Introduction

This chapter describes options that you specify in the LIBNAME statement to associate a SAS libref with a DBMS database, schema, server, or group of tables and views.

Using Librefs that Refer to DBMS Data

When you use the LIBNAME statement to associate a libref with DBMS data, you might observe some behavior that differs from that of normal SAS librefs. Because these librefs refer to DBMS objects, such as DBMS tables and views, they are stored in the format of your DBMS, which differs from the format of normal SAS data sets. It is helpful to keep this in mind when you access and work with DBMS data.

For example, you can sort the observations in a normal SAS data set and store the output to another data set. However, in a DBMS, sorting data often has no effect on how it is stored. Because you cannot depend on your data to be sorted in the DBMS, you must sort the data at the time of query, by using an ORDER BY clause in PROC SQL, a BY statement in the DATA step, the DBCONDITION=LIBNAME option described in this chapter, or another method. When you sort DBMS data, the results might also vary, depending on whether your DBMS places data with NULL values (which are translated in SAS to missing values) at the beginning or end of the result set.

When you use librefs that refer to DBMS data with SAS functions, some functions might return a value that differs from what is returned when you use the functions with normal SAS data sets. For example, the PATHNAME function might return a blank value. This does not indicate that the libref is invalid, as it does with a normal SAS libref, but that there is no pathname associated with the libref.

Usage of some functions might also vary. For example, the LIBNAME function can accept an optional SAS-data-library argument. When you use the LIBNAME function to assign or deassign a libref that refers to DBMS data, you omit this argument. For full details on how to use these functions, see the SAS Language Reference: Dictionary.
SAS/ACCESS LIBNAME Statement

Associates a SAS libref with a DBMS database, schema, server, or group of tables and views.
Valid: Anywhere

Syntax

1. LIBNAME libref SAS/ACCESS-engine-name
   <SAS/ACCESS-engine-connection-options>
   <SAS/ACCESS-engine-LIBNAME-options>;

2. LIBNAME libref CLEAR | _ALL_ CLEAR;

3. LIBNAME libref LIST | _ALL_ LIST;

Arguments

libref

is any SAS name that serves as an alias to associate the SAS System with a
database, schema, server, or group of tables and views.

When you are disassociating a currently-assigned libref or when listing attributes
with the LIBNAME statement, specify a libref that was previously assigned with a
LIBNAME statement.

SAS/ACCESS-engine-name

is the SAS/ACCESS engine name for your DBMS, such as oracle or db2. SAS/
ACCESS engines are implemented differently in different operating environments.
See your DBMS-specific documentation for your engine's name. The engine name is
required.

CLEAR
disassociates one or more currently assigned librefs.

Specify libref to disassociate a single libref. Specify _ALL_ to disassociate all
currently assigned librefs.

_ALL_
specifies that the CLEAR or LIST argument applies to all currently-assigned librefs.

LIST
writes the attributes of one or more SAS/ACCESS libraries or SAS data libraries to
the SAS Log.

Specify libref to list the attributes of a single SAS/ACCESS library or SAS data
library. Specify _ALL_ to list the attributes of all libraries that have librefs in your
current session.
SAS/ACCESS-engine-connection-options

are options that you specify to connect to a particular database; these options are
different for each database. If the connection options contain characters that are not
allowed in SAS names, enclose the values of the options in quotation marks. On
some DBMSs, if you specify the appropriate system options or environment variables
for your database, you can often omit the connection options. See your DBMS-specific
documentation for details.

SAS/ACCESS-LIBNAME-options

are options that apply to the processing of objects and data in a DBMS, such as its
tables or indexes. For example, the PRESERVE_COL_NAMES= option enables you
to specify whether to preserve spaces, special characters, and mixed case in DBMS
column names. Support for many of these options is DBMS specific.

Some SAS/ACCESS LIBNAME options have the same names as SAS/ACCESS
engine data set options. When you specify an option in the LIBNAME statement, it
applies to objects and data that are referenced by the libref. A SAS/ACCESS data set
option applies only to the data set on which it is specified. If a like named option is
specified in both the SAS/ACCESS engine LIBNAME statement and after a data set
name (which references a DBMS table or view), the SAS System uses the value that
is specified later, on the data set name. For more information, see “SAS/ACCESS
Data Set Options” on page 39.

