CHAPTER

9

ACCESS Procedure Reference

Introduction

In most of the interfaces that are provided in Version 6 of SAS/ACCESS software, the ACCESS procedure enabled you to create descriptor files that are used to access and update DBMS data in SAS. The ACCESS 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 ACCESS procedure. The PROC ACCESS statement and its options are presented first, followed by the procedure statements. The last section, "Performance and Efficient View Descriptors" on page 125, presents several efficiency considerations for using SAS/ACCESS software.

See your DBMS chapter to determine whether the ACCESS procedure is supported for your DBMS, and for DBMS-specific information pertaining to the ACCESS procedure. Refer to the *SAS Language Reference: Dictionary* and to the *SAS Companion* for your operating system environment for information about 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 new SAS/ACCESS LIBNAME statement instead of PROC ACCESS in order to access your DBMS data more directly and to take full advantage of the Version 7 and later enhancements. See "SAS/ACCESS LIBNAME Statement" on page 27 for more information about the new LIBNAME statement. △

Naming Limits in the ACCESS Procedure

In Version 7 and later, the ACCESS procedure still limits descriptor and SAS variable names to a maximum of eight characters in length. If you need to use 32–character names in SAS, it is recommended that you use the SAS/ACCESS LIBNAME statement or the SQL Procedure Pass-Through Facility to access your DBMS data instead of PROC ACCESS.

If you attempt to use long names in PROC ACCESS, you will get an error message advising you that long names are not supported. Long member names, such as access descriptor and view descriptor names, are truncated to eight characters. Long DBMS column names are truncated to eight–character SAS variable names within the SAS access descriptor. As in previous versions, you can use the PROC ACCESS RENAME= statement to specify eight–character SAS variable names, or accept the default truncated SAS variable names that are assigned by PROC ACCESS.

Case Sensitivity in the ACCESS Procedure

SAS names can be entered in either uppercase or lowercase. However, some host-operating environments and some DBMSs are case sensitive and therefore, special consideration should be used when the names of DBMS objects (such as tables and columns) are used in the SAS System. See the "Naming Conventions" section in your DBMS chapter for more information on case sensitivity.

The ACCESS procedure generally converts DBMS object names to uppercase unless they are enclosed in quotes. Any DBMS objects that were given lowercase names when they were created, or whose names contain special or national characters, must be enclosed in quotes.

ACCESS Procedure Syntax

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

PROC ACCESS<*statement-options*>;

Creating and Updating Statements

CREATE *libref.member-name.*ACCESS | VIEW <*password-option*>;

UPDATE *libref.member-name.*ACCESS | VIEW <*password-option*>;

Database Connection Statements

These statements are used to connect to your DBMS and they 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

ASSIGN <=>YES | NO | Y | N;

DROP <'>*column-identifier-1*<'> <…<'>*column-identifier-n*<'>>;

FORMAT <'>*column-identifier-1*<'> <=>*SAS-format-name-1*
<…<'>*column-identifier-n*<'> <=> *SAS-format-name-n*>;

LIST <ALL | VIEW |<'>*column-identifier*<'>>;

QUIT;

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

RESET ALL |<'>*column-identifier-1*<'> <…<'>*column-identifier-n*<'>>;

SELECT ALL |<'>*column-identifier-1*<'> <…<'>*column-identifier-n*<'>>;

SUBSET *selection-criteria*;

UNIQUE <=> YES | NO | Y | N;

RUN;

Description

You can use the ACCESS procedure to create and update access descriptors, view descriptors, and SAS data files. Descriptor files describe DBMS data so that you can read, update, or extract the DBMS data directly from within a SAS session or in a SAS program.

The following sections provide complete information on PROC ACCESS options and statements.

PROC ACCESS Statement Options

The ACCESS procedure statement takes the following sets of options:

PROC ACCESS <*statement-options*>;

Depending on which options you use, the ACCESS procedure statement performs several tasks. You use the PROC ACCESS statement with database connection statements and certain procedure statements to create and update descriptors or SAS data files from DBMS data. The following sections describe PROC ACCESS options in greater detail.

Options

This section describes the options that you use to create and update access descriptors or to create and update a view descriptor.

See Table 9.2 on page 108 for examples of options and statements. To invoke the ACCESS procedure, you use the options and certain procedure statements. The options and statements that you choose are determined by your task.

ACCDESC=*libref.access-descriptor*
specifies an access descriptor.

ACCDESC= is used with the DBMS= option to create or update a view descriptor that is based on the specified access descriptor. You use a CREATE or UDPATE statement to name and create or update the view. You can also use a SAS data set option on the ACCDESC= option to specify any passwords that have been assigned to the access descriptor. The ACCDESC= option has two aliases: AD= and ACCESS=.

DBMS=*database-management-system*
specifies which database management system you want to use. DBMS= can be used with the ACCDESC= option to create or update a view descriptor, which is then named in the CREATE or UPDATE statement. This option is required. See your DBMS chapter for the value to enter for your DBMS.

OUT=*libref.member-name*
specifies the SAS data file to which DBMS data is output. OUT= is used with the VIEWDESC= option. VIEWDESC= specifies the view descriptor through which you extract the DBMS data.

Note: This option cannot be specified when you create or update a view descriptor. △

VIEWDESC=*libref.view-descriptor*
specifies a view descriptor through which you extract the DBMS data. VIEWDESC= is used with the OUT= option.

For example:

```
proc access viewdesc=mydblib.newstaff
            out=dlib.newstaff;
run;
```

SAS System Passwords for SAS/ACCESS Descriptors

The SAS System enables you to control access to SAS data sets and access descriptors by associating one or more SAS System passwords with them. You must first create the descriptor files before you assign SAS passwords to them, as described in "Assigning Passwords" on page 107.

