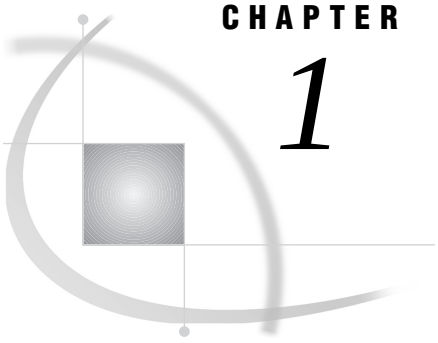

CHAPTER

1

Teradata Chapter, First Edition

Introduction

This chapter introduces SAS System users to the Teradata Database System (DBMS). It accompanies and should be used with *SAS/ACCESS Software for Relational Databases: Reference* (order #57204).*

This chapter focuses on the terms and concepts that help you use the SAS/ACCESS Interface to Teradata. It describes the statements and options that are specific to Teradata and the SAS SQL procedure's CONNECT statement.

For more information about a Teradata concept or term, refer to your Teradata documentation.

The SAS/ACCESS Teradata Client

Historically, NCR supplied customers a comprehensive, high-end package: proprietary hardware and an operating system to support the Teradata DBMS. For example, porting Teradata to UNIX, NCR enhanced a UNIX version for Teradata and supplied NCR hardware to support the operating system, providing dedicated connections between the symmetric multiprocessing (SMP) machines.

Because Teradata customers run many processors at the same time for queries of the database, users enjoy excellent DBMS *server* performance. Such high-level performance challenges client software: a single pipeline, the network connection, must deliver data from a sophisticated, parallel processing *server* to the *client* machine.

SAS/ACCESS for Teradata is client software that is optimized to read and process Teradata data. The emphasis of the software is read performance of Teradata rows to support the *client's* use of SAS as an analysis tool for information that the *server* stores in Teradata DBMS tables.

For this reason, the Chapter provides information throughout on how to optimize DBMS read operations. SAS/ACCESS also supports creating and updating Teradata

tables. However, SAS/ACCESS performance for loads of very large tables does not yet match the performance of FASTLOAD, a native Teradata tool.

Version 8 Information for SAS Users

SAS/ACCESS for Teradata does not support SAS Version 6 DBLOAD and ACCESS procedures. The LIBNAME engine technology available in Version 8 enhances and replaces the functionality of these procedures. Consequently, you must revise SAS jobs that include DBLOAD or ACCESS procedures, which were written for a different SAS/ACCESS database Interface, to run them with SAS/ACCESS for Teradata.

Processing Tips For SAS Users

Reading and Inserting to the Same Teradata Table

If you use SAS/ACCESS to read rows from a Teradata table and then attempt to insert these rows into the same table, you will hang (suspend) your SAS session.

Behind the scenes

- a SAS/ACCESS connection requests a standard Teradata READ lock for the read operation.
- a SAS/ACCESS connection then requests a standard Teradata WRITE lock for the insert operation.
- the WRITE lock request suspends because the read connection already holds a READ lock on the table. Consequently, your SAS session hangs (is suspended).

Example: SAS Code That Hangs a Session

```
libname tra teradata user=kamdar password=ellis;
proc sql;
insert into tra.sametable
   select * from tra.sametable;
```

In this example

- SAS/ACCESS creates a read connection to Teradata to fetch the rows selected (select *) from TRA.SAMETABLE, requiring a standard Teradata READ lock; Teradata issues a READ lock.
- SAS/ACCESS creates an insert connection to Teradata to insert the rows into TRA.SAMETABLE, requiring a standard Teradata WRITE lock. But, the WRITE lock request suspends—the table is locked already by the READ lock.
- Your SAS/ACCESS session hangs.

Obviously, to avoid the situation described, do not submit this code. There is an alternative. You can add SAS/ACCESS locking options. These options *modify Teradata's standard locking.* For a usage example, see "Example 3: Preventing a Hung SAS Session When Reading and Inserting to the Same Table" on page 16.

Using a BY Clause to Order Query Results

SAS/ACCESS returns table results from a query in random order because Teradata returns the rows to SAS/ACCESS randomly. In contrast, traditional SAS processing returns SAS data set observations in the same order every run of your job. If maintaining row order is important, then you should add a BY clause to your SAS statements. A BY clause ensures consistent ordering of the table results from Teradata.