Details

1 Using Data from a DBMS  You can use a LIBNAME statement to read from and
write to a DBMS table or view as though it were a SAS data set. The LIBNAME
statement associates a libref with a SAS/ACCESS engine to access tables or views in a
DBMS. The SAS/ACCESS engine enables you to connect to a particular DBMS and to
specify a DBMS table or view name in a two-level SAS name.

For example, in MYDBLIB.EMPLOYEES_Q2, MYDBLIB is a SAS libref that points
to a particular group of DBMS objects, and EMPLOYEES_Q2 is a DBMS table name.
When you specify MYDBLIB.EMPLOYEES_Q2 in a DATA step or procedure, you
dynamically access the DBMS table. Version 7 of the SAS System supports reading,
updating, creating, and deleting DBMS tables dynamically.

2 Disassociating a Libref from a SAS Data Library  To disassociate or clear a libref
from a DBMS, use a LIBNAME statement, specifying the libref (for example,
MYDBLIB) and the CLEAR option as follows:

   libname mydblib CLEAR;

You can clear a single specified libref or all current librefs.

The database engine will disconnect from the database and close any free threads or
resources that are associated with that connection.

3 Writing SAS Data Library Attributes to the SAS Log  Use a LIBNAME statement to
write the attributes of one or more SAS/ACCESS libraries or SAS data libraries to the
SAS log. Specify libref to list the attributes of a single SAS/ACCESS library or SAS
data library, as follows:

   libname mydblib LIST;

Specify _ALL_ to list the attributes of all libraries that have librefs in your current
session.
SAS/ACCESS LIBNAME Options

When you specify any of the following options on the LIBNAME statement, the option is applied to all objects (such as tables and views) in the database that the libref represents. These options can be used in the SAS/ACCESS interfaces that support the SAS/ACCESS LIBNAME functionality:

ACCESS= on page 26
CONNECTION= on page 26
CONNECTION_GROUP= on page 28
DBCONINIT= on page 29
DBCONTERM= on page 30
DBINDEX= on page 31
DBLIBINIT= on page 31
DBLIBTERM= on page 32
DBPROMPT= on page 33
DEFER= on page 34
DIRECT_SQL= on page 34
PRESERVE_COL_NAMES= on page 34
PRESERVE_TAB_NAMES= on page 35
READ_LOCK_TYPE= on page 35
REREAD_EXPOSURE= on page 36
SPOOL= on page 36
UPDATE_LOCK_TYPE= on page 37

The LIBNAME options are described here in detail.

Note: Control over locking might not be available for every DBMS. See your DBMS-specific documentation for details on the availability of each of these options.

ACCESS=READONLY

determines the access level with which a libref connection is opened. Using this option prevents writing to the DBMS. If you specify ACCESS=READONLY, tables and views can be read but not updated. If ACCESS= is omitted, tables and views can be read and updated if you have the necessary DBMS privileges.

CONNECTION=SHAREDREAD | GLOBALREAD | UNIQUE

indicates whether multiple opens in a DBMS can use the same connection.

Default value: SHAREDREAD, unless noted otherwise in your DBMS chapter.

The CONNECTION= option enables you to control the number of connections, and therefore transactions, that your DBMS engine executes and supports for each LIBNAME statement.

This option is supported by the DBMS engines that support multiple, simultaneous connections to the DBMS. For most DBMS engines, there must be a connection, also known as an attach, to the DBMS server before any data can be accessed. Typically, each DBMS connection has one transaction, or work unit, that is active in the connection. This transaction is affected by any SQL COMMITs or ROLLBACKs that the engine performs within the connection while executing the SAS application.
A DBMS table can be opened by the DBMS engine for reading (a read-only open), for creation (an output open), or for updating (an update open).

The values for CONNECTION= are as follows:

**SHAREDREAD**

When CONNECTION=SHAREDREAD, the SAS/ACCESS LIBNAME statement makes one connection to the DBMS. All tables that are opened for reading by this LIBNAME or libref share this connection.

A separate connection is established for each table that is opened for update or output.

SHAREDREAD is the default value for CONNECTION= because it offers the best performance and it guarantees data integrity.