Table 9.1 on page 106 summarizes the levels of protection that SAS System passwords have and their effects on access descriptors and view descriptors:

Table 9.1 Password and Descriptor Interaction

	READ=	WRITE=	ALTER=
access descriptor	no effect on descriptor	no effect on descriptor	protects descriptor from being read or edited
view descriptor	protects DBMS data from being read or updated	protects DBMS data from being updated	protects descriptor from being read or edited

When you create or update view descriptors, you can use a SAS data set option after the ACCDESC= option to specify the access descriptor's password (if one exists). In this case, you are *not* assigning a password to the view descriptor that is being created or updated; rather, using the password grants you permission to use the access descriptor to create or update the view descriptor. For example:

```
proc access dbms=sybase accdesc=adlib.customer
            (alter=rouge);
  create vlib.customer.view;
  select all;
```

```
run;
```

By specifying the ALTER level of password, you can read the ADLIB.CUSTOMER access descriptor and create the VLIB.CUSTOMER view descriptor.

Assigning Passwords

To assign, change, or delete a SAS password, you can also use the DATASETS procedure's MODIFY statement or the global SETPASSWORD command, which opens a dialog box. Here is the basic syntax for using PROC DATASETS to assign a password to an access descriptor, a view descriptor, or a SAS data file:

PROC DATASETS LIBRARY=*libref* MEMTYPE=*member-type*;
    MODIFY *member-name* (*password-level* = *password-modification*);

RUN;

In this syntax statement, the *password-level* argument can have one or more of the following values: READ=, WRITE=, ALTER=, or PW=. PW= assigns read, write, and alter privileges to a descriptor or data file. The *password-modification* argument enables you to assign a new password or to change or delete an existing password. For example, this PROC DATASETS statement assigns the password MONEY with the ALTER level of protection to the access descriptor ADLIB.SALARIES.

```
proc datasets library=adlib memtype=access;
   modify salaries (alter=money);
run;
```

In this case, users are prompted for the password whenever they try to browse or update the access descriptor or to create view descriptors that are based on ADLIB.SALARIES.

You can assign multiple levels of protection to a descriptor or SAS data file. In the next example, the PROC DATASETS statement assigns the passwords MYPW and MYDEPT with READ and ALTER levels of protection to the view descriptor VLIB.JOBC204:

```
proc datasets library=vlib memtype=view;
   modify jobc204 (read=mypw alter=mydept);
run;
```

In this case, users are prompted for the SAS password when they try to read the DBMS data, or try to browse or update the view descriptor VLIB.JOBC204. You need both levels to protect the data and descriptor from being read. However, a user could still update the data accessed by VLIB.JOBC204, for example, by using a PROC SQL UPDATE. Assign a WRITE level of protection to prevent data updates.

Note: When you assign multiple levels of passwords, use a different password for each level to ensure that you grant only the access privileges that you intend. △

To delete a password on an access descriptor or any SAS data set, put a slash after the password:

```
proc datasets library=vlib memtype=view;
   modify jobc204 (read=mypw/ alter=mydept/);
run;
```

Procedure Statements

To invoke the ACCESS procedure, you use the options described in "Options" on page 105 and certain procedure statements. The options and statements that you choose are determined by your task, as summarized in Table 9.2 on page 108. These statements vary per DBMS and might be optional; see your DBMS chapter for more information. The new SAS/ACCESS data set options that are described in "SAS/ACCESS Data Set Options" on page 43 are available for use with view descriptors, where applicable.

Table 9.2 Options and Statements Required for the ACCESS Procedure

Tasks	Options and Statements You Use
create an access descriptor	**PROC ACCESS** *statement-options*; **CREATE** *libref.member-name*.ACCESS; *database-connection-statements*; *editing-statements*; **RUN**;
create an access descriptor and a view descriptor	**PROC ACCESS** *statement-options*; **CREATE** *libref.member-name*.ACCESS; *database-connection-statements*; *editing-statements*; **CREATE** *libref.member-name*.VIEW; **SELECT** *column-list*; *editing-statements*; **RUN**;
create a view descriptor from an existing access descriptor	**PROC ACCESS** *statement-options, including* ACCDESC=*libref.access-descriptor*; **CREATE** *libref.member-name*.VIEW; **SELECT** *column-list*; *editing-statements*; **RUN**;
update an access descriptor	**PROC ACCESS***statement-options*; **UPDATE** *libref.member-name*.ACCESS; *database-connection-statements*; *editing-statements*; **RUN**;
update an access descriptor and a view descriptor	**PROC ACCESS** *statement-options*; **UPDATE** *libref.member-name*.ACCESS; *database-connection-statements*; *editing-statements*; **UPDATE** *libref.member-name*.VIEW; *editing-statements*; **RUN**;

Tasks	Options and Statements You Use
update an access descriptor and create a view descriptor	**PROC ACCESS** *statement-options*; **UPDATE** *libref.member-name*.ACCESS; *database-connection-statements*; *editing-statements*; **CREATE** *libref.member-name*.VIEW; **SELECT** *column-list*; *editing-statements*; **RUN**;
update a view descriptor from an existing access descriptor	**PROC ACCESS** *statement-options, including* ACCDESC=*libref.access-descriptor*; **UPDATE** *libref.member-name*.VIEW; *editing-statements*; **RUN**;
create a SAS data set from a view descriptor	**PROC ACCESS** *statement-options, including* DBMS=*dbms-name*; VIEWDESC=*libref.member*; OUT=*libref.member*; **RUN**;

To review the contents of an existing view descriptor, you can use the CONTENTS procedure. For an example, see "Reviewing Variables" on page 186.

To review the contents of an access descriptor, you can use the LIST statement in PROC ACCESS. In this Oracle Rdb example, the LIST statement displays all of the variables in the newly created access descriptor, ADLIB.EMPLOY.

```
   /* create access descriptor  */

proc access dbms=rdb;
   create adlib.employ.access;
   database='qa:[dubois]textile';
   table=employees;
   assign=no;
   list all;
```

You can create or update one or more access descriptors and view descriptors in one execution of PROC ACCESS, or you can create or update the descriptors in separate executions.

To create descriptors, the procedure statements must be coded in a particular order. See "CREATE" on page 111 for information on this order and for information on database connection and editing statements. The PROC ACCESS statements are described in alphabetic order in the following sections.

ASSIGN

Indicates whether SAS variable names and formats are generated.

Optional statement

Applies to: access descriptor
Interacts with: FORMAT, RENAME, RESET, UNIQUE
Default: NO

Syntax

ASSIGN <=>YES | NO | Y | N;

Details

The ASSIGN statement indicates whether SAS variable names are automatically generated and whether users can change SAS variable names and formats in the view descriptors created from this access descriptor.