In the following example, a Teradata table, ORD, has columns NAME and NUMBER. The PROC PRINT examples illustrate consistent and inconsistent ordering in the display of the ORD table rows.

```
libname prt teradata user=kamdar password=ellis;
```

PROC PRINT Example 1: Inconsistent Ordering

```
proc print data=prt.ORD; var name number; run;
```

If this statement is run several times, the ORD rows are likely to be arranged differently each time. Because SAS/ACCESS displays the rows in the order that Teradata returns them, that is, randomly.

PROC PRINT Example 2: More Consistent Ordering

```
proc print data=prt.ORD; var name number; by name; run;
```

With this statement, PROC PRINT output is ordered according to the NAME value. However, on successive runs of the statement, display of rows with a different number and an identical name can vary.

Output 1.1 PROC PRINT Display 1

```
Rita Calvin 2222
Rita Calvin 199
```

Output 1.2 PROC PRINT Display 2

```
Rita Calvin 199
Rita Calvin 2222
```

PROC PRINT Example 3: Consistent Ordering

```
proc print data=prt.ORD; var name number; by name number; run;
```

With this statement the ordering is always identical because every column is specified in the BY clause. Thus, your PROC PRINT output always looks the same.

Replacing PROC SORT with a BY Clause

In general, PROC SORT steps are not useful to output a Teradata table. In traditional SAS processing, PROC SORT is used to order observations in a SAS data set. Subsequent SAS steps that use the sorted data set receive and process the observations in the sorted order. Teradata will not store output rows in the sorted order. Consequently, do not sort rows with PROC SORT if the destination sorted file is a Teradata table.

The examples that follow modify traditional SAS code for SAS/ACCESS for Teradata.

Example 1: A Traditional PROC SORT Job

```
libname sortprt '.';
proc sort data=sortprt.salaries; by income;
proc print data=sortprt.salaries;
```

Example 1 illustrates a PROC SORT statement found in typical SAS processing. This statement is useless in SAS/ACCESS for Teradata.

Example 2: A Sort Job Modified for SAS/ACCESS for Teradata

```
libname sortprt teradata user=kamdar password=ellis;
proc print data=sortprt.salaries; by income;
```

Example 2 removes the PROC SORT statement shown in Example 1. Instead, it uses a BY clause with PROC PRINT. The BY clause returns Teradata rows ordered by the INCOME column.

SAS/ACCESS LIBNAME Statement

Chapter 3, "SAS/ACCESS LIBNAME Statement" describes options that you specify in the LIBNAME statement to associate a SAS libref with a DBMS server, database, and a group of tables and views. The following section describes DBMS-specific options and option values that you can use in the SAS/ACCESS Interface to Teradata.

LIBNAME Statement: Teradata Specifics

Associates your SAS libref with a server and database containing a group of DBMS tables and views.

Valid: Anywhere

Syntax

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

Arguments

libref
is any SAS name that serves as an alias to associate the SAS System with a Teradata server and database containing a group of DBMS tables and views.

SAS/ACCESS-engine-name
is the SAS/ACCESS engine name for the Teradata DBMS. In this case, the name **teradata**. Alternatively, you can use **sasiotra**. The engine name is required. Note that SAS/ACCESS implements engines differently in different operating environments.

SAS/ACCESS-engine-connection-options
are options that you specify to connect to the Teradata DBMS. If the connection options that you specify contain characters that are not valid in a SAS name, enclose the values in quotation marks. When you specify the appropriate system options or environment variables for the Teradata DBMS, you often can omit the SAS/ACCESS engine connection options.

SAS/ACCESS-LIBNAME-options
are options that you apply to the processing of objects and data in the DBMS. In this case, Teradata tables, views, or indexes. For example, the READ_LOCK_TYPE= option specifies whether SAS/ACCESS should lock at the table or view level when processing Teradata objects.

