CHAPTER
10

DBLOAD Procedure Reference

Introduction

In most of the interfaces that are provided in Version 6 of SAS/ACCESS software, the DBLOAD procedure enabled you to create and load (that is, transfer data to) a database management system table from a SAS data set. It also enabled you to submit non-query, DBMS-specific SQL statements to the DBMS without leaving your SAS session. The DBLOAD procedure continues to be supported in Version 7 and later. However, there are some changes in the support, as described in "Version 8 Compatibility for Version 6 Procedures" on page 99.

This chapter provides general reference information for the DBLOAD procedure, including its options and statements. See your DBMS chapter to determine whether the DBLOAD procedure is supported for your DBMS, and for DBMS-specific information that pertains to the DBLOAD procedure.

Refer to *SAS Language Reference: Dictionary* and to the *SAS Companion* for your operating environment for information on SAS data sets, SAS data libraries, and their naming conventions. If you are using this documentation from a book, remember that help is available from the `Help` menu.

Note: It is recommended that you use the SQL Procedure Pass-Through Facility with its new options to take full advantage of Version 7 and later enhancements. For more information, see "SQL Procedure Pass-Through Facility Statements" on page 74. △

Naming Limits in the DBLOAD Procedure

In Version 8, the DBLOAD procedure still limits SAS variable names to a maximum of eight characters in length. If you need to load 32–character column names into the DBMS, it is recommended that you use the SAS/ACCESS LIBNAME statement or the SQL Procedure Pass-Through Facility instead of PROC DBLOAD.

You can use long member names in the PROC DBLOAD DATA= option, such as the name of a SAS data set that you want to load into a DBMS table. However, if you attempt to use long SAS variable names, you will get an error message advising you that long variable names are not supported in PROC DBLOAD. As in previous versions, you can use the PROC DBLOAD RENAME statement to rename the eight–character SAS variable names to long DBMS column names when you load the data into a DBMS table. You can also use the RENAME data set option to rename the columns after they are loaded into the DBMS.

Case Sensitivity in the DBLOAD Procedure

Most DBLOAD procedure statements convert lowercase characters in user-specified values and default values to uppercase. If your host or database is case sensitive and you want to specify a value that includes lowercase alphabetic characters (for example, a user ID or password), you should enclose the entire value in quotation marks. You must also put quotation marks around any value that contains special characters or national characters.

The only exception is the DBLOAD SQL statement. The DBLOAD SQL statement is passed to the DBMS exactly as you type it, with case preserved.

DBLOAD Procedure Syntax

The statements for your DBMS may differ from those listed here. See your DBMS chapter for details.

PROC DBLOAD <*statement-options*>;

Database Connection Statements

These statements are used to connect to your DBMS and vary depending on which SAS/ACCESS interface you are using. See your DBMS chapter for details. Examples include USER=, PASSWORD=, and DATABASE=.

Table Statement

TABLE= <'>*table-name* <'>;

Editing Statements

ACCDESC= <*libref.*>*access-descriptor*;

COMMIT= *commit-frequency*;

DELETE *variable-identifier-1*
<*…variable-identifier-n*>;

ERRLIMIT= *error-limit*;

LABEL;

LIMIT= *load-limit*;

LIST <ALL | COLUMN | *variable-identifier*>;

NULLS *variable-identifier-1* = Y | N
<*…variable-identifier-n* = Y | N>;

QUIT;

RENAME *variable-identifier-1* = <'>*column-name-1*<'>
<*...variable-identifier-n* = <'>*column-name-n*<'>>;

RESET ALL | *variable-identifier-1*<*...variable-identifier-n*>;

SQL *DBMS-specific SQL-statement*;

TYPE *variable-identifier-1* = '*column-type-1*' <*...variable-identifier-n* =
'*column-type-n*'>;

WHERE *SAS-where-expression*;

Creating and Loading Statement

LOAD;

RUN;

Details

The DBLOAD procedure enables you to create and load a DBMS table, append rows to an existing table, and submit non-query DBMS-specific SQL statements to the DBMS for processing without leaving your SAS session. The procedure constructs DBMS-specific SQL statements to create and load or append to a DBMS table by using any one of the following:

- a SAS data file
- a PROC SQL view or DATA step view
- a view descriptor that was created with the SAS/ACCESS interface to your DBMS or with another SAS/ACCESS interface product
- another DBMS table referenced by a SAS libref created with the Version 8 SAS/ACCESS LIBNAME statement.