An editing statement, such as ASSIGN, must be specified after the CREATE and database connection statements when you create an access descriptor. See "CREATE" on page 111 for more information.

Note: The ASSIGN statement cannot be used with the UPDATE statement. △

The value **NO** (or **N**) enables you to modify SAS variable names and formats when you create an access descriptor and when you create view descriptors that are based on this access descriptor. During an access descriptor's creation, you use the RENAME statement to change SAS variable names; you use the FORMAT statement to change SAS formats.

Specify a **YES** (or **Y**) value for this statement to generate unique SAS variable names from the first eight characters of the DBMS column names, according to the following rules. With **YES**, you can change the SAS variable names only in the access descriptor. The SAS variable names that are saved in an access descriptor are *always* used when view descriptors are created from the access descriptor; you cannot change them in the view descriptors. Note that the ACCESS procedure only allows names up to eight characters.

Default SAS variable names are generated according to these rules:

- If the column name is longer than eight characters, the SAS System uses only the first eight characters. If truncating results in duplicate names, numbers are appended to the ends of the names. For example, the DBMS names **clientsname** and **clientsnumber** become the SAS names **clientsn** and **clients0**.

  If the same descriptor has another set of columns with duplicate names, the numeric suffix begins at the next highest number from the previous set of duplicate names. For example, if the descriptor has the duplicate names above and also the DBMS names **customername**, **customernumber**, and **customernode**, the default SAS names would be **customer**, **custome1**, and **custome2**.
- If the column name contains characters that are invalid in SAS names (including national characters), the SAS System replaces the invalid characters with underscores (_). For example, the column name **func$** becomes the SAS variable name **func_**.

If you specify **YES** for this statement, the SAS System automatically resolves any duplicate variable names. However, if you specify **YES**, you cannot specify the RENAME, FORMAT, RESET or UNIQUE statements when you create view descriptors that are based on the access descriptor.

When the SAS/ACCESS interface encounters the next CREATE statement to create an access descriptor, the ASSIGN statement is reset to the default **NO** value.

AN is the alias for the ASSIGN statement.

CREATE

Creates a SAS/ACCESS descriptor file.

Required statement

Applies to: access descriptor or view descriptor

Syntax

CREATE *libref.member-name.*ACCESS | VIEW;

Details

The CREATE statement identifies the access descriptor or view descriptor that you want to create. This statement is required for creating a descriptor.

To create a descriptor, use a three-level name. The first level identifies the libref of the SAS data library where you will store the descriptor. You can store the descriptor in a temporary (WORK) or permanent SAS data library. The second level is the descriptor's name (member name). The third level is the type of SAS file: specify **ACCESS** for an access descriptor or **VIEW** for a view descriptor.

You can use the CREATE statement as many times as necessary in one procedure execution. That is, you can create multiple access descriptors, as well as one or more view descriptors based on these access descriptors, within the same execution of the ACCESS procedure. Or, you can create access descriptors and view descriptors in separate executions of the procedure.

Access descriptors

When you create an access descriptor, you must place statements or groups of statements in a certain order after the PROC ACCESS statement and its options, as listed below:

1. CREATE statement for the access descriptor: this statement must always follow the PROC ACCESS statement if a descriptor is being created.
2. Database connection statements: DBMS-specific statements for your DBMS. Information from database connection statements is stored in an access descriptor. Therefore, you do not repeat this information when you create view descriptors. See your DBMS chapter for information on database connection statements for your DBMS.
3. TABLE statement and/or editing statements: ASSIGN, DROP, FORMAT, LIST, RENAME, RESET.
4. Specify the RUN statement to execute the ACCESS procedure. If you specify QUIT instead of RUN, PROC ACCESS terminates without creating your descriptor. Alternately, you can specify another CREATE or UPDATE statement to

execute the previous CREATE or UPDATE statement; your changes are saved when a new CREATE, UPDATE, RUN, or other SAS statement is executed.

The order of the statements within the database connection group does not matter. The order of the statements within the editing group sometimes matters; see the individual statement descriptions for any restrictions.

Note: Altering a DBMS table that has descriptor files defined on it might invalidate these files or cause them to be outdated. If you re-create a table, add a new column to a table, or delete an existing column from a table, use the UDPATE statement to modify your descriptors to use the new information. △

View descriptors

You can create view descriptors and access descriptors in the same execution of the ACCESS procedure or in separate executions.

To create a view descriptor and the access descriptor on which it is based within the *same* PROC ACCESS execution, you must place the statements or groups of statements in a particular order after the PROC ACCESS statement and its options, as listed below:

1. Create the access descriptor as described in "Access descriptors" on page 111, but do not include the RUN statement.
2. CREATE statement for the view descriptor: this statement must follow the PROC ACCESS statements that created the access descriptor.
3. Editing statements: SELECT, SUBSET, and UNIQUE are used only when creating view descriptors. FORMAT, LIST, RENAME, and RESET are used when creating both view and access descriptors. FORMAT, RENAME, and UNIQUE can be specified only when ASSIGN=NO is specified in the access descriptor referenced by this view descriptor. QUIT is also an editing statement but using it terminates PROC ACCESS without creating your descriptor.

   The order of the statements within this group usually does not matter; see the individual statement descriptions for any restrictions.

   Note: You cannot use the DROP statement when you create a view descriptor. Instead, use SELECT to specify the columns you want. △
4. Specify the RUN statement to execute the ACCESS procedure. If you specify QUIT instead of RUN, PROC ACCESS terminates without creating your descriptor. Alternately, you can specify another CREATE or UPDATE statement to execute the previous CREATE or UPDATE statement; your changes are saved when a new CREATE, UPDATE, RUN, or other SAS statement is executed.

To create a view descriptor based on an access descriptor that was created in a *separate* PROC ACCESS step, you specify the access descriptor's name in the ACCDESC= option in the new PROC ACCESS statement. You must specify the CREATE statement before any of the editing statements for the view descriptor.

If you create only one descriptor in a PROC step, the CREATE statement and its accompanying statements are checked for errors when you submit PROC ACCESS for processing. If you create multiple descriptors in the same PROC step, each CREATE statement (and its accompanying statements) is checked for errors as it is processed.