In the following example, MYDBLIB and MYDBLIB2 have separate connections. The first connection is used to print the data from MYDBLIB.TAB while a third connection is made for updating MYDBLIB.TAB. The third connection is closed at the end of the step, whereas the first and second connections are closed with the CLEAR option.

```sas
libname mydblib oracle user=testuser
  pw=testpass path=abc
  connection=sharedread;

libname mydblib2 oracle user=testuser
  pw=testpass path=abc
  connection=sharedread;

proc print data=mydblib.tab
proc sql;
  update mydblib.tab ...

libname mydblib clear;
libname mydblib2 clear;
```

**GLOBALREAD**

When CONNECTION=GLOBALREAD, multiple SAS/ACCESS LIBNAME statements or librefs that use identical values for all SAS/ACCESS engine connection options can share the same connection to the DBMS. All tables that are opened for reading by any of these LIBNAME statements share this read-only access.

A separate connection is established for each table that is opened for update or output.

GLOBALREAD can be used if you want to minimize the cost of having a separate connection for each LIBNAME statement.

In the following example, the two librefs, MYDBLIB and MYDBLIB2, share the same connection for read access because CONNECTION=GLOBALREAD and the connection options are identical. The first connection is used to print the data from MYDBLIB.TAB while a second connection is made for updating MYDBLIB.TAB. The second connection is closed at the end of the step. Note that

```sas
libname mydblib clear;
```
does not close the first connection. The first connection is closed with the final
LIBNAME statement.

libname mydblib oracle user=testuser
    pw=testpass path=abc
    connection=globalread;

libname mydblib2 oracle user=testuser
    pw=testpass path=abc
    connection=globalread;

proc print data=mydblib.tab
proc sql;
    update mydblib.tab ...

libname mydblib clear;
libname mydblib2 clear;

UNIQUE

When CONNECTION=UNIQUE, a new connection to the DBMS is made for
every table that is opened in your SAS application. This is useful if you want
each use of a table to have its own unique connection and transaction.

In the following example, the libref, MYDBLIB, makes the first connection.
The first connection is used to print the data from MYDBLIB.TAB while a
second connection is made for updating MYDBLIB.TAB. The second
connection is closed at the end of the step. The first connection is closed with
the CLEAR option in the LIBNAME statement.

libname mydblib oracle user=testuser
    pw=testpass path=abc
    connection=unique;

proc print data=mydblib.tab
proc sql;
    update mydblib.tab ...

libname mydblib clear;

See the glossary for the definitions of connection, commit, rollback, query, and
transaction.

See also the options DEFER= on page 34, ACCESS= on page 26, and
CONNECTION_GROUP= on page 28.

Note: The number of simultaneous connections supported by each DBMS
varies. See your DBMS documentation for details.

CONNECTION_GROUP=connection_group_name

specifies a connection that can be shared among several LIBNAME statements (or
librefs) or by connections made with the SQL Procedure Pass-Through Facility
CONNECT statement.

Default value: none

By specifying the name of a connection group, you can share one DBMS
connection among several different LIBNAME statements. The connection to the
DBMS can be shared only if each LIBNAME statement specifies the same CONNECTION_GROUP= value and specifies identical DBMS connection options.

When CONNECTION_GROUP= is specified, it implies that the value of the CONNECTION= option will be GLOBALREAD.

In the following example, the MYDBLIB libref shares a connection with MYDBLIB2 by specifying CONNECTION_GROUP=MYGROUP and by specifying identical connection options. The libref, MYDBLIB3, makes a second connection to another connection group called ABC. The first connection is used to print the data from MYDBLIB.TAB while a third connection is made for updating MYDBLIB.TAB. The third connection is closed at the end of the step. Note that

```sas
libname mydblib clear;
```
does not close the first connection. The first connection is closed with the final LIBNAME statement for that connection

```sas
libname mydblib2 clear;
```
Similarly, the second connection is closed with

```sas
libname mydblib3 clear;
```

```sas
libname mydblib oracle user=testuser
    pw=testpass
    connection_group=mygroup;
libname mydblib2 oracle user=testuser
    pw=testpass
    connection_group=mygroup;
libname mydblib3 oracle user=testuser
    pw=testpass
    connection_group=abc;
```

```sas
proc print data=mydblib.tab
proc sql;
    update mydblib.tab ...;
```

```sas
libname mydblib clear;
libname mydblib2 clear;
libname mydblib3 clear;
```

**DBCONINIT=** specifies a user-defined initialization command to be executed immediately after every connection to the DBMS that is within the scope of the LIBNAME statement or libref.

Default value: none

The initialization command that you select can be a script, stored procedure, or any DBMS SQL language statement that might provide additional control over the interaction between your SAS/ACCESS engine and the DBMS.