Some SAS/ACCESS LIBNAME options have the same names as SAS/ACCESS data set options. When you specify an option in the LIBNAME statement, the option applies to all objects (Teradata tables and views) that are referenced by the libref. In contrast, a SAS/ACCESS data set option applies only to the specified data set. If you specify a like-named option in both the SAS/ACCESS engine LIBNAME statement and after a SAS data set name (referencing a DBMS table or view), SAS/ACCESS uses the value of the data set option. See “Data Set Options: Teradata Specifics” on page 16 for more information.

Details The LIBNAME statement associates a libref with the SAS/ACCESS engine for Teradata to access DBMS tables or views. You specify a particular Teradata table or view name using a two-level SAS name. An example of a two-level SAS name is MYDBLIB.EMPLOYEES_Q2. In this example, MYDBLIB is the SAS libref that points to a particular group of Teradata DBMS objects. EMPLOYEES_Q2 is the name of a specific Teradata table.

When you specify MYDBLIB.EMPLOYEES_Q2 in a DATA step or procedure, you dynamically access the DBMS table. Version 8 and above of the SAS System support reading, updating, creating, and dropping (deleting) Teradata tables.

To disassociate or clear a libref from the DBMS, use a LIBNAME statement, specifying the libref (for example, MYDBLIB) and the CLEAR option as shown in the example that follows.

Example: Clearing a Libref

```
libname mydblib CLEAR;
```

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

SAS/ACCESS Engine Connection Options The following SAS/ACCESS engine connection options are *required* for Teradata and must be used together:

- USER= on page 7
- PASSWORD= on page 7.

The following SAS/ACCESS engine connection options are *optional* for Teradata:

- ACCOUNT= on page 7
- TDPID= on page 7.

USER='*Teradata-user-name*'
specifies a Teradata user name. If the name contains blanks or national characters, enclose the name in quotation marks.

Alias: USERNAME=

PASSWORD='*Teradata-password*'
specifies a Teradata password. The password that you specify must be correct for your USER= value.

Aliases: PASS= and PW=

ACCOUNT=<'>*account_ID*<'>
specifies the account number that you want to charge for the Teradata session.

TDPID=<'> *dbcname*<'>

The following information applies to NETWORK-ATTACHED systems (PC and UNIX).

dbcname specifies an entry in your (client) HOSTS file that provides an IP address for a database server connection.

By default, SAS/ACCESS connects to the Teradata server that corresponds to the *dbccop1* entry in your HOSTS file. When you run only one Teradata server, and your HOSTS file defines the *dbccop1* entry correctly, you do not need to specify TDPID=.

However, if you run more than one Teradata server, you can use the TDPID= option to override the default by specifying a dbcname, eight characters or less. SAS/ACCESS adds the specified dbcname to the login string that it submits to Teradata. (Note: Teradata documentation refers to this name as the tdpid component of the login string.)

After SAS/ACCESS submits a dbcname to Teradata, Teradata searches your HOSTS file for all entries that begin with the same dbcname. In order for Teradata to recognize the HOSTS file entry, the dbcname suffix must be COPx (x is a number). If there is only one entry that matches the dbcname, x must be 1. If there are multiple entries for the dbcname, x must begin with 1 and increment sequentially for each related entry. (See the example HOSTS file entries on page 8).

When there are multiple, matching entries for a dbcname in your HOSTS file, Teradata does simple load balancing by selecting one of the Teradata servers

specified for login. Teradata distributes your queries across these servers so that it can return your results as fast as possible.

The TDPID= examples below assume that your HOSTS file contains the following dbcname entries and IP addresses:

dbccop1 10.25.20.34

myservercop1 130.96.8.207

xyzcop1 33.44.55.66

xyzcop2 11.22.33.44

Example 1: TDPID= is not specified.

In example 1, the TDPID= option is not specified, establishing a login to the Teradata server that runs at 10.25.20.34.

Example 2: TDPID= myserver

In example 2, you specify a login to the Teradata server that runs at 130.96.8.207.

Example 3: TDPID=xyz

In example 3, you specify a login to a Teradata server that runs at 11.22.33.44 or to a Teradata server that runs at 33.44.55.66.

The following information applies to CHANNEL-ATTACHED systems (MVS).