The DBLOAD procedure associates each SAS variable with a DBMS column and assigns a default name and data type to each column. It also specifies whether each column accepts null values. You can use the default information or change it as necessary. When you are finished customizing the columns, the procedure creates the DBMS table and loads or appends the input data.

PROC DBLOAD Statement Options

The following options can be used in the PROC DBLOAD statement.

Options

DBMS=*database-management-system*
specifies which database management system you want to access. The DBMS= option is required. See your DBMS chapter for the value to enter for your DBMS.

DATA=<*libref.*>*SAS-data-set*
specifies the input data set. The input data can be retrieved from a SAS data file, a PROC SQL view, a DATA step view, a SAS/ACCESS view descriptor, or another DBMS table referenced by a SAS/ACCESS libref. If the SAS data set is permanent, you must use its two-level name, *libref.SAS-data-set*. If you omit the DATA= option, the default is the last SAS data set that was created.

APPEND

appends data to an existing DBMS table that you identify by using the TABLE= statement. When you specify APPEND, the input data specified with the DATA= option is inserted into the existing DBMS table. Your input data can be in the form of a SAS data set, PROC SQL view, or SAS/ACCESS view (view descriptor).

Note: When you use APPEND, you must ensure that your input data corresponds exactly to the columns in the DBMS table. If your input data does not include values for all columns in the DBMS table, you might corrupt your DBMS table by inserting data into the wrong columns. You can use the COMMIT, ERRLIMIT, and LIMIT statements to help safeguard against data corruption. The ERRLIMIT statement defaults to 10 when used with APPEND. △

The DELETE and RENAME statements can be used with APPEND to drop and rename SAS input variables that do not have corresponding DBMS columns. The RENAME statement indicates the column name in the DBMS table for the SAS data set variable that you specify. For example, this statement loads data that is associated with the SAS variable COUNTRY into the DBMS column named ORIGIN:

```
rename country=origin;
```

All PROC DBLOAD statements and options can be used with APPEND, except for the NULLS and TYPE statements, which have no effect when used with APPEND. The LOAD statement is required.

The following example appends new employee data from the SAS data set NEWEMP to the DBMS table EMPLOYEES. The COMMIT statement causes a DBMS commit to be issued after every 100 rows are inserted. The ERRLIMIT statement causes processing to stop after 10 errors occur.

```
proc dbload dbms=oracle data=newemp append;
   user=testuser;
   password=testpass;
   path='myorapath';
   table=employees;
   commit=100;
   errlimit=5;
   load;
run;
```

Note: By omitting the APPEND option from the DBLOAD statement, you can use the PROC DBLOAD SQL statements to create a DBMS table and append to it in the same PROC DBLOAD step. △

Procedure Statements

To invoke PROC DBLOAD, you use the options listed in “Options” on page 133 along with certain statements. The statements that you choose are determined by your task and your database. These statements vary per DBMS and might be optional; see your DBMS chapter for more information.

Table 10.1 on page 135 summarizes the PROC DBLOAD options and statements.

Table 10.1 Options and Statements Required for the DBLOAD Procedure

Tasks	Options and Statements You Use
create and load a DBMS table	**PROC DBLOAD** *statement-options*; *database-connection-options*; **TABLE=** <'>*table-name*<'>; **LOAD**; **RUN**;
submit a dynamic, non-query DBMS-SQL statement to DBMS (without creating a table)	**PROC DBLOAD** *statement-options*; *database-connection-options*; **SQL** *DBMS-specific-SQL-statements*; **RUN**;

The PROC DBLOAD statements are described in alphabetic order in the following sections.

ACCDESC=

Creates an access descriptor based on the new DBMS table.

Optional statement

Syntax

ACCDESC=<*libref.*>*access-descriptor*;

Details

The ACCDESC= statement creates an access descriptor based on the DBMS table that you are creating and loading. After the new table is created and loaded, the access descriptor is automatically created. You must specify an access descriptor that does not already exist.

An editing statement, such as ACCDESC=, must be specified after the database connection statements when you create and load a DBMS table. See "LOAD" on page 139 for more information.

The ACCDESC= statement has two aliases: ACCESS= and AD=.

COMMIT=

Issues a commit or saves rows after a specified number of inserts.

Optional statement

Default: 1000

Syntax

COMMIT=*commit-frequency*;

Details

The COMMIT= statement issues a commit (that is, generates a DBMS-specific SQL COMMIT statement) after the specified number of rows has been inserted.

The *commit-frequency* argument must be a nonnegative integer. To commit or save observations only after all the rows have been inserted, specify COMMIT=0.

Using this statement might improve performance by releasing DBMS resources each time the specified number of rows has been inserted.