When the RUN statement is processed, all descriptors are saved if no errors are found. If errors are found, error messages are written to the SAS log, processing is terminated, and the descriptors are not saved. After you correct the errors, resubmit your statements.

Example

The following example creates an access descriptor ADLIB.EMPLOY on the Oracle Rdb table EMPLOYEES and a view descriptor VLIB.EMP1204 based on ADLIB.EMPLOY in the same PROC ACCESS step.

```
proc access dbms=rdb;

   /* create access descriptor  */

   create adlib.employ.access;
   database='qa:[dubois]textile';
   table=employees;
   assign=no;
   list all;

   /* create view descriptor  */

   create vlib.emp1204.view;
   select empid lastname hiredate salary dept
   gender birthdate;
   format empid 6.
          salary dollar12.2
          jobcode 5.
          hiredate datetime9.
          birthdate datetime9.;
   subset where jobcode=1204;
run;
```

The following example creates a view descriptor VLIB.BDAYS from the ADLIB.EMPLOY access descriptor, which was created in the previous PROC ACCESS step.

```
proc access dbms=rdb accdesc=adlib.employ;
   create vlib.bdays.view;
   select empid lastname birthdate;
   format empid 6.
          birthdate datetime7.;
run;
```

DROP

Drops a column so that it cannot be selected in a view descriptor.

Optional statement

Applies to: access descriptor

Interacts with: RESET, SELECT

Syntax

DROP <'>*column-identifier-1*<'>
<…<'>*column-identifier-n*<'>>;

Details

The DROP statement drops the specified column from an access descriptor. Therefore, the column cannot be selected by a view descriptor that is based on the access descriptor. However, the specified column in the DBMS table remains unaffected by this statement.

An editing statement, such as DROP, must follow the CREATE or UPDATE and database connection statements when you create or update an access descriptor. See "CREATE" on page 111 for more information on the order of statements.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to drop the third and fifth columns, submit the following statement:

```
drop 3 5;
```

If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotes. You can drop as many columns as you want in one DROP statement.

To display a column that was previously dropped, specify that column name in the RESET statement. However, doing so also resets all the column's attributes (such as SAS variable name, format, and so on) to their default values.

FORMAT

Changes a SAS format for a DBMS column.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, DROP, RESET

Syntax

FORMAT <'>*column-identifier-1*<'>
<=>*SAS-format-name-1*
<…<'>*column-identifier-n*<'>
<=>*SAS-format-name-n*>;

Details

The FORMAT statement changes a SAS variable format from its default format; the default SAS variable format is based on the data type of the DBMS column. (See your DBMS chapter for information on the default formats that the SAS System assigns to your DBMS data types.)

An editing statement, such as FORMAT, must follow the CREATE or UPDATE statement and the database connection statements when you create or update a descriptor. See "CREATE" on page 111 for more information on the order of statements.

The *column-identifier* argument can be either the column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to associate the DATE9. format with the BIRTHDATE column and with the second column in the access descriptor, submit the following statement:

```
format 2=date9. birthdate=date9.;
```

The column identifier is specified on the left and the SAS format on the right of the expression. The equal sign (=) is optional. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotes. You can enter formats for as many columns as you want in one FORMAT statement.

The following example creates the access descriptor ADLIB.PRODUCT on the DB2 table SASDEMO.SPECPROD. The FORMAT statement is used to specify new SAS variable formats for four columns from the DBMS table.

```
proc access dbms=db2;
   create adlib.product.access;
   ssid=db2a;
   table=sasdemo;
   assign=yes;
   rename productid prodid
          fibername fiber;
   format productid  4.
          weight     e16.9
          fibersize  e20.13
          width      e16.9;
run;
```

You can use the FORMAT statement with a view descriptor only if the ASSIGN statement that was used when creating the access descriptor was specified with the **NO** value.

FMT is the alias for the FORMAT statement.

Note: When you use the FORMAT statement with access descriptors, the FORMAT statement also reselects columns that were previously dropped with the DROP statement. △

LIST

Lists columns in the descriptor and gives information about them.

Optional statement

Applies to: access descriptor or view descriptor

Default: ALL

Syntax

LIST <ALL | VIEW |<'>*column-identifier*<'>>;

Details

The LIST statement lists columns in the descriptor along with information about the columns. The LIST statement can be used when creating an access descriptor or a view descriptor. The LIST information is written to your SAS log.

If you use an editing statement, such as LIST, it must follow the CREATE or UPDATE statement and the database connection statements when you create or update a descriptor. You can specify LIST as many times as you want while creating or updating a descriptor; specify LIST last in your PROC ACCESS code to see the entire descriptor. Or, if you are creating or updating multiple descriptors, specify LIST before the next CREATE or UPDATE statement in order to list all the information about the descriptor you are creating or updating.

The LIST statement can take one or more of the following arguments:

ALL
lists all the DBMS columns in the table, the positional equivalents, the SAS variable names, and the SAS variable formats that are available for a descriptor. When you are creating an access descriptor, ***NON-DISPLAY*** appears next to the column description for any column that has been dropped; ***UNSUPPORTED*** appears next to any column whose data type is not supported by your DBMS interface view engine. When you are creating a view descriptor, ***SELECTED*** appears next to the column description for columns that you have selected for the view.

VIEW
lists all the DBMS columns that are selected for the view descriptor, along with their positional equivalents, their SAS names and formats, and any subsetting clauses. Any columns that were dropped in the access descriptor are not displayed. The VIEW argument is valid only for a view descriptor.

column-identifier
lists the specified DBMS column name, its positional equivalent, its SAS variable name and format, and whether the column has been selected. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotes.

The *column-identifier* argument can be either the column name or the positional equivalent, which is the number that represents the column's place in the descriptor. For example, to list information about the fifth column in the descriptor, submit the following statement:

```
list 5;
```

You can use one or more of these previously described arguments in a LIST statement, in any order.

QUIT

Terminates the procedure.