TDPID= specifies the subsystem name, which must be TDPx, where x can be 0-9, A-Z (not case- sensitive), or $, # or @. If there is only one Teradata server, and your MVS System Administrator has set up the HSISPB and HSHSPB modules, you do not need to specify TDPID=. For further information, see your Teradata TDPID documentation for MVS.

Alias: SERVER=

SAS/ACCESS LIBNAME Options When you specify options in the LIBNAME statement, the option applies to all tables and views that the libref represents. If you do not specify an option, the default value is in effect and applies to the same objects.

The SAS/ACCESS interface to Teradata supports all of the SAS/ACCESS LIBNAME options listed in Chapter 3, "SAS/ACCESS LIBNAME Statement" , except for DBMAX_TEXT=. In addition to the supported options, the following LIBNAME options are used only in the interface to Teradata or have Teradata–specific aspects to them:

- DBINDEX= on page 9
- DBPROMPT= on page 9
- PREFETCH= on page 9
- READ_ISOLATION_LEVEL= on page 12
- READ_LOCK_TYPE= on page 12
- READ_MODE_WAIT= on page 12
- DATABASE= on page 9
- SPOOL= on page 10
- UPDATE _ISOLATION_LEVEL= on page 13
- UPDATE_LOCK_TYPE= on page 13
- UPDATE_MODE_WAIT= on page 14.

This section groups the SAS/ACCESS LIBNAME options by function.

- “LIBNAME Miscellaneous Options” on page 9

LIBNAME Miscellaneous Options

DBPROMPT=YES | NO
specifies whether SAS displays a window that prompts for SAS/ACCESS engine connection options, instead of having you specify them on the LIBNAME statement.

The SAS/ACCESS Interface to Teradata allows an entry up to 30 characters for the USERNAME and PASSWORD.

Default: DBPROMPT=NO

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement".

DATABASE=<'> *alternate database*<'>
specifies an alternative database.

By default, a libref points to the database that is named the same as your user name. You can use this option to point to a different database. The DATABASE= option enables you to view or modify a different user's DBMS tables or views, assuming that you have the requisite Teradata privileges to that user's tables and views. For example, to read a different user's tables, you must have the Teradata privilege SELECT for that user's tables.

Default: The database that is named the same as your user name.

Aliases: SCHEMA= or DB=

In the example that follows, user KAMDAR prints the EMP table which is located in the OTHERUSER database.

```
libname mydblib teradata user=kamdar pw=ellis schema=otheruser;
proc print data=mydblib.emp;
run;
```

Note: For more information about changing the default database, see the DATABASE statement in your Teradata documentation. △

LIBNAME Performance Options

DBINDEX=YES|NO
specifies whether SAS should use table indexes for certain types of processing.

Use DBINDEX=YES to improve performance of specific processing. For example, a DATA step that contains the KEY= option. Or, when using a libref with PROC SQL to join a Teradata table and a SAS data set.

Default: DBINDEX=NO

For a more complete description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement"

PREFETCH='*unique_storename*, [*#sessions*,*algorithm*]'
enables the PreFetch facility on tables that are accessed by the libref defined with the LIBNAME statement. Before using PreFetch, see “About the PreFetch Facility” on page 33 for a complete discussion, including when and how the option enhances read performance of a job run more than once. For usage examples, see

"Using PreFetch as a LIBNAME Option" on page 35 and "Using Prefetch as a Global Option" on page 36.

The PreFetch Arguments include:

unique_storename a unique name that you specify. This value names the Teradata macro that PreFetch creates to store selected SQL statements in the first run of a job. During subsequent runs of the job,SAS/ACCESS presubmits the stored SQL statements in parallel to the Teradata DBMS.

#sessions controls the number of statements that PreFetch submits in parallel to Teradata. A valid value is 1 through 9. If you do not specify a *#sessions* value, the default is 3.

algorithm specifies the algorithm that PreFetch uses to order the selected SQL statements. Currently, the only valid value is SEQUENTIAL.

Default: Prefetch is not enabled.

SPOOL=YES | NO

specifies whether SAS/ACCESS should create a utility spool file when read processing requires that data be read more than once.

Some SAS procedures, for example PROC TRANSREG and PROC DISCRIM, require two passes to complete analysis of the data. SPOOL=YES specifies for SAS on the first pass to spool all rows to a file. Then, on the second pass, for SAS to read the row data from the SAS spool file.

