CHAPTER

4

SAS/ACCESS Data Set Options

Introduction

This chapter describes the SAS/ACCESS options that you can specify on a SAS data set in the form `SAS/ACCESS-libref.dbms_table_name`. A data set option applies only to the data set on which it is specified. A data set option remains in effect for the duration of the DATA step or procedure.

For information about DBMS-specific data set options and details such as default values, refer to your DBMS chapter. In general, whenever there are like-named LIBNAME and data set options, the value of the data set option defaults to the value of the LIBNAME option, if not specified. See “SAS/ACCESS LIBNAME Statement” on page 24 for a description of the new SAS/ACCESS LIBNAME statement and its options, which can be assigned to a group of DBMS tables or views.

SAS/ACCESS Data Set Options

The SAS/ACCESS data set options are as follows:

- “DBCOMMIT=’” on page 40
- “DBCONDITION=’” on page 41
- “DBCREATE_TABLE_OPTS” on page 42
- “DBFORCE=’” on page 43
- “DBGEN_NAME=’” on page 43
- “DBINDEX=’” on page 44
- “DBKEY=’” on page 45
- “DBLABEL=’” on page 46
- “DBNULL=’” on page 47
- “DBPROMPT=’” on page 48
- “DBTYPE=’” on page 49
- “ERRLIMIT=’” on page 50
- “NULLCHAR=’” on page 50
- “NULLCHARVAL=’” on page 51
In addition to the SAS/ACCESS data set options that are described in this chapter, you can also use several other SAS data set options when you access DBMS data. The following list includes the SAS data set options that can be used with DBMS data. For a complete listing of data set options that can be used with data sets that are composed of SAS data, refer to the SAS Language Reference Dictionary.

CNTLLEV=
DROP=
FIRSTOBS=
IN=
KEEP=
LABEL=
OBS=
RENAME=
TYPE=
WHERE=

Note: The REPLACE= data set option is not supported by DBMS engines. If this option is specified, the engine supervisor will print a warning message.

Note: When used with DBMS data, the LABEL= and TYPE= options apply only to input for the duration of the SAS procedure in which they are used. The CONTENTS procedure uses these options as input options.

In this example, the DROP= option causes the SAS/ACCESS engine to omit the SALARY column when it reads the MYDBLIB.EMPLOYEES table.

```sas
libname mydblib db2 ssid=db2 authid=sasdemo;
proc sql;
select *
from mydblib.employees(drop=salary)
where dept='ACC024';
quilt;
```

## Dictionary

### DBCOMMIT=

Enables you to issue a DBMS commit statement automatically after a specified number of rows have been processed.

Default value: 0

See Also: ERRLIMIT=
**Syntax**

```
DBCOMMIT=n
```

*n* is an integer.

**Details**

The default value of DBCOMMIT= is 0, which means that a commit is issued only once at the end of processing.

In the following example, a commit is issued after every 10 rows are inserted:

```
data oracle.dept(dbcommit=10);
  set myoralib.staff;
run;
```

DBCOMMIT= affects update, delete, and insert processing. The number of rows processed includes rows that are not processed successfully.

The DBCOMMIT= option overrides the ERRLIMIT= option. This means that a commit may be issued prior to a rollback that is needed by the ERRLIMIT= option.

---

**DBCONDITION=**

Specifies criteria for subsetting and ordering DBMS data.

Default value: none

**Syntax**

```
DBCONDITION="DBMS-SQL-query-clause"
```

*DBMS-SQL-query-clause* is a DBMS-specific SQL query clause, such as WHERE, GROUP BY, HAVING, or ORDER BY.

**Details**

The DBCONDITION= data set option enables you to pass a DBMS-specific SQL selection condition to the DBMS for processing.

When you create a view descriptor, you can use the PROC ACCESS SUBSET statement to specify selection criteria. These criteria are in the form of DBMS specific SQL query clauses, which the SAS/ACCESS engine passes directly to the DBMS for processing. In Version 7, the DBCONDITION= data set option enables you to perform the same task. When selection criteria are passed directly to the DBMS for processing, performance is often enhanced. The DBMS checks the criteria for syntax errors when it receives the SQL query.
In the following example, the function passed to the DBMS with the
DBCONDITION= option causes the DBMS to return to SAS only the rows that satisfy
the condition.

```
proc sql;
  create view maxsal as
  select maxsalary from mydblib.staff
    (dbcondition="where maxsalary = (select max(salary)
      from mydblib.staff)")
  using libname mydblib db2 ssid=db2 authid=allairpt;
```

---

**DBCREATE_TABLE_OPTS**

Specifies DBMS-specific syntax to be added to the CREATE TABLE statement.