Control statement

Applies to: access descriptor or view descriptor

Syntax

QUIT;

Details

The QUIT statement terminates the ACCESS procedure without any further descriptor creation. Changes made since the last CREATE, UPDATE, or RUN statement are not saved; changes are saved only when a new CREATE, UPDATE, or RUN statement is submitted.

EXIT is an alias for the QUIT statement.

RENAME

Modifies the SAS variable name.

Optional statement

Applies to: access descriptor or view descriptor

Interacts with: ASSIGN, RESET

Syntax

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

Details

The RENAME statement sets or modifies the SAS variable name that is associated with a DBMS column. The RENAME statement can be used when creating an access descriptor or a view descriptor.

An editing statement, such as RENAME, must follow the CREATE or UPDATE statement and the database connection statements when you create or update a descriptor. See "CREATE" on page 111 for more information on the order of statements.

Two factors affect the use of the RENAME statement: whether you specify the ASSIGN statement when you are creating an access descriptor, and the kind of descriptor you are creating.

- If you omit the ASSIGN statement or specify it with a **NO** value, the renamed SAS variable names that you specify in the access descriptor are retained throughout an ACCESS procedure execution. For example, if you rename the CUSTOMER column to CUSTNUM when you create an access descriptor, that column continues to be named CUSTNUM when you select it in a view descriptor unless a RESET statement or another RENAME statement is specified.

  When creating a view descriptor that is based on this access descriptor, you can specify the RESET statement or another RENAME statement to rename the variable again, but the new name applies only in that view. When you create other view descriptors, the SAS variable names are derived from the access descriptor.

- If you specify the **YES** value in the ASSIGN statement, you can use the RENAME statement to change SAS variable names only while creating an access descriptor. As described earlier in the ASSIGN statement, SAS variable names and formats that are saved in an access descriptor are always used when creating view descriptors that are based on it.

The *column-identifier* argument can be either the DBMS column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the descriptor. For example, to rename the SAS variable names that are associated with the seventh column and the nine-character FIRSTNAME column in a descriptor, submit the following statement:

```
rename 7 birthdy  'firstname'=fname;
```

The DBMS column name (or positional equivalent) is specified on the left side of the expression, with the SAS variable name on the right side. The equal sign (=) is optional. If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotes. You can rename as many columns as you want in one RENAME statement.

When you are creating a view descriptor, the RENAME statement automatically selects the renamed column for the view. That is, if you rename the SAS variable associated with a DBMS column, you do not have to issue a SELECT statement for that column.

RESET

Resets DBMS columns to their default settings.

Optional statement
Applies to: access descriptor or view descriptor
Interacts with: ASSIGN, DROP, FORMAT, RENAME, SELECT

Syntax

RESET ALL |<'>*column-identifier-1*<'> <…<'>*column-identifier-n*<'>>;

Details

The RESET statement resets either the attributes of all the columns or the attributes of the specified columns to their default values. The RESET statement can be used when creating an access descriptor or a view descriptor. However, this statement has different effects on access and view descriptors, as described in the following sections.

If you use an editing statement, such as RESET, it must follow the CREATE statement and the database connection statements when you create a descriptor. See "CREATE" on page 111 for more information on the order of statements.

Note: The RESET statement cannot be used with the UPDATE statement. △

Access descriptors

When you create an access descriptor, the default setting for a SAS variable name is a blank. However, if you have previously entered or modified any of the SAS variable

names, the RESET statement resets the modified names to the default names that are generated by the ACCESS procedure. How the default SAS variable names are set depends on whether you included the ASSIGN statement. If you omitted ASSIGN or set it to **NO**, the default names are blank. If you set **ASSIGN=YES**, the default names are the first eight characters of each DBMS column name.

The current SAS variable format is also reset to the default SAS format, which was determined from the column's data type. Any columns that were previously dropped, but that are specified in the RESET statement, become available; they can be selected in view descriptors that are based on this access descriptor.

View descriptors

When you create a view descriptor, the RESET statement clears any columns that were included in the SELECT statement (that is, it "de-selects" the columns).

When creating the view descriptor, if you reset a SAS variable and then select it again within the same procedure execution, the SAS variable names and formats are reset to their default values, which are generated from the column names and data types. This applies only if you have omitted the ASSIGN statement or set the value to **NO** when you created the access descriptor on which the view descriptor is based. If you specified **ASSIGN=YES** when you created the access descriptor, the RESET statement has no effect on the view descriptor.

The RESET statement can take ALL or column identifiers as arguments:

ALL
: for access descriptors, resets all the DBMS columns that have been defined to their default names and format settings and reselects any dropped columns.

  For view descriptors, ALL resets all the columns that have been selected so that no columns are selected for the view; you can then use the SELECT statement to select new columns. See "SELECT" on page 119 for more information on that statement.

column-identifier
: can be either the DBMS column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor. For example, to reset the SAS variable name and format associated with the third column, submit the following statement:

```
reset 3;
```

  If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotes. You can reset as many columns as you want in one RESET statement, or use the ALL option to reset all the columns.

  When creating or updating an access descriptor, the *column-identifier* is reset to its default name and format settings. When creating a view descriptor, the specified column is no longer selected for the view.

SELECT

Selects DBMS columns for the view descriptor.

Optional statement

Applies to: view descriptor

Interacts with: RESET

Syntax

SELECT ALL |<'>*column-identifier-1*<'>
<…<'>*column-identifier-n*<'>>;

Details

The SELECT statement specifies which DBMS columns in the access descriptor to include in the view descriptor. This is a required statement and is used only when defining view descriptors.

If you use an editing statement, such as SELECT, it must follow the CREATE statement when you create a view descriptor. See "CREATE" on page 111 for more information on the order of statements.

Note: The SELECT statement cannot be used with the UPDATE statement. △

The SELECT statement can take ALL or column identifiers as arguments:

ALL
includes in the view descriptor all the columns that were defined in the access descriptor and that were not dropped.

column-identifier
can be either the DBMS column name or the positional equivalent from the LIST statement, which is the number that represents the column's place in the access descriptor on which the view is based. For example, to select the first three columns, submit the following statement:

```
select 1 2 3;
```

If the column name contains lowercase characters, special characters, or national characters, enclose the name in quotes. You can select as many columns as you want in one SELECT statement.