SPOOL=NO requires SAS/ACCESS to issue the identical SELECT statement to Teradata twice. As a result, Teradata must do unnecessary, extra work. Additionally, because the Teradata table can be modified between passes, SPOOL=NO can cause data integrity problems. For both reasons use SPOOL=NO with discretion.

Default: SPOOL=YES

For a complete description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement"

SAS/ACCESS Lock Options for Teradata

Note: This section discusses the LIBNAME SAS/ACCESS lock options for Teradata. When reading, do not confuse the Teradata keyword value, ACCESS, and the SAS/ACCESS Interface to the Teradata DBMS. △

Before using the SAS/ACCESS lock options for Teradata, it is important to understand that these options modify Teradata's standard locking for row, tables, and views. Teradata usually locks at the row level; SAS/ACCESS lock options lock at the table or view level. For details about the scope of SAS/ACCESS lock options, read the next topic, Understanding the Scope of SAS/ACCESS Lock Options. For a complete description of Teradata locking, see the LOCKING statement in your Teradata SQL Reference manual.

In general, Teradata's row locks are preferable. Although SAS/ACCESS lock options do not support row-level locking, they can be appropriate for special situations, see "Identifying Situations to Use SAS/ACCESS Lock Options" on page 14. If SAS/ACCESS lock options do not meet your specialized needs, Teradata provides additional locking features using views. See CREATE VIEW in your Teradata SQL Reference manual for details.

Understanding the Scope of SAS/ACCESS Lock Options

Table 1.1 SAS/ACCESS Lock Options Versus Standard Teradata Locking

SAS /ACCESS Lock Options	lock the ...	modifying standard Teradata locking which locks the ...
READ_ISOLATION_LEVEL= READ_LOCK_TYPE= READ_MODE_WAIT=	TABLE [1]	ROW
UPDATE_ISOLATION_LEVEL= UPDATE_LOCK_TYPE= UPDATE_MODE_WAIT=	TABLE [2]	ROW

1 Specifying READ_LOCK_TYPE=VIEW locks the view.
2 Specifying UPDATE_LOCK_TYPE=VIEW locks the view.

Changing the scope of the lock from row-level to the entire table does not affect Teradata ACCESS (value) locks. However, the level change does affect READ or WRITE locks. An important consequence of a READ or WRITE table lock: you increase the time that other users must wait to access the table.

Use of SAS/ACCESS lock options can decrease overall system performance. Apply READ or WRITE locks *only* when you must apply special locking on Teradata tables. Further, when using SAS/ACCESS lock options, limit the span (see next topic) of the locks as much as possible.

Limiting the Span (Effect) of SAS/ACCESS Lock Options LIBNAME read or update lock options affect all the tables referenced by your libref that you open. In contrast, a like-named data set option applies only to the table specified. Since the span of LIBNAME SAS/ACCESS locks can be broader than you intend, limit the effect of SAS/ACCESS locks by using only the comparable data set options whenever possible.

Using SAS/ACCESS to Generate Teradata Locking Modifiers To use SAS/ACCESS locking options you must specify a set of three "SAS/ACCESS LIBNAME Read Lock Options" on page 12 or "SAS/ACCESS LIBNAME Update Lock Options" on page 13. Which set you specify depends on the type of processing you are doing. But, if you specify an incomplete set, SAS/ACCESS returns an error message.

After you correctly specify a set of SAS/ACCESS lock options, SAS/ACCESS generates locking modifiers on your behalf to Teradata. (SAS/ACCESS lock options merely cause the LIBNAME engine to transmit a locking request to the DBMS; the Teradata DBMS performs all the data locking.)

For example, assume that you specify all three READ lock options. SAS/ACCESS generates locking modifiers that contain values for each of the READ options. Similarly, if you specify all three UPDATE lock options, SAS/ACCESS generates locking modifiers that contain values for each UPDATE option.

