	**preserve_col_names=no**
Case-sensitive DBMS column name, such as **Flight**	Case-sensitive SAS variable name, such as **Flight**	**preserve_col_names=yes**
DBMS column name with characters that are not valid in SAS names, such as **My$Flight**	Case-sensitive SAS variable name where an underscore replaces the invalid characters, such as **My_Flight**	**preserve_col_names=yes**
DBMS column name with characters that are not valid in SAS names, such as **My$Flight**	Nonstandard, case-sensitive SAS variable name, such as **My$Flight**	**proc sql dquote=ansi** and **preserve_col_names=yes** or, in a DATA or PROC step, use a SAS name literal such as **'My$Flight'n** and **preserve_col_names=yes validvarname=any**

1 These options might not be required. Default values for these options are DBMS-specific.

Table 2.2 *DBMS Table to SAS Data Set Names When Reading DBMS Data*

If your DBMS *table* name is a ...	...And you want this SAS data set name...	...Then use these LIBNAME, PROC SQL, or System Options[1]
Default DBMS table name, such as **STAFF**	Default SAS data set or member name (uppercase), such as **STAFF**	**preserve_tab_names=no**
Case-sensitive DBMS table name, such as **Staff**	Case-sensitive SAS data set, such as **Staff**	**preserve_tab_names=yes**
DBMS table name with characters that are not valid in SAS names, such as **All$Staff**	Nonstandard, case-sensitive SAS data set name, such as **All$Staff**	**proc sql dquote=ansi** and **preserve_tab_names=yes** or, in a DATA step or PROC, use a SAS name literal such as **'All$Staff'n** and **preserve_tab_names=yes**

1 These options might not be required. Default values for these options are DBMS-specific.

SAS Variable Names to DBMS Column Names

The following two tables describe how SAS variable names are handled when you use SAS/ACCESS software to create DBMS objects such as tables and views. This information applies generally; see your DBMS chapter for possible exceptions. See "Naming Examples" on page 19 for examples that illustrate the different kinds of naming actions and defaults.

Table 2.3 *SAS Data Set to DBMS Column Names*

If the SAS *variable* name as input is ...	...And you want this DBMS column name...	...Then use these LIBNAME, PROC SQL, or System Options[1]
Any SAS variable name, such as **`Miles`**	Default DBMS column name (normalized to follow the DBMS's naming conventions), such as **`MILES`**	**`preserve_col_names=no`**
A case-sensitive SAS variable name, such as **`Miles`**	Case-sensitive DBMS column name, such as **`Miles`**	**`preserve_col_names=yes`**
A SAS variable name with characters that are not valid in a normalized SAS name, such as **`Miles-to-Go`**	Case-sensitive DBMS column name that matches the SAS name, such as **`Miles-to-Go`**	**`proc sql dquote=ansi`** and **`preserve_col_names=yes`** or, in a DATA or PROC step, use a SAS name literal and **`preserve_col_names=yes`** **`validvarname=any`**

1 These options might not be required. Option default values are DBMS-specific.

Table 2.4 *SAS Data Set to DBMS Table Names*

If the SAS *data set* name as input is...	...And you want this DBMS table name...	...Then use these LIBNAME or PROC SQL Options[1]
Any SAS data set name, such as **`Payroll`**	Default DBMS table name (normalized to follow the DBMS's naming conventions), such as **`PAYROLL`**	**`preserve_tab_names=no`**
A case-sensitive SAS data set name, such as **`Payroll`**	Case-sensitive DBMS table name, such as **`Payroll`**	**`preserve_tab_names=yes`**
A case-sensitive SAS data set name with characters that are not valid in a normalized SAS name, such as **`Payroll-for-QC`**	Case-sensitive DBMS table name that matches the SAS name, such as **`Payroll-for-QC`**	**`proc sql dquote=ansi`** and **`preserve_tab-names=yes`** or, in a DATA or PROC step, use a SAS name literal and **`preserve_tab_names=yes`**

1 These options might not be required. Option default values are DBMS-specific.

Naming Examples

In this example, you create a simple table to test for yourself how the options work. To use name literals, you must specify the SAS system option VALIDVARNAME=ANY. Notice that you print the new DBMS table using PROC SQL because name literals work only with PROC SQL and the DATA step.

```
options ls=64 validvarname=any nodate;

libname mydblib oracle user=yao password=cary path='ora8servr'
preserve_col_names=yes preserve_tab_names=yes ;
```

```
data mydblib.'Sample Table'n;
  'EmpID#'n=12345;
  Lname='Chen';
  'Salary in $'n=63000;

proc sql;
title "Sample Table";
select * from mydblib.'Sample Table'n;
```

Output 2.9 DBMS Table to Test Column Names

```
          Sample Table

                    Salary
   EmpID#  Lname      in $
  -------------------------
    12345  Chen      63000
```

DBMS column and table names that contain characters or blanks that are not valid in SAS cannot be specified directly in a SAS DATA step or procedure, except if you are using:

- the DQUOTE=ANSI option in PROC SQL

or

- SAS name literals written as

  '*string*'n

Therefore, you must first rename these kinds of DBMS names as standard SAS names in a PROC SQL or data set view, and then reference that view in a DATA step or SAS procedure.