If you omit the COMMIT= statement, a commit is issued (or a group of rows is saved) after each 1,000 rows are inserted and after the last row is inserted.

An editing statement, such as COMMIT=, must be specified after the database connection statements when you create and load a DBMS table. See “LOAD” on page 139 for more information.

DELETE

Does not load specified variables into the new table.

Optional statement

Syntax

DELETE *variable-identifier-1* <*...variable-identifier-n*>;

Details

The DELETE statement drops the specified SAS variables before the DBMS table is created. The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable’s place in the data set. For example, if you want to drop the third variable, submit the following statement:

```
delete 3;
```

You can drop as many variables as you want in one DELETE statement. If you drop more than one variable, separate the identifiers with spaces, not commas.

Even if you drop a variable from the list of variables, the positional equivalents of the variables do not change. For example, if you drop the second variable, the third variable is still referenced by the number 3, not 2.

An editing statement, such as DELETE, must be specified after the database connection statements when you create and load a DBMS table. See “LOAD” on page 139 for more information.

ERRLIMIT=

Stops the loading of data after a specified number of errors.

Optional statement

Default: 100; see your DBMS chapter for possible exceptions.

Syntax

ERRLIMIT=*error-limit*;

Details

The ERRLIMIT= statement stops the loading of data after the specified number of DBMS SQL errors has occurred. Errors include an observation that failed to be inserted or a commit that failed to execute.

The *error-limit* argument must be a nonnegative integer. To allow an unlimited number of DBMS SQL errors to occur, specify `errlimit=0`. If the SQL CREATE TABLE statement that is generated by the procedure fails, the procedure terminates.

An editing statement, such as ERRLIMIT=, must be specified after the database connection statements when you create and load a DBMS table. See “LOAD” on page 139 for more information.

LABEL

Causes DBMS column names to default to SAS labels.

Optional statement

Interacts with: RESET

Default: DBMS column names default to SAS variable names

Syntax

LABEL;

Details

The LABEL statement causes the DBMS column names to default to the SAS variable labels when the new table is created. If a SAS variable has no label, the variable name is used. If the label is too long to be a valid DBMS column name, the label is truncated.

For the LABEL statement to take effect, the RESET statement must be used *after* the LABEL statement.

An editing statement, such as LABEL, must be specified after the database connection statements when you create and load a DBMS table. See “LOAD” on page 139 for more information.

LIMIT=

Limits the number of observations that are loaded.

Optional statement

Default: 5000

Syntax

LIMIT=*load-limit*;

Details

The LIMIT= statement places a limit on the number of observations that can be loaded into the new DBMS table. The *load-limit* argument must be a nonnegative integer. To load all the observations from your input data set, specify `limit=0`.

An editing statement, such as LIMIT=, must be specified after the database connection statements when you create and load a DBMS table. See “LOAD” on page 139 for more information.

LIST

Lists information about the variables to be loaded.

Optional statement

Default: ALL

Syntax

LIST <ALL | FIELD | *variable-identifier*>;

The LIST statement lists information about all or some of the SAS variables to be loaded into the new DBMS table. By default, the list is sent to the SAS log.

The LIST statement can take the ALL, FIELD, or *variable-identifier* arguments:

ALL
lists information about all the variables in the input SAS data set, whether or not those variables are selected for the load.

FIELD
lists information about only the input SAS variables that are selected for the load.

variable-identifier
lists information about only the specified variable. The *variable-identifier* argument can be either the SAS variable name or the positional equivalent. The positional equivalent is the number that represents the variable's position in the data set. For example, if you want to list information for the column associated with the third SAS variable, submit the following statement:

```
list 3;
```

Details

You can specify LIST as many times as you want while creating a DBMS table; specify LIST before the LOAD statement to see the entire table. LIST must be specified after the database connection statements. See "LOAD" on page 139 for more information.

LOAD

Creates and loads the new DBMS table.

Required statement for loading or appending data

Syntax

LOAD;

Details

The LOAD statement informs the DBLOAD procedure to execute the action that you request, including loading or appending data. This statement is required to create and load a new DBMS table or to append data to an existing table.

When you create and load a DBMS table, you must place statements or groups of statements in a certain order after the PROC DBLOAD statement and its options, as listed below:

1. Database connection statements: Check your DBMS chapter for the appropriate statements for your DBMS. After the database connection statements, specify the TABLE statement.
2. Editing statements: ACCDESC=, COMMIT=, DELETE, ERRLIMIT=, LABEL, LIMIT=, LIST, NULLS, RENAME, RESET, SQL, TYPE, and WHERE. The order within this group usually does not matter; see the individual statements for more information.
3. Creating and Loading statement: LOAD must appear last before RUN to create and load a table or append data to a table.
4. RUN statement: This statement is used to process the DBLOAD procedure. If you specify QUIT instead of RUN, PROC DBLOAD terminates without completing your request.