Note: As mentioned earlier, if you do not use SAS/ACCESS locking options, SAS/ACCESS will not generate locking modifiers. Without locking modifiers, Teradata's lock defaults are in effect. △

SAS/ACCESS Read Lock Options

Table 1.2 SAS/ACCESS Locking Options for Read Operations

SAS/ACCESS Read Lock Option	Values
READ_ISOLATION_LEVEL=	ACCESS|READ|WRITE
READ_LOCK_TYPE=	TABLE|VIEW
READ_MODE_WAIT=	YES|NO

SAS/ACCESS LIBNAME Read Lock Options

- Submit locking modifiers to Teradata which contain the values that you specify for the three read lock options.
- Span the scope of the LIBNAME, locking every table that you open.
- Override Teradata's standard row-level locking.

READ_ISOLATION_LEVEL=ACCESS|READ|WRITE
specifies the level of isolation from other table users that is required when reading Teradata tables.

ACCESS — obtains an ACCESS lock even when a WRITE lock is in effect by another user. This lock can cause inconsistent or unusual results during a read operation.

Effect: permits other users any type of lock.

READ — obtains a READ lock.

Effect: permits other users to READ lock but not modify the table. Typically, READ is adequate for most SAS/ACCESS processing.

WRITE — obtains a WRITE lock. (You cannot explicitly release a WRITE lock. It is released when the DBMS object on which it is applied is closed.)

Effect: Excludes other users from requesting a READ or WRITE lock on the table. For this reason, a WRITE lock is unnecessarily restrictive.

READ_LOCK_TYPE=TABLE|VIEW
specifies and changes the scope of the Teradata lock during SAS/ACCESS read operations.

TABLE — locks the entire Teradata table.

VIEW — locks the entire Teradata view.

If you do not use SAS/ACCESS read lock options, Teradata applies a lock either at the row or table level. Since SAS/ACCESS does not support a ROW lock, use SAS/ACCESS lock options only when you must lock the entire table or view.

READ_MODE_WAIT=YES|NO
specifies during SAS/ACCESS read operations whether Teradata should wait to acquire a lock or fail the request when the DBMS resource is already locked by a different user.

YES — specifies for Teradata to wait to acquire the lock.

If you specify READ_MODE_WAIT=YES, SAS/ACCESS waits indefinitely until it can acquire the lock.

NO specifies for Teradata to fail the lock request if the specified DBMS resource is locked.

If you specify READ_MODE_WAIT=NO, and a different user holds a restrictive on page 21 lock, then the executing SAS step will fail. SAS/ACCESS continues processing the job by executing the next step.

SAS/ACCESS Update Lock Options

Table 1.3 Options for Update Operations

SAS/ACCESS Update Lock Option	Values
UPDATE_ISOLATION_LEVEL=	ACCESS|READ|WRITE
UPDATE_LOCK_TYPE=	TABLE|VIEW
UPDATE_MODE_WAIT=	YES|NO

SAS/ACCESS LIBNAME Update Lock Options

- Submit locking modifiers to Teradata which contain specified values for the three update lock options.
- Span the scope of the LIBNAME, locking every table that you open.
- Override Teradata standard row-level locking.

UPDATE_ISOLATION_LEVEL=ACCESS/READ/WRITE
specifies the level of isolation from other table users that is required when SAS/ACCESS reads Teradata rows in preparation for an update.

ACCESS obtains an ACCESS lock during update preparation.

Effect: Permits other users to READ or WRITE lock the table. An ACCESS lock avoids a potential deadlock but can cause data corruption if another user is updating the same data.

READ obtains a READ lock during update preparation.

Effect: Permits other users to READ lock the table.

Note: If two users specify UPDATE_ISOLATION_LEVEL=READ, and attempt to update the same DBMS object, there is a high probability of a deadlock because the lock is held at the table or view level. △

WRITE obtains a WRITE lock during update preparation.

Effect: Excludes all other users, except those who specify ACCESS locks, from reading the table. A WRITE lock eliminates a potential deadlock and ensures data integrity.

Note: Important: Since WRITE locks exclude all users, except those who specify ACCESS locks, see "Limiting the Span (Effect) of SAS/ACCESS Lock Options" on page 11 for how to scope your lock appropriately. △

UPDATE_LOCK_TYPE=TABLE|VIEW
specifies and changes the scope of the Teradata lock during SAS/ACCESS update operations.

TABLE locks the entire Teradata DBMS table.