In the following example, notice that the LIBNAME statement is embedded in the PROC SQL view. Output follows the example.

```
libname mysaslib 'SAS-data-library';

proc sql dquote=ansi;
create view mysaslib.sampleview as
  select "EmpID#" as Empid, "Salary in $" as Salary
  from mydblib."Sample Table"
using libname mydblib oracle user=karin
    password=haggis path='ora8servr'
    preserve_col_names=yes preserve_tab_names=yes;

proc print data=mysaslib.sampleview;
  title 'Sample View';
run;
```

Output 2.10 PROC SQL View to Test Column Names

```
      Sample View

   Obs    Empid    Salary

     1    12345     63000
```

For more information about embedded libnames in PROC SQL views, see the SQL Procedure chapter in the *SAS Procedures Guide*.

You can then drop your sample DBMS table and PROC SQL view by using a PROC SQL DROP statement. Notice that the VALIDVARNAME=ANY option must be set in order for you to specify a name literal in the DROP statement:

```
options validvarname=any nodate;
libname mysaslib 'SAS-data-library';
libname mydblib oracle user=yao password=cary path='ora8servr'
preserve_tab_names=yes;

proc sql;
drop table mydblib.'Sample Table'n;
drop view mysaslib.sampleview;
quit;
```

In this example, you use PROC SQL to create a new DBMS table based on data from other DBMS tables. By using PRESERVE_COL_NAMES=YES, you preserve the case-sensitivity of the aliased column names. A partial output is displayed after the code.

```
libname mydblib oracle user=shella password=moiri
  path='hrdata99' schema=personnel preserve_col_names=yes;

proc sql;
  create table mydblib.gtforty as
  select lname as LAST_NAME,
         fname as FIRST_NAME,
         salary as ANNUAL_SALARY
  from mydblib.staff a,
          mydblib.payroll b
  where (a.idnum eq b.idnum) and
        (salary gt 40000)
  order by lname;

proc print noobs;
  title 'Employees with Salaries over $40,000';
run;
```

Output 2.11 Updating DBMS Data

```
       Employees with Salaries over $40,000

                                           ANNUAL_
   LAST_NAME          FIRST_NAME          SALARY

   BANADYGA           JUSTIN               88606
   BAREFOOT           JOSEPH               43025
   BRADY              CHRISTINE            68767
   BRANCACCIO         JOSEPH               66517
   CARTER-COHEN       KAREN                40260
   CASTON             FRANKLIN             41690
   COHEN              LEE                  91376
   FERNANDEZ          KATRINA              51081
```

In the next example, you create a temporary SAS data set that has case-sensitive names. You define your LIBNAME statement and then use a SAS DATA step to create the new DBMS table, **`College-Hires-1999`**. Because you are using a DATA step to create the DBMS table, you must specify the table name as a name literal and specify the PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options (in this case, by using the alias PRESERVE_NAMES=YES) .

```
options validvarname=any nodate;

data College_Hires_1999;
  input IDnum $4. +3 Lastname $11. +2
    Firstname $10. +2 City $15. +2
    State $2.;
datalines;
3413    Schwartz     Robert       New Canaan     CT
3523    Janssen      Heike        Stamford       CT
3565    Gomez        Luis         Darien         CT
;

libname mydblib oracle user=shella password=moiri
  path='hrdata99' schema=hrdept
  preserve_names=yes;

data mydblib.'College-Hires-1999'n;
  set College_Hires_1999;

proc print;
title 'College Hires in 1999';
run;
```

Output 2.12 DBMS Table with Case-Sensitive Table and Column Names

```
                        College Hires in 1999

Obs   IDnum    Lastname       Firstname     City               State

  1   3413     Schwartz       Robert        New Canaan           CT
  2   3523     Janssen        Heike         Stamford             CT
  3   3565     Gomez          Luis          Darien               CT
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Software for Relational Databases: Reference, Version 8

ISBN 1–58025–558–2

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999