You can specify any DBMS command that can be executed by the SAS/ACCESS engine and that does not return a result set or output parameters. The command executes immediately after each DBMS connection is successfully established. If the command fails, a disconnect occurs, and the libref is not assigned. You must
specify the command as a single, quoted string, unless it is an environment variable.

In the following example, the DBCONINIT= option causes the DBMS to apply the SET statement to every connection that uses the MYDBLIB libref.

```sas
libname mydblib db2 ssid=db2
dbconinit="SET CURRENT SQLID='myauthid'";

proc sql;
select * from mydblib.customers;
insert into mydblib.customers
values('33129804', 'VA', '22809', 'USA',
'540/545-1400', 'BENNETT SUPPLIES',
'2199 LAUREL ST', 'ELKTON', '22APR97');
update mydblib.invoice
set amtbilled = amtbilled*1.10
where country = 'USA';
delete mydblib.specprod
where productid = 8934;
quit;
```

In the next example, a UNIX environment variable, DBMSINIT, contains a procedure to be passed to DBCONINIT=. The DBMS engine checks for this environment variable and executes it.

```sas
libname mydblib oracle user=testuser pass=testpass
dbconinit=dbmsinit;

The DBMS engine recognizes the environment variable, retrieves the stored procedure, and executes it.

See also DBCONTERM= on page 30.

Note: The initialization command might execute more than once, since one LIBNAME statement might have multiple connections; for example, one for reading and one for updating.

```

DBCONTERM=<<DBMS-user-command>>
```

specifies a user-defined termination command to be executed before every disconnect from the DBMS that is within the scope of the LIBNAME statement or libref.

Default value: none

The termination command that you select can be a script, stored procedure, or any DBMS SQL language statement that might provide additional control over the interaction between the SAS/ACCESS engine and the DBMS. You can specify any valid command that can be executed by the SAS/ACCESS engine and that does not return a result set or output parameters. The command executes immediately before SAS terminates each connection to the DBMS. If the command fails, SAS provides a warning message but the library deassignment and disconnect still occurs. You must specify the command as a single, quoted string.

In this example, the DBMS drops the Q1_SALES table before SAS disconnects from the DBMS.
libname mydblib db2 user=testuser using=testpass
db=invoice
dbconterm='drop table q1_sales';

In this example, the stored procedure, SALESTAB_STORED_PROC, is executed each time SAS connects to the DBMS, and the BONUSES table is dropped when SAS terminates each connection.

libname mydblib db2 user=testuser
using=testpass db=sales
dbconinit='exec salestab_stored_proc'
dbconterm='drop table bonuses';

See also DBCONINIT= on page 29.

Note: The termination command might execute more than once, since one LIBNAME statement might have multiple connections; for example, one for reading and one for updating.

DBINDEX=YES | NO
indicates whether SAS applications attempt to use any indexes on DBMS tables in the specified libref.

Default value: specific to your DBMS

If you specify DBINDEX=YES in SAS applications, such as PROC SQL and the DATA step, SAS attempts to use indexes on a DBMS table to improve performance.

If you specify DBINDEX=NO, SAS makes no attempt to use indexes on a DBMS table.

See your DBMS chapter for DBMS-specific details.
See Advanced Topics for more information on setting this option for performance enhancement.

In this example, setting DBINDEX=YES in the LIBNAME statement improves the efficiency of the PROC SQL join because the EMPLOYEES.BIRTHDATE column has an index defined on it.

libname mydblib oracle user=testuser
    password=testpass dbindex=yes;

proc sql;
    select employees.lastname,
           employees.idnum,
           payroll.salary
    from mydblib.employees, mydblib.payroll
    where employees.birthdate=payroll.birth;
quit;

DBLIBINIT= <'DBMS-user-command'>
specifies a user-defined initialization command to be executed once within the scope of the LIBNAME statement or libref that established the first connection to the DBMS.

Default value: none

The initialization command that you select can be a script, stored procedure, or any DBMS SQL language statement that might provide additional control over the interaction between your SAS/ACCESS engine and the DBMS.
You can specify any DBMS command that can be executed by the SAS/ACCESS engine and that does not return a result set or output parameters. The command executes immediately after the first DBMS connection is successfully established. If the command fails, a disconnect occurs, and the libref is not assigned. You must specify the command as a single, quoted string, unless it is an environment variable.

This option will fail if either the CONNECTION=UNIQUE or DEFER=YES or both of these LIBNAME options are specified. When CONNECTION=GLOBALREAD is specified, the initialization command will be executed for each LIBNAME statement that has the GLOBALREAD specification. However, any of the LIBNAME statements that have CONNECTION=GLOBALREAD specified, but do not have the same initialization command as the first LIBNAME statement, will fail.

In this example, CONNECTION=GLOBALREAD is specified on both LIBNAME statements. Therefore, the TEST command will be executed after each of the LIBNAME statements.