SELECT statements are cumulative within the same view creation. That is, if you submit the following two SELECT statements, columns 1, 5, and 6 are selected, not just columns 5 and 6:

```
select 1;
select 5 6;
```

To clear all your current selections when creating a view descriptor, use the **RESET ALL** statement; you can then use another SELECT statement to select new columns.

SUBSET

Adds or modifies selection criteria for a view descriptor.

Optional statement

Applies to: view descriptor

Syntax

SUBSET *selection-criteria*;

Details

You use the SUBSET statement to specify selection criteria when you create a view descriptor. This statement is optional; if you omit it, the view retrieves all the data (that is, all the rows) in the DBMS table.

An editing statement, such as SUBSET, must follow the CREATE or UPDATE statement when you create or update a view descriptor. See "CREATE" on page 111 for more information on the order of statements.

The *selection-criteria* argument can be one or more DBMS-specific SQL expressions that are accepted by your DBMS, such as WHERE, ORDER BY, HAVING, and GROUP BY. You use DBMS column names, not SAS variable names, in your selection criteria. For example, for a view descriptor that retrieves rows from a DBMS table, you could submit the following SUBSET statement:

```
subset where firstorder is not null;
```

If you have multiple selection criteria, enter them all in one SUBSET statement, as in the following example:

```
subset where firstorder is not null
  and country = 'USA'
  order by country;
```

Unlike other ACCESS procedure statements, the SUBSET statement may be case sensitive. The SQL statement is sent to the DBMS exactly as you type it. Therefore, you must use the correct case for any DBMS object names. See your DBMS chapter for details.

The SAS System does not check the SUBSET statement for errors. The statement is verified only when the view descriptor is used in a SAS program.

If you specify more than one SUBSET statement per view descriptor, the last SUBSET overwrites the earlier SUBSETs. To delete the selection criteria, submit a SUBSET statement without any arguments.

TABLE=

Identifies the DBMS table on which the access descriptor is based.

Required statement with the CREATE statement; optional with the UPDATE statement

Table statement: varies with each DBMS

Applies to: access descriptor

Syntax

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

Details

The TABLE= statement specifies the name of the DBMS table on which the access descriptor is based. The *table-name* argument must be a valid DBMS table name. If it contains lowercase characters, special characters, or national characters, you must enclose it in quotes. See your DBMS chapter for details on the TABLE= statement.

When you use database connection statements or the TABLE= statement to create or update an access descriptor, you must specify them after the CREATE or UPDATE statement. See "CREATE" on page 111 for more information on the order of statements.

UNIQUE

Generates SAS variable names based on DBMS column names.

Optional statement

Applies to: view descriptor

Interacts with: ASSIGN

Syntax

UNIQUE <=> YES | NO | Y | N;

Details

The UNIQUE statement specifies whether the SAS/ACCESS interface should generate unique SAS variable names for DBMS columns for which SAS variable names have not been entered.

An editing statement, such as UNIQUE, must follow the CREATE statement when you create a view descriptor. See "CREATE" on page 111 for more information on the order of statements.

Note: The UNIQUE statement cannot be used with the UPDATE statement. △

The UNIQUE statement is affected by whether you specified the ASSIGN statement when you created the access descriptor on which the view is based, as follows:

- If you specified the **ASSIGN=YES** statement, you cannot specify UNIQUE when creating a view descriptor. **YES** causes the SAS System to generate unique names, so UNIQUE is not necessary.
- If you omitted the ASSIGN statement or specified **ASSIGN=NO**, you must resolve any duplicate SAS variable names in the view descriptor. You can use UNIQUE to generate unique names automatically, or you can use the RENAME statement to resolve duplicate names yourself. See "RENAME" on page 117 for information on that statement.

If duplicate SAS variable names exist in the access descriptor on which you are creating a view descriptor, you can specify UNIQUE to resolve the duplication. When you specify **UNIQUE=YES**, the SAS/ACCESS interface appends numbers to any duplicate SAS variable names, thus making each variable name unique. (See the rules for default SAS names in "ASSIGN" on page 109.)

If you specify **UNIQUE=NO**, the SAS/ACCESS interface continues to allow duplicate SAS variable names to exist. You must resolve these duplicate names before saving (and thereby creating) the view descriptor.

Note: It is recommended that you use the UNIQUE statement and specify **UNIQUE=YES** . If you omit it or specify **UNIQUE=NO** and the SAS System encounters duplicate SAS variable names in a view descriptor, your job fails. △

The equals sign (=) is optional in the UNIQUE statement. UN is the alias for UNIQUE.

UPDATE

Updates a SAS/ACCESS descriptor file.

Optional statement

Applies to: access descriptor or view descriptor

Syntax

UPDATE *libref.member-name.*ACCESS | VIEW;

Details

The UPDATE statement identifies an existing access descriptor or view descriptor that you want to update. UPDATE is normally used to update database connection information, such as user names and passwords; if your descriptor requires many changes, it might be easier to use the CREATE statement to overwrite the old descriptor with a new one.

The descriptor that you update can exist in either a temporary (WORK) or permanent SAS data library. If the descriptor has been protected with a SAS password that prohibits editing of the ACCESS or VIEW descriptor, then the password must be specified on the UPDATE statement.

To update a descriptor, use its three-level name. The first level identifies the libref of the SAS data library where you stored the descriptor. The second level is the descriptor's name (member name). The third level is the type of SAS file.

You can use the UPDATE statement as many times as necessary in one procedure execution. That is, you can update multiple access descriptors, as well as one or more view descriptors based on these access descriptors, within the same execution of the ACCESS procedure. Or, you can update access descriptors and view descriptors in separate executions of the procedure.

Rules that applied to the CREATE statement under Version 6 of SAS/ACCESS software apply to the UPDATE statement. For example, the SUBSET statement is valid only for updating view descriptors and it is not valid for access descriptors.

Note: The following statements are not supported when using the UPDATE statement: ASSIGN, RESET, SELECT, and UNIQUE. △

Updating access and view descriptors

You can update view descriptors and access descriptors in the same execution of the ACCESS procedure or in separate executions.