**Default value:** none

**Syntax**

```DBCREATE_TABLE_OPTS=DBMS-SQL-query-clause```

**DBMS-SQL-query-clause**

is a DBMS-specific SQL query clause, such as WHERE, GROUP BY, HAVING, or
ORDER BY.

**Details**

The DBCREATE_TABLE_OPTS= data set option allows you to add DBMS-specific
syntax to the end of the SQL CREATE TABLE statement. This data set option only
applies when you are creating a DBMS table by specifying a libref associated with
DBMS data.

In the following example, a DB2 table TEMP is created with the value of the
DBCREATE_TABLE_OPTS= option appended to the CREATE TABLE statement. For
this example, the table would be created using

```
CREATE TABLE TEMP (X DOUBLE) PARTITIONING
  KEY (X) USING HASHING
libname mydblib db2 user=testuser
  pwd=testpass dsn=sample;

data mydblib.temp (DBCREATE_TABLE_OPTS=’PARTITIONING
  KEY (X) USING HASHING’);
  x=1; output;
  x=2; output;
run;
```
**DBFORCE=**

Specifies whether to force the truncation of data during insert processing.

Default value: NO

See Also: DBTYPE=

**Syntax**

**DBFORCE** = YES | NO

YES

inserts rows into the specified DBMS table, truncating data values that exceed the length of the DBMS column.

NO

does not insert rows into the specified DBMS table when data values exceed the length of the DBMS column.

**Details**

The DBFORCE= option is overridden by the FORCE option when used with PROC APPEND and the UPDATE statement when used with PROC SQL. The PROC SQL UPDATE statement does not provide a warning before truncating the data.

In the following example, the table MYDBLIB.DEPT is created from the SAS data set MYORALIB.STAFF. The LASTNAME variable is a character variable of length 20 in MYORALIB.STAFF. During the creation of MYDBLIB.DEPT, the LASTNAME variable is stored as a column of type character and length 10 by using DBFORCE=YES.

```
data mydblib.dept(dbtype=(lastname='char(10)')
                  dbforce=yes);
   set myoralib.staff;
run;
```

See your DBMS chapter for details on the option values that are supported by your DBMS.

**DBGEN_NAME=**

Specifies whether to automatically rename columns that contain characters that SAS does not allow, such as '$', to valid SAS variable names.

Default value: DBMS

**Syntax**

**DBGEN_NAME** = DBMS | SAS
DBMS
renames DBMS columns that contain characters that SAS does not allow to valid SAS variable names.

SAS
renames DBMS columns that contain characters that SAS does not allow to the format _COLn, where n is the column number (starting with zero).

Details
SAS retains column names when reading data from tables, unless a column name contains characters that SAS does not allow, such as '. ' SAS allows alphanumeric characters and the underscore ('_').

If you specify DBGEN_NAME=SAS, a DBMS column named 'dept$amt' is renamed to '_COLn,' where n is the column number. If you specify DBGEN_NAME=DBMS, unallowed characters are converted to underscores, so the same column would be renamed 'dept_amt.' If a name is converted to a name that already exists, a sequence number is appended to the end.

This option is intended primarily for National Language Support, notably the conversion of Kanji to English characters because the English characters converted from Kanji are often those that are not allowed in SAS.

Note These rules apply when the SAS system option VALIDVARNAME=V7. When you set VALIDVARNAME=V6, DBGEN_NAME=SAS behaves the same way, but DBGEN_NAME=DBMS handles duplicate column names differently. Instead of appending a sequence number to the end of the column name, SAS replaces the last character with the sequence number, for example, a column named "_DEPT" becomes "_DEP0." △

DBINDEX=

Indicates whether SAS queries the DBMS to find indexes on the specified table.

Default value: DBMS specific

See Also: DBKEY=

Syntax

DBINDEX=YES | NO | <>index-name<>

YES
queries the DBMS to find all of the table's indexes and attempts to use indexes on a DBMS table to improve performance.

NO
does not attempt to use indexes on a DBMS table.

index-name
uses the specified index name to improve performance.
Details

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.

If you specify DBINDEX=index-name, the specified index name is used by SAS applications to improve performance. The index name must exist and must be correct for the given context.

When you use the DBINDEX= option with the DATA step KEY= option, you must provide the index name as the value for the KEY= option.

In this example, setting DBINDEX=YES allows a DBMS index on the MYDBLIB.EMPLOYEES table to be used, if it exists.

data mydata;
  set mydblib.employees(dbindex=yes);
  where lastname like 'A%';
run;

See your DBMS chapter for DBMS specific details.

DBKEY=

Specifies the column to use as an index.

Default value: none

See Also: DBINDEX=

Syntax

DBKEY=(<column-1><...<column-n>>)

column

is the name of the column that forms the index on the DBMS table.

Details

When DBKEY= is specified, you provide the names of the columns that form the index. The index may or may not actually exist on the DBMS table. The DBKEY= option is similar to the DBINDEX=index-name option except that instead of providing the index name, you provide the column name or names that form the index. SAS uses the specified column name or names as an index by constructing and passing a WHERE clause to the DBMS. DBINDEX= and DBKEY= are mutually exclusive; if you specify them together, SAS overrides the DBINDEX= data set option and the DBINDEX= LIBNAME option.

You can use the DBKEY= option if the index that you need does not exist in the DBMS or if the engine cannot retrieve index information from the DBMS due to insufficient privileges.
The DBKEY= option can be used to improve performance just as indexing can improve performance. With the DBKEY= option, it isn’t necessary to have an existing index on the DBMS table.

NoteWhen you use DBKEY= with the DATA step KEY= option, you must specify DBKEY as the value of the KEY= option, as shown in the following examples.