VIEW locks the Teradata DBMS view.

If you do not use SAS/ACCESS update lock options, Teradata applies a lock either at the row or table level. Since SAS/ACCESS does not support a ROW level lock, use SAS/ACCESS lock options only when you must lock the entire table or view.

UPDATE_MODE_WAIT=YES | NO
specifies during SAS/ACCESS update operations whether Teradata should wait to acquire a lock or fail the request when the DBMS resource is locked by a different user.

YES specifies for Teradata to wait to acquire the lock.

If you specify UPDATE_MODE_WAIT=YES, SAS/ACCESS waits indefinitely until it can acquire the lock.

NO specifies for Teradata to fail the lock request if the specified DBMS resource is locked.

If you specify UPDATE_MODE_WAIT=NO, and a different user holds a restrictive on page 21 lock, then the executing SAS step will fail. SAS/ACCESS continues processing the job by executing the next step.

Identifying Situations to Use SAS/ACCESS Lock Options This section describes situations that might require SAS/ACCESS lock options instead of the standard locking provided by Teradata. As mentioned earlier, the options change the scope of the lock from row to table and thereby affect concurrent access to DBMS objects.

Situation 1: Reducing the Isolation Level for a Read Operation

When you READ lock a table, you can lock out both yourself and other users from updating or inserting into the table. Conversely, when other users update or insert into the table, they can lock you out from reading the table. In situation 1, you want to reduce the isolation level during a read operation. To do this, you specify the following read SAS/ACCESS lock options and values:

READ_ISOLATION_LEVEL=ACCESS

READ_LOCK_TYPE_TABLE

READ_MODE_WAIT=YES.

The effect of the options and settings in Situation 1

- Specifies ACCESS locking, eliminating a lock out of yourself and other users. Since ACCESS can return inconsistent results to a table reader, specify ACCESS only if you are casually surveying data, not if you require precise data.
- Changes the scope of the lock from row-level to the entire table.
- Requests that Teradata wait if it attempts to secure your lock and finds the resource already locked.

Situation 2: Avoiding Contention

When you read or update a table, contention can occur: the DBMS is waiting for other users to release their locks on the table that you want to access. This contention

suspends your SAS/ACCESS session. In situation 2, to avoid contention during a read operation, you specify the following SAS/ACCESS read lock options and values:

READ_ISOLATION_LEVEL=READ

READ_LOCK_TYPE=TABLE

READ_MODE_WAIT=NO.

The effect of the options and settings in Situation 2

- Specifies a READ lock.
- Changes the scope of the lock. Because SAS/ACCESS does not support row locking when you obtain the lock requested, you lock the entire table until your read operation finishes.
- Tells SAS/ACCESS to fail the job step if Teradata cannot immediately obtain the READ lock.

Usage: SAS Code Examples for SAS/ACCESS Lock Options

Example 1: Setting the Isolation Level to ACCESS for Teradata Tables

```
   /*Generates a quick survey of unusual customer purchases.*/
libname cust teradata user=kamdar password=ellis
              READ_ISOLATION_LEVEL=ACCESS
              READ_LOCK_TYPE=TABLE
              READ_MODE_WAIT=YES;
proc print data=cust.purchases(where= (bill<2));
run;
data local;
 set cust.purchases (where= (quantity>1000));
run;
```

In Example 1, SAS/ACCESS

- Connects to the Teradata DBMS, specifying the three SAS/ACCESS LIBNAME read lock options.
- Opens the PURCHASES table, obtaining an ACCESS lock if a different user does not hold an EXCLUSIVE lock on the table.
- Reads and displays table rows with a value less than two in the BILL column.
- Closes the PURCHASES table, releasing the ACCESS lock.
- Opens the PURCHASES table again, obtaining an ACCESS lock if a different user does not hold an EXCLUSIVE lock on the table.
- Reads table rows with a value greater than 1000 in the QUANTITY column.
- Closes the PURCHASES table, releasing the ACCESS lock.