```
libname mydblib oracle user=testuser pass=testpass
c   connection=globalread dblibinit='Test';
```

```
libname mydblib2 oracle user=testuser pass=testpass
c   connection=globalread;
```

In this example, CONNECTION=GLOBALREAD is specified on both LIBNAME statements but the DBLIBINIT commands are different. Therefore, the second LIBNAME statement will fail.

```
libname mydblib oracle user=testuser pass=testpass
c   connection=globalread dblibinit='Test';
```

```
libname mydblib2 oracle user=testuser pass=testpass
c   connection=globalread dblibinit='NoTest';
```

See also DBLIBTERM= on page 32.

**DBLIBTERM= <'DBMS-user-command'>**

specifies a user-defined termination command to be executed once before the DBMS disconnect that is associated with the first connection made by the LIBNAME statement or libref.

Default value: none

The termination command that you select can be a script, stored procedure, or any DBMS SQL language statement that might provide additional control over the interaction between the SAS/ACCESS engine and the DBMS. You can specify any valid command that can be executed by the SAS/ACCESS engine and that does not return a result set or output parameters. The command executes immediately before SAS terminates the last connection to the DBMS. If the command fails, SAS provides a warning message but the library deassignment and disconnect still occurs. You must specify the command as a single, quoted string.

This option will fail if either the CONNECTION=UNIQUE or DEFER=YES or both of these LIBNAME options are specified. When CONNECTION=GLOBALREAD is specified, the termination command will be executed for each LIBNAME statement that has the GLOBALREAD specification. However, any of the LIBNAME statements that have CONNECTION=GLOBALREAD specified, but do not have the same termination command as the first LIBNAME statement, will fail.
In this example, CONNECTION=GLOBALREAD is specified on both LIBNAME statements. Therefore, the TEST command will be executed after each of the LIBNAME statements.

```sas
libname mydblib oracle user=testuser pass=testpass
    connection=globalread dblibterm='Test';
```

```sas
libname mydblib2 oracle user=testuser pass=testpass
    connection=globalread;
```

In this example, CONNECTION=GLOBALREAD is specified on both LIBNAME statements but the DBLIBTERM commands are different. Therefore, the second LIBNAME statement will fail.

```sas
libname mydblib oracle user=testuser pass=testpass
    connection=globalread dblibterm='Test';
```

```sas
libname mydblib2 oracle user=testuser pass=testpass
    connection=globalread dblibterm='NoTest';
```

See also DBLIBINIT= on page 31.

**DBPROMPT=YES | NO**

Specifies whether SAS displays a window that prompts the user to enter DBMS connection information prior to connecting to the DBMS.

Default value: NO

If you specify DBPROMPT=YES, SAS displays a window that interactively prompts you for the DBMS connection options the first time the libref is used. Therefore, it is not necessary to provide connection options with the LIBNAME statement. If you do specify connection options with the LIBNAME statement and you specify DBPROMPT=YES, the connection option values are displayed in the window. These values can be overridden interactively.

If you specify DBPROMPT=NO, SAS does not display the prompting window.

The DBPROMPT= option interacts with the DEFER= option to determine when the prompt window appears. If DEFER=NO, the DBPROMPT window opens when the LIBNAME statement is executed. If DEFER=YES, the DBPROMPT window opens the first time a table or view is opened. The DEFER= option normally defaults to NO but defaults to YES if DBPROMPT=YES. You can override this default by explicitly setting DEFER=NO.

The DBPROMPT window usually opens only once for each time that the LIBNAME statement is specified. It might open multiple times if DEFER=YES and the connection fails when SAS tries to open a table. In these cases, the DBPROMPT window opens until a successful connection occurs or the user selects Cancel.

In this example, the DBPROMPT window does not open when the LIBNAME statement is submitted because DEFER=YES. The DBPROMPT window opens when the PRINT procedure is processed, a connection is made, and the table is opened.