When you update an access or view descriptor, you must place statements or groups of statements in a certain order after the PROC ACCESS statement and its options, as listed below:

1. UPDATE is usually the first statement after the PROC ACCESS statement or in a code block within a PROC ACCESS statement. Usually, you update a descriptor that already exists; however, you can create and update a descriptor in the same PROC ACCESS step.
2. Next, specify any information that you want to update, including any database connection statements, the TABLE statement, or any editing statements. If you are updating an access descriptor, you can use the DROP, FORMAT, LIST, or RENAME editing statements. If you are updating a view descriptor, you can use the DROP, FORMAT, LIST, RENAME, and SUBSET editing statements. FORMAT and RENAME can be specified only when ASSIGN=NO is specified in the access descriptor referenced by the view descriptor you are updating. The order of the statements within this group usually does not matter; see the individual statement descriptions for any restrictions.

   The following editing statements are not allowed when you specify the UPDATE statement: SELECT, RESET, ASSIGN, and UNIQUE. LIST can only be used with views.
3. Specify the RUN statement to execute the ACCESS procedure. If you specify QUIT instead of RUN, PROC ACCESS terminates without updating your descriptor. Alternately, you can specify another CREATE or UPDATE statement to execute the previous CREATE or UPDATE statement; your changes are saved when a new CREATE, UPDATE, or RUN statement is entered.

Note: Altering a DBMS table that has descriptor files defined on it might invalidate these files or cause them to be outdated. If you recreate a table, add a new column to a table, or delete an existing column from a table, use the UDPATE statement to modify your descriptors to use the new information. △

Examples

The following example updates an access descriptor ADLIB.EMPLOY on the Oracle Rdb table EMPLOYEES and then re-creates a view descriptor VLIB.EMP1204, which was based on ADLIB.EMPLOY. The original access descriptor included all of the columns in the table. Using the LIST statement enables you to write all of the variables to the SAS log so you can see the complete access descriptor before you update it.

In this example, the SALARY and BIRTHDATE columns are dropped from the access descriptor so that users cannot see this data. Because SELECT and RESET are not supported when UPDATE is used, the view descriptor VLIB.EMP1204 must be re-created to omit the SALARY and BIRTHDATE columns.

```
proc access dbms=rdb;

   /* update access descriptor  */

   update adlib.employ.access;
   drop salary birthdate;
   list all;

   /* re-create view descriptor  */

   create vlib.emp1204.view;
   select empid hiredate dept jobcode gender
      lastname firstname middlename phone;
   format empid 6.
          jobcode 5.
          hiredate datetime9.;
   subset where jobcode=1204;
run;
```

The following example updates a view descriptor VLIB.BDAYS from the ADLIB.EMPLOY access descriptor, which was created previously. In this example, the WHERE clause replaces the WHERE clause that was specified in the original view descriptor. The SUBSET statement contains an ORACLE-specific SQL statement.

```
proc access dbms=oracle;
   update vlib.bdays.view;
   subset where hiredate=
      to_date('10OCT1988','ddmonyyyy');
run;
```

Performance and Efficient View Descriptors

When you create, update, and use view descriptors, follow these guidelines to minimize the use of SAS System resources and to reduce the time it takes the DBMS to access data.

General Information

When you create or update view descriptors, select only the columns that your program needs. Selecting unnecessary columns adds extra processing time due to data conversions.

When you use a view descriptor in a DATA step or SAS procedure, columns that you specified in the VAR and KEEP statements are passed to the DBMS for processing. Columns that you specified in DROP statements are not passed to the DBMS. Therefore, only a subset of the columns is returned to the SAS System, and performance is usually enhanced. (This approach can be applied in some of DBMSs, such as CA-OpenIngres, ORACLE, Oracle Rdb, and SYBASE.)

Sorting DBMS Data

Sorting DBMS data can be resource-intensive, whether it is done using the SORT procedure, a BY statement, or an ORDER BY clause in a view descriptor or SQL

procedure. You should sort data only when sorted data are needed for your program. The following list includes guidelines and information about sorting data:

- If you specify a BY statement in a DATA or PROC step that references a view descriptor, be sure that the BY variable is associated with an indexed DBMS column.

  If you reference a view descriptor in a SAS program and the program includes a BY statement for a variable that corresponds to a column in the DBMS table, the SAS/ACCESS interface view engine automatically generates an ORDER BY clause for that variable. Thus, the ORDER BY clause causes the DBMS to sort the data before the SAS procedure or DATA step uses the data in a SAS program. If the DBMS table is very large, this sorting can adversely affect your performance. Use a BY variable that is based on an indexed DBMS column to help reduce this negative impact.

- The outermost BY or ORDER BY clause overrides any embedded BY or ORDER BY clauses, including those specified in the selection criteria.

  For example, if the view descriptor has an ORDER BY clause and you have specified a BY statement in your SAS program, the BY statement overrides the view descriptor's ORDER BY clause. If you use an ORDER BY clause in an SQL procedure statement that references a view descriptor, this ORDER BY clause also overrides the view descriptor's ORDER BY clause.

Extracting Data Using a View

In some cases, it might be more efficient to use a view descriptor to extract (that is, copy) DBMS data and place it in a SAS data file instead of using the view descriptor to read the data directly.

A DBMS table is read every time a view descriptor is referred to in a SAS program and the program is executed; the program's output reflects the latest updated level of the DBMS table. If many users are reading the same DBMS table repeatedly, DBMS performance may decrease. If you create several reports during the same SAS session, they may not be based on the same DBMS data due to updating by other users. Therefore, in the following circumstances, it is better to extract data:

- Extract DBMS data if the table is large and you use the data repeatedly in SAS programs.

  If a view descriptor describes a large DBMS table and you plan to use the same DBMS data in several procedures or DATA steps during the same SAS session, you might improve performance by extracting the data. Placing the data into a SAS data file requires a certain amount of disk space to store the data and I/O to write the data. However, SAS data files are organized to provide optimal performance with PROC and DATA steps. Programs that use SAS data files are often more efficient than programs that read DBMS data directly.

  (Exception: If you are using a SAS WHERE statement to create small subsets of data and you are reading the subsets based on indexed columns, it may be better not to extract the data from a large DBMS table. In this case, the DBMS can probably retrieve subsets of data based on indexed columns faster.)