Sending a DBMS-specific, Nonquery Statement

If you use the DBLOAD procedure *only* to submit DBMS-specific, nonquery SQL statements to the DBMS (and not to load a table), omit the LOAD statement. The order of the statements listed above is the same. See "SQL" on page 143 for more information on this process.

Example

The following example creates the SUMMERTEMPS table in ORACLE based on the DLIB.TEMPEMPS data file. See Appendix 1, "Sample Data," on page 217 for a description of this file.

```
proc dbload dbms=oracle data=dlib.tempemps;
    user=testuser; password=testpass;
    path='testpath';
    table=summertemps;
    rename firstnam=firstname
           middlena=middlename;
    type hiredate 'date'
         empid 'number(6,0)'
         familyid 'number(6,0)';
    nulls 1=n;
    list;
    load;
run;
```

NULLS

Specifies whether DBMS columns accept null values.

Optional statement
Default: DBMS specific

Syntax

NULLS *variable-identifier-1* = Y | N<…*variable-identifier-n* = Y | N>;

Details

The NULLS statement specifies whether the DBMS columns that are associated with the listed input SAS variables allow null values. Specify **Y** to accept null values. Specify **N** to reject null values and to require data in that column. The default is DBMS specific, although for most DBMSs the default is **Y**.

If you specify **N** for a numeric column, none of the observations that contain missing values in the corresponding SAS variable are loaded into the table, and a message is written to the SAS log. The current error count is increased by one for each observation that is not loaded. See "ERRLIMIT=" on page 137 for more information.

If a character column contains blanks (the SAS missing value) and you have specified **N** for the DBMS column, then blanks are inserted. If you specify **Y**, null values are inserted.

The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, if you want the column that is associated with the third SAS variable to accept null values, submit the following statement:

```
nulls 3=y;
```

If you omit the NULLS statement, the DBMS default action occurs. You can list as many variables as you want in one NULLS statement. If you have previously defined a column as NULLS=N, you can use the NULLS statement to redefine it to accept null values.

An editing statement, such as NULLS, must be specified after the database connection statements when you create and load a DBMS table. See "LOAD" on page 139 for more information.

QUIT

Terminates the procedure.

Control statement

Syntax

QUIT;

Details

The QUIT statement terminates the DBLOAD procedure without further processing. EXIT is an alias for the QUIT statement.

RENAME

Renames DBMS columns.

Optional statement

Interacts with: DELETE, LABEL, RESET

Syntax

RENAME *variable-identifier-1* = <'>*column-name-1*<'>
    <*...variable-identifier-n* = <'>*column-name-n*<'>>;

Details

The RENAME statement changes the names of the DBMS columns that are associated with the listed SAS variables. If you omit the RENAME statement, all the

DBMS column names default to the corresponding SAS variable names (unless the LABEL statement is specified).

The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, if you want to rename the column associated with the third SAS variable, submit the following statement:

```
rename 3=employeename;
```

Note: The *column-name* argument must be a valid DBMS column name. If the column name includes lowercase characters, special characters, or national characters, you must enclose the column name in single or double quotes. If no quotes are used, the DBMS column name is created in uppercase. To preserve case, use the following syntax:

```
rename 3='''employeename'''
```

△

The RENAME statement enables you to include variables that you have previously deleted. For example, suppose you submit the following statements:

```
delete 3;
rename 3=empname;
```

The DELETE statement first drops the third variable. The RENAME statement includes the third variable and assigns the name **`EMPNAME`** and the default column type to it.

You can list as many variables as you want in one RENAME statement. The RENAME statement overrides the LABEL statement for columns that are renamed. COLUMN is an alias for the RENAME statement.

An editing statement, such as RENAME, must be specified after the database connection statements when you create and load a DBMS table. See "LOAD" on page 139 for more information.

RESET

Resets column names and data types to their default values.

Optional statement

Interacts with: DELETE, LABEL, RENAME, TYPE

Syntax

RESET ALL | *variable-identifier-1*
<*...variable-identifier-n*>;

Details

The RESET statement resets the columns that are associated with the listed SAS variables to the default DBMS column name, column data type, and ability to accept null values. If you specify **ALL**, all columns are reset to their default values, and any dropped columns are restored with their default values. The default values are as follows:

column name	defaults to the SAS variable name, or to the SAS variable label (if you have used the LABEL statement).
column type	is generated from the SAS variable format.
nulls	uses the DBMS default value.