```sas
libname mydblib oracle dbprompt=yes
    defer=yes;

proc print data=mydblib.staff;
run;
```
In the next example, the DBPROMPT window opens while the LIBNAME statement is processing. The DBPROMPT window does not open in subsequent statements because the DBPROMPT window opens only once per LIBNAME.

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

In the next example, values provided in the LIBNAME statement are pulled into the DBPROMPT window. The values `testuser` and `ABC_server` appear in the DBPROMPT window and can be edited and confirmed by the user.

```
libname mydblib oracle
    user=testuser pw=testpass
    path=ABC_server dbprompt=yes defer=no;
```

See also DEFER= on page 34.

**DEFER=**

Determines when the connection to the DBMS occurs.

- Default value: NO
  - If DEFER=YES, the connection to the DBMS occurs when a table in the DBMS is opened. If DEFER=NO, the connection to the DBMS occurs when the libref is assigned by a LIBNAME statement. The DEFER= option is ignored when CONNECTION=UNIQUE because a connection is performed for every open.

**DIRECT_SQL=**

Allows you to specify whether the SQL Procedure uses the Direct SQL Join feature.

- Default value: YES
  - If DIRECT_SQL=YES, joins are sent to the DBMS for processing, when possible. If DIRECT_SQL=NO, direct joins are processed in SAS.

**PRESERVE_COL_NAMES=**

Preserves spaces, special characters, and case sensitivity in DBMS column names.

- Default value: specific to your DBMS
  - If PRESERVE_COL_NAMES=NO, column names that are read from the DBMS are converted to SAS variable names by using the SAS name normalization rules. These rules allow the name to be mixed case but to contain only alphanumerics or the underscore character (_). If a character in a DBMS column name is not an alphanumeric or underscore, it is converted to an underscore in the corresponding SAS variable name. For example, the ORACLE column, "Total$Cost", can be referenced in a SAS program as "Total_Cost".
  - If PRESERVE_COL_NAMES=NO, column names that are passed to the DBMS from a SAS application must conform to the SAS name normalization rules or an error message will be printed.
  - If PRESERVE_COL_NAMES=YES, column names are read from and passed to the DBMS with special characters and the exact, case-sensitive spelling of the name preserved. To use column names in your SAS program that are not valid SAS names, you must use one of the following techniques that are supported by the SAS language:
Use the DQUOTE option in PROC SQL and then reference your columns using double quotes. For example:

```sas
proc sql dquote=ansi;
  select "Total$Cost" from mydblib.mytable;
```

Specify the global system option VALIDVARNAME=ANY and use name literals in the SAS language. For example:

```sas
proc print data=mydblib.mytable;
  format 'Total$Cost' 22.2;
```

PRESERVE_COL_NAMES= does not apply to the PROC SQL Pass-Through facility.

PRESERVE_COL_NAMES= does interact with the PROC SQL option, DQUOTE=ANSI, and the VALIDVARNAME=ANY system option as described above.

See your DBMS chapter for DBMS specific details.

See also PRESERVE_TAB_NAMES= on page 35. For more information on SAS names, see the SAS Language Reference: Dictionary.

PRESERVE_TAB_NAMES=NO | YES

preserves spaces, special characters, and case-sensitivity in DBMS table names.

Default value: specific to your DBMS

If PRESERVE_TAB_NAMES=NO, table names that are read from the DBMS are converted to SAS data set names by using the SAS name normalization rules. These rules allow the name to be mixed case but to contain only alphanumerics or the underscore (_). If a character in a DBMS table name is not an alphanumeric or an underscore, it is converted to an underscore in the corresponding SAS data set name.

If PRESERVE_TAB_NAMES=YES, table names are read from and passed to the DBMS with special characters and the exact, case-sensitive spelling of the name preserved.

See your DBMS chapter for details.

See also PRESERVE_COL_NAMES= on page 34.

READ_LOCK_TYPE=ROW | PAGE | TABLE | NOLOCK

specifies how data in a DBMS table is locked when data is read.

Default value: specific to your DBMS

See your DBMS chapter for details on the option values supported for your DBMS and details on locking.

READ_LOCK_TYPE= can take one of the following values:

ROW

locks a row if any of its columns are accessed. If you are using the SAS/ACCESS Interface to ODBC or DB2CS, READ_LOCK_TYPE=ROW indicates that locking is based on the READ_ISOLATION_LEVEL= option.

PAGE

locks a page of data, which is a DBMS specific number of bytes.

TABLE

locks the entire DBMS table.
NOLOCK

does not lock the DBMS table, pages, or any rows during a read transaction.

If you omit READ_LOCK_TYPE=, you get the default action for the DBMS that you are using. You can set a lock for one DBMS table by using the data set option or for all tables in a particular DBMS by using the LIBNAME option.

If you specify READ_LOCK_TYPE=TABLE, you must also specify CONNECTION=UNIQUE, or you will receive an error message. Setting CONNECTION=UNIQUE ensures that your table lock is not lost, for example, due to another table closing and committing rows in the same connection.

See also the data set option "READ_LOCK_TYPE=" on page 52 and the LIBNAME option UPDATE_LOCK_TYPE on page 37.

REREAD_EXPOSURE=NO | YES

specifies whether the DBMS engine will behave like a random access engine for the scope of the LIBNAME statement.

Default value: NO

If you specify REREAD_EXPOSURE=YES, the DBMS engine will behave like a random access engine when rereading a row but you cannot guarantee that the same row will be returned. For example, if a row is read and then deleted, it is possible that the next time you try to read that same row you actually will be reading a different row. You will have the potential for data integrity exposures within the scope of your SAS session.

If you specify REREAD_EXPOSURE=NO, the DBMS engine will behave as an RMOD engine which means that your data is protected by the normal data protection that SAS provides.

SPOOL=YES | NO | DBMS

specifies whether SAS creates a utility spool file during read transactions that read data more than once.

Default value: YES

In some cases, SAS processes data in more than one pass through the same set of rows. Spooling is the process of writing rows, that have been retrieved during the first pass of a data read, to a spool file. In the second pass, rows can be reread without performing I/O to the DBMS a second time. When data must be read more than once, spooling improves performance. Spooling also guarantees that the data remains the same between passes, as most SAS/ACCESS engines do not support member-level locking.

If you specify SPOOL=YES, SAS creates a utility spool file into which it writes the rows that are read the first time. For subsequent passes through the data, the rows are read from the utility spool file rather than rereading them from the DBMS table. This guarantees that the row set is the same for every pass through the data.

If you specify SPOOL=NO, the required rows for all passes of the data are read from the DBMS table. No spool file is written. There is no guarantee that the row set will be the same for each pass through the data.

If you specify SPOOL=DBMS, the required rows for all passes of the data are read from the DBMS table but additional enforcements are made on the DBMS server side to ensure the row set is the same for every pass through the data.
UPDATE_LOCK_TYPE=ROW | PAGE | TABLE | NOLOCK

specifies how data in a DBMS table is locked during an update transaction.

Default value: specific to your DBMS

See your DBMS chapter for details on the option values supported for your DBMS and details on locking.

UPDATE_LOCK_TYPE= can take one of the following values:

ROW
locks a row if any of its columns are going to be updated.

PAGE
locks a page of data, which is a DBMS specific number of bytes.

TABLE
locks the entire DBMS table.

NOLOCK

does not lock the DBMS table, page, or any rows when reading them for update.

You can set a lock for one DBMS table by using the data set option or for all tables in a particular DBMS by using the LIBNAME option.

See also the data set option “UPDATE_LOCK_TYPE=” on page 53 and the LIBNAME option READ_LOCK_TYPE on page 35.

Specifying a LIBNAME Statement to Access DBMS Data

In this example, the libref MYDBLIB uses the ORACLE engine to connect to an ORACLE database. The SAS/ACCESS engine connection options are USER=, PASSWORD=, and PATH=. The LIBNAME options specify that row-level locking be used when data is read or updated.

libname mydblib oracle user=testuser
    password=testpass path=myorapath
    read_lock_type=row update_lock_type=row;

proc print data=mydblib.employees;
    where jobcode=602;
run;

In the next example, the DBPROMPT= option enables you to enter connection information in a prompting window rather than in the LIBNAME statement. In this example, the DEFER=NO option specifies that the LIBASSIGN window opens when the libref is assigned rather than when the table is opened.

libname mydblib oracle dbprompt=yes
dbindex=yes defer=no;

proc print data=mydblib.payroll;
run;
For this example, the libref MYDBLIB uses the ORACLE engine to create a table. The DATA step ABORT will cause the DBMS engine to issue a ROLLBACK command. The resulting behavior of the engine is DBMS specific.

```plaintext
libname mydblib oracle user=testuser
    password=testpass path=myorapath;
    data mydblib.x;
    j=1;
    abort;
    run;
```