- Extract DBMS data if you use sorted data several times in a SAS program.

  If you intend to use DBMS data in a particular sorted order several times, run the SORT procedure on the view descriptor by using the OUT= option to extract the data. This OUT= option is required whenever PROC SORT references a view descriptor. PROC SORT sends an ORDER BY clause to the DBMS so that the data is returned in sorted order to the SAS data file. PROC SORT does not perform any additional sorting of the data. Extracting the data in this way is more

efficient than requesting the same sort repeatedly (with an ORDER BY clause) on the DBMS data.

- Extract DBMS data for added security.

  If you are the owner of a DBMS table and do not want anyone else to read the data, you might want to extract the data (or a subset of the data) and not distribute information about either the access descriptor or view descriptor. Or, you might want to assign DBMS security features to your DBMS tables to prevent unauthorized reading or writing to them.

  On the SAS System side, you might also want to assign SAS System passwords to your descriptors for additional security. If a view descriptor has a password assigned to it and you extract the data, the new SAS data file is automatically assigned the same password. If a view descriptor does not have a password, you can assign a password to the extracted SAS data file. You can assign a password by using PROC DATASETS on your descriptor. See the *SAS Procedures Guide* for more information.

Using a Subset of the DBMS Data

When you create or update view descriptors, you should specify selection criteria (where possible) to subset the number of rows that the DBMS returns to the SAS System.

As a general rule, a view descriptor's WHERE clause is passed to the DBMS for processing. Table 9.3 on page 127 and the explanation afterwards describe this process in more detail.

Table 9.3 How WHERE Clauses Are Processed with View Descriptors

View descriptor with a WHERE clause	System that processes the WHERE clause
used in a DATA step, PROC step, or PROC SQL query	
without additional WHERE clause	DBMS
with additional WHERE clause	DBMS: compound WHERE clause built with AND; otherwise, SAS System[1]
used with a PROC SQL Pass-Through query[2]	
without additional WHERE clause	DBMS
with additional WHERE clause	SAS System. Each DBMS table in the join is processed by the DBMS and then returned to the SAS System for final processing. PROC SQL cannot optimize a join of this kind.

1 The interface view engine builds a compound WHERE clause using AND operator(s) if the clauses are valid for the DBMS. Otherwise, only the valid part of the WHERE clause is sent to the DBMS for processing and the SAS System processes the remaining part(s). See "Using Multiple WHERE Clauses" below for more information.

2 When a view descriptor and a Pass-Through query are used within the same PROC SQL query, they are usually joined in the FROM clause.

Note: See Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software," on page 65 for more information on Pass-Through queries. △

Using Multiple WHERE Clauses

When a view descriptor that includes a DBMS-specific SQL WHERE clause is used in a DATA step or procedure that includes a SAS WHERE statement, the SAS/ACCESS

interface view engine tries to pass both WHERE conditions to the DBMS for processing. If all or part of the WHERE clause is valid for the DBMS, SAS/ACCESS software uses one or more logical AND operators to build a compound WHERE clause.

However, if a WHERE clause contains SAS enhancements or features that are not supported by the DBMS, the WHERE conditions are split up, and only the valid condition (that is, valid DBMS-specific syntax) is sent to the DBMS. The SAS System processes the remaining part(s) of the WHERE clause.

For example, a DBMS-specific SQL cannot parse a colon modifier on a comparison operator, and therefore, the first half of the following WHERE clause cannot be passed to the DBMS.

The second half of the WHERE clause is passed to the DBMS, and the DBMS returns to the SAS System all the rows in the INVOICE table for which PAIDON is on or after 01JAN94. The SAS System must then process the first half of the WHERE clause to subset the rows that were returned from the DBMS for the countries Argentina and Australia:

```
where country =: 'A' and paidon >= '01JAN94'd;
```

It is more efficient to use a LIKE operator that is valid in both the DBMS-specific SQL and the SAS System:

```
where country like 'A%' and paidon >= '01JAN94'd;
```

In this case, the DBMS can process both halves of the WHERE clause.

Note that the SAS/ACCESS interface view engine can translate certain SAS conventions, such as datetime formats or the IS MISSING operator, to their DBMS-specific SQL equivalents and pass these clauses to the DBMS.

In most cases it is more efficient for the DBMS to process the WHERE conditions. Therefore, write WHERE conditions using features (such as SQL operators) that are valid in both the SAS System and the DBMS so that the DBMS can process the entire WHERE clause.

It is also more efficient to use a SAS WHERE statement instead of a subsetting IF statement. As just described, a WHERE clause is passed to the DBMS for processing and returns a subset of rows to the SAS System for further processing. In contrast, when you use a subsetting IF statement, every row is returned to the SAS System to be evaluated by the IF statement. Therefore, using a SAS WHERE statement often improves performance.

Writing Efficient WHERE Clauses for View Descriptors

You should write WHERE clauses that enable the DBMS to use its indexes, where possible. This is a good practice whether you specify the WHERE clause in the view descriptor's selection criteria or you use a SAS WHERE statement in a DATA step that references a view descriptor. This practice is especially important when you are accessing large DBMS tables.

You cannot tell the DBMS to use an index, but you can write WHERE clauses that enable it to use its DBMS indexes effectively. Here are some guidelines for writing WHERE clauses that enable the DBMS to use indexes effectively.

Note: The guidelines for each specific DBMS may vary. △

- Avoid the NOT operator when you can use an equivalent form:

  Inefficient: **`where zipcode not>8000`**

  Efficient: **`where zipcode<=8000`**

- Avoid the >= and <= operators when you can use the BETWEEN predicate.

  Inefficient: **`where ZIPCODE>=70000 and ZIPCODE<=80000`**

Efficient: **`where ZIPCODE between 70000 and 80000`**

- Avoid using LIKE predicates that begin with % or _ .

  Inefficient: **`where COUNTRY like '%INA'`**

  Efficient: **`where COUNTRY like 'A%INA'`**

- Avoid arithmetic expressions in a predicate.

  Inefficient: **`where SALARY>12*4000.00`**

  Efficient: **`where SALARY>48000.00`**

For a list of operators that are generally accepted by the SQLs of most DBMSs, see the *SAS Procedures Guide*.

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