```
libname mydblib oracle user=testuser
   pw=testpass;
data keyvalues;
   input empid;
datalines;
1247
4444;
;
data mydata;
   set keyvalues;
   set mydblib.employees(dbkey=empid) key=dbkey;
run;
```

The next example shows two columns specified as keys. These columns are used to simulate a composite index.

```
libname mydblib oracle user=testuser
   pw=testpass;
data keyvalues;
   input empid jobcode;
datalines;
1247 10
1266 20;
;
data mydata;
   set keyvalues;
   set mydblib.employees(dbkey=(empid jobcode))
      key=dbkey;
run;
```

### DBLABEL=

Specifies whether to use SAS variable labels as DBMS column names during output processing.

**Default value:** NO

#### Syntax

```
DBLABEL= YES | NO
```

**YES**

uses SAS variable labels as DBMS column names.
NO
uses existing DBMS column names.

Details
This option is valid only for creating DBMS tables.
In the following example, a SAS data set NEW is created with one variable C1. This
variable is assigned a label of DEPTNUM. In the second DATA step, the
MYDBLIB.MYDEPT table is created by using DEPTNUM as the DBMS column name.
Setting DBLABEL=YES enables the label to be used as the column name.

```sas
data new;
  label c1='deptnum';
  c1=001;
run;

data mydblib.mydept(dblabel=yes);
  set new;
run;

proc print data=mydblib.mydept;
run;
```

**DBNULL=**

Indicates whether NULL is a valid value for the specified columns when a table is created.

Default value:  DBMS specific
See Also:  NULLCHAR=, NULLCHARVAL=

**Syntax**

```
DBNULL=(<column-name-1=YES | NO> <... <column-name-n=YES | NO>>)  
```

*column-name*
  is a DBMS column in the table that is being created.

**YES**
  indicates that a NULL value is valid for the specified columns in the DBMS table.

**NO**
  indicates that a NULL value is not valid for the specified columns in the DBMS table.

**_ALL_**
  indicates that the YES or NO value applies to all columns in the table.

**Details**
This option is valid only for creating DBMS tables.
If you specify more than one column name, the names must be separated with spaces. In the following example, the EMPID and JOBCODE columns in the new MYDBLIB.MYDEPT2 table are prevented from accepting null values by using the DBNULL= option. Any subsequent attempt to insert NULL values into these columns will fail.

```sas
data mydblib.mydept2(dbnull=(empid=no jobcode=no));
  set mydblib.employees;
run;
```

In the next example, all columns in the new MYDBLIB.MYDEPT3 table, except for the JOBCODE column, are prevented from accepting null values.

```sas
data mydblib.mydept3(dbnull=(_ALL_=no jobcode=YES));
  set mydblib.employees;
run;
```

Note that the DBNULL= option processes values from left to right, so if you specify a column name twice, or if you use the _ALL_ value, the last value overrides the first value specified for the column.

See your DBMS chapter for details on the option values that are supported by your DBMS.

---

**DBPROMPT=**

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

**Alias:** PWREQ

**Default value:** NO

**Syntax**

DBPROMPT=**YES | NO**

**YES**

causes SAS to display the prompting window.

**NO**

does not cause SAS to display the prompting window.

**Details**

As a data set option, DBPROMPT= is supported only for Version 6 view descriptors.

In the following example, connection options are specified in the ACCESS procedure. The DBPROMPT= data set option defaults to NO during the PRINT procedure because it is not specified.

```sas
proc access dbms=oracle;
  create alib.mydesc.access;
  user=testuser;
```
In the next example, the DBPROMPT window opens during connection to the DBMS. Values that were previously specified during the creation of MYVIEW are pulled into the DBPROMPT window fields. The user must edit or accept the connection information in the DBPROMPT window to proceed.