The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, if you want to reset the column associated with the third SAS variable, submit the following statement:

```
reset 3;
```

You can reset as many columns as you want in one RESET statement.

The RESET statement must be used after the LABEL statement for the LABEL statement to take effect.

An editing statement, such as RESET, must be specified after the database connection statements when you create and load a DBMS table. See "LOAD" on page 139 for more information.

SQL

Submits a DBMS-specific SQL statement to the DBMS.

Optional statement

Syntax

SQL *DBMS-specific SQL-statement*;

Details

The SQL statement submits a dynamic, nonquery DBMS-specific SQL statement to the DBMS. You can use the DBLOAD statement to submit these DBMS-specific SQL statements whether or not you create and load a DBMS table.

You must enter the keyword SQL before each DBMS-specific SQL statement you submit. The *SQL-statement* argument can be any valid dynamic DBMS-specific SQL statement except the SELECT statement. However, you can enter a SELECT statement as a substatement within another statement, such as in a CREATE VIEW statement. You use DBMS-specific SQL object names and syntax in the DBLOAD SQL statement.

You cannot create a DBMS table and reference it in your DBMS-specific SQL statements within the same PROC DBLOAD step. The new table is not created until the RUN statement is processed.

To submit dynamic, nonquery DBMS-specific SQL statements to the DBMS *without* creating a DBMS table, you use the DBMS= option, any database connection statements, and the SQL statement.

The SQL statement might be case sensitive; see your DBMS chapter for details.

An editing statement, such as SQL, must be specified after the database connection statements.

Example

The following PROC DBLOAD example grants UPDATE privileges to user MARURI on the DB2 SASDEMO.ORDERS table.

```
proc dbload dbms=db2;
   in sample;
   sql grant update on sasdemo.orders to maruri;
run;
```

TABLE=

Names the DBMS table to be created and loaded.

Table statement: varies with each DBMS

Required statement

Syntax

TABLE= <'>*table-name*<'>;

Details

The TABLE= statement specifies the name of the DBMS table to be created and loaded into a DBMS database. The table name must be a valid table name for the DBMS. (See the naming conventions for your DBMS listed in your DBMS chapter.) If it contains lowercase characters, special characters, or national characters, it must be enclosed in quotes.

In addition, you must specify a table name that does not already exist. If a table by that name exists, an error message is written to the SAS log, and the table specified in this statement is not loaded.

When you create and load or append to a DBMS table, the TABLE= statement is required. It must follow other database connection statements such as DATABASE= or USER=.

When you are submitting dynamic DBMS-specific SQL statements to the DBMS without creating and loading a table, this statement is not used.

TYPE

Changes default DBMS data types in the new table.

Optional statement

Syntax

TYPE *variable-identifier-1* = *'column-type-1'*
    <*...variable-identifier-n* = *'column-type-n'*>;

Details

The TYPE statement changes the default DBMS column data types that are associated with the corresponding SAS variables.

The *variable-identifier* argument can be either the SAS variable name or the positional equivalent from the LIST statement. The positional equivalent is the number that represents the variable's place in the data set. For example, if you want to change the data type of the DBMS column associated with the third SAS variable, submit the following statement:

```
type 3='char(17)';
```

The argument *column-type* must be a valid data type for the DBMS and must be enclosed in quotes.

If you omit the TYPE statement, the column data types are generated with default DBMS data types that are based on the SAS variable formats. You can change as many data types as you want in one TYPE statement. See your DBMS chapter for a complete list of the default conversion data types for the DBLOAD procedure.

An editing statement, such as TYPE, must be specified after the database connection statements when you create and load a DBMS table. See "LOAD" on page 139 for more information.

WHERE

Loads a subset of data into the new table.

Optional statement

Syntax

WHERE *SAS-where-expression*;

Details

The WHERE statement causes a subset of observations to be loaded into the new DBMS table. The *SAS-where-expression* must be a valid SAS WHERE statement that uses SAS variable names (not DBMS column names) as defined in the input data set. The following example loads only the observations in which the SAS variable COUNTRY has the value **`Brazil`**.

```
where country='Brazil';
```

For more information on the syntax of the SAS WHERE statement, see *SAS Language Reference: Dictionary*.

An editing statement, such as WHERE, must be specified after the database connection statements when you create and load a DBMS table. See “LOAD” on page 139 for more information.

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