```
proc print data=myview(dbprompt=yes);
run;
```

### DBTYPE=

**Specifies a data type to use instead of the default DBMS data type when SAS creates a DBMS table.**

**Default value:** DBMS specific

**See Also:** DBFORCE=

#### Syntax

```
DBTYPE=(<column-name-1='<DBMS type>'>
   ...
   <column-name-n='<DBMS-type'>)
```

- **column-name**
  - specifies a DBMS column name.

- **DBMS-type**
  - specifies a DBMS data type.

#### Details

This option is valid only for creating DBMS tables.

By default, the SAS/ACCESS engine for your DBMS converts each SAS data type to a predetermined DBMS data type when outputting data to your DBMS. When you need a different data type, you can use `DBTYPE=` to override the default data type chosen by the SAS/ACCESS engine.

In the following example, `DBTYPE=` specifies the data types that are used when creating columns in the DBMS table.

```
data mydblib.newdept(dbtype=(deptno='number(10,2)' city='char(25)'));
set mydblib.dept;
run;
```
ERRLIMIT=

Specifies the number of errors that are allowed before SAS stops processing and issues a rollback.

Default value: 1

See Also: DBCOMMIT=

Syntax

ERRLIMIT=integer

integer

is a positive integer that represents the number of errors after which SAS stops processing and issues a rollback.

Details

SAS calls the DBMS to issue a rollback after the specified number of errors occurs during the processing of inserts, deletes, updates, and appends. If ERRLIMIT= is set to 0, SAS processes all rows, regardless of the number of errors that occur. SAS displays the total number of rows processed and the number of failed rows, if applicable, in the log.

If you use the DBCOMMIT= option, DBCOMMIT= overrides the ERRLIMIT= option.

Note: If you specify a value for DBCOMMIT= other than zero, rollbacks affected by the ERRLIMIT= option might not include records that are processed unsuccessfully because they were already committed by DBCOMMIT=.

In the following example, SAS stops processing and issues a rollback to the DBMS at the occurrence of the tenth error.

```
data mydata;
  set mydb.dept(errlimit=10);
  where salary > 40000;
run;
```

NULLCHAR=

Indicates whether a SAS character missing value is inserted into the DBMS column as a NULL value.

Default value: SAS

See Also: NULLCHARVAL=, DBNULL=
Syntax

NULLCHAR=SAS | YES | NO

SAS
indicates that a NULL value is inserted if the DBMS allows NULL. Otherwise, the value that is specified by the NULLCHARVAL=data set option is inserted. If NULLCHARVAL= is not specified, a blank is inserted.

YES
indicates that a NULL value is inserted if the DBMS allows NULL. Otherwise, an error is returned.

NO
indicates that the value specified by the NULLCHARVAL=data set option is inserted. If NULLCHARVAL= is not specified, a blank is inserted.

Details
The NULLCHAR= option determines whether or not a NULL is inserted into the DBMS column when there is a character missing value in the SAS data set.

The NULLCHAR= option works in conjunction with the NULLCHARVAL=data set option, which determines what is inserted if NULL values are not allowed. If NULLCHAR=NO, the value specified by NULLCHARVAL= is inserted, regardless of whether the DBMS allows NULLs for the column. NULLCHAR= affects insert and update processing. Note that all SAS numeric missing values (represented in SAS as '.') are inserted into the DBMS as NULLs.

NULLCHARVAL=

Defines the character string to insert into a DBMS table when a character missing value is encountered in a SAS data set.

Default value: a blank character

See Also: NULLCHAR, DBFORCE, DBNULL=

Syntax

NULLCHARVAL='character-string'

character-string
is a string of characters.

Details
The NULLCHARVAL= option works with the NULLCHAR= option and affects insert and update processing. The NULLCHAR= option determines whether or not a SAS
READ_LOCK_TYPE=

Specifies how data in a DBMS table is locked during a read transaction.

Default value: DBMS specific

See Also: UPDATE_LOCK_TYPE=
READ_LOCK_TYPE=LIBNAME option on page 35

Syntax
READ_LOCK_TYPE=ROW | PAGE | TABLE | NOLOCK

ROW
locks a row if any of its columns are accessed.

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 or any rows during a read transaction.

Details
If you omit READ_LOCK_TYPE=, you get either the default action for the DBMS that you are using, or a lock for the DBMS that was set with the LIBNAME statement. 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 statement 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 your DBMS chapter for details on the option values supported for your DBMS.

SASDATEFMT=

Changes the SAS date format of a DBMS column.
SAS/ACCESS Data Set Options

UPDATE_LOCK_TYPE=

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

Default value: DBMS specific

See Also: READ_LOCK_TYPE=

UPDATE_LOCK_TYPE=LIBNAME option on page 37
Syntax

UPDATE_LOCK_TYPE=ROW | PAGE | TABLE | NOLOCK

ROW
locks a row if any of its columns are accessed.

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 row or table during an update transaction.

Details

If you omit UPDATE_LOCK_TYPE=, you get either the default action for the DBMS that you are using, or a lock for the DBMS that was set with the LIBNAME statement. 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 your DBMS chapter for details on the option values that are supported for your DBMS.