Example 2: Setting Isolation Level to WRITE to Update a Teradata Table

```
   /*Updates the critical Rebate row.*/
libname cust teradata user=kamdar password=ellis;
proc sql;
 update cust.purchases(UPDATE_ISOLATION_LEVEL=WRITE
                       UPDATE_MODE_WAIT=YES
                       UPDATE_LOCK_TYPE=TABLE)
 set rebate=10 where bill>100;
```

```
quit;
```

In Example 2, SAS/ACCESS

- Connects to the Teradata DBMS, specifying the three SAS/ACCESS data set update lock options.
- Opens the PURCHASES table, obtaining a WRITE lock if a different user does not hold a READ, WRITE or EXCLUSIVE lock on the table.
- Updates table rows with BILL greater than 100, setting the REBATE column to 10.
- Closes the PURCHASES table, releasing the WRITE lock.

Example 3: Preventing a Hung SAS Session When Reading and Inserting to the Same Table

```
  /* The SAS/ACCESS lock options prevent the session hang */
  /* that occurs when I read and insert into sametable    */
libname tra teradata user=kamdar password=ellis
                     connection=unique;
proc sql;
insert into tra.sametable
   select * from tra.sametable(read_isolation_level=access
                               read_mode_wait=yes
                               read_lock_type=table);
```

In Example 3, SAS/ACCESS

- creates a read connection to fetch the rows selected (select *) from TRA.SAMETABLE, specifying an ACCESS lock (READ_ISOLATION_LEVEL=ACCESS). Teradata grants the ACCESS lock.
- creates an insert connection to Teradata to process the insert operation to TRA.SAMETABLE. Because the ACCESS lock that is already on the table permits access to the table, Teradata grants a WRITE lock.
- performs the insert operation without hanging (suspending) your SAS session on page 3.

Example: Specifying a LIBNAME Statement to Access Teradata Data

In this example, the libref MYDBLIB uses the SAS/ACCESS Interface to Teradata to connect to a Teradata database. The SAS/ACCESS engine connection options are USER= and PASSWORD=.

```
libname mydblib teradata user=kamdar
   password=ellis;

proc print data=mydblib.employees;
   where dept='CSR010';
run;
```

Data Set Options: Teradata Specifics

Chapter 4, "SAS/ACCESS Data Set Options" describes the SAS/ACCESS options that you can use when you specify a SAS data set in a DATA or PROC step; in this case, the SAS data set accesses data from a Teradata DBMS table or view. The following section describes the DBMS-specific options and option values that you can use in the SAS/ACCESS Interface to Teradata.

A data set option applies only to the specified table or DBMS object. Unless otherwise noted, when you omit a SAS/ACCESS data set option, and specify a like-named LIBNAME option in the LIBNAME statement, the value specified for the LIBNAME option applies to all tables in the libref.

Note: Not all LIBNAME options have corresponding data set options. Refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" , Chapter 4, "SAS/ACCESS Data Set Options" , and this chapter for a full listing of SAS/ACCESS LIBNAME options, data set options, as well as Teradata-specific LIBNAME and data set options. △

The SAS/ACCESS interface to Teradata supports all of the SAS/ACCESS data set options listed in Chapter 4, "SAS/ACCESS Data Set Options" , except for DBMAX_TEXT=. In addition to the supported options, the following data set options are used only in the interface to Teradata or have Teradata–specific aspects to them:

DBINDEX=

Specifies whether SAS should use table indexes for certain types of processing.

Default value: NO

Syntax

DBINDEX=YES | NO

YES
use available table indexes.

NO
do not use available table indexes.

Details DBINDEX=YES can improve performance in certain processing situations, for example, a DATA step with the KEY= option. Or, when using a libref with PROC SQL to join a Teradata table and a SAS data set.

DBNULL=

Indicates whether NULL is a valid value for the specified column(s).

Default value: YES

Syntax

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

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

YES
indicates that a NULL value is valid for the specified column(s) in the Teradata table.

NO
indicates that a NULL value is not valid for the specified column(s) in the Teradata table.

ALL
indicates that the YES or NO value applies to all columns in the Teradata table.

Details When you use SAS/ACCESS to create tables, specifying YES tells Teradata to allow a null value for the column specified. In contrast, specifying NO causes SAS/ACCESS to append 'NOT NULL' to the column definition of the CREATE TABLE statement that it submits to Teradata. Consequently, Teradata will not accept a null value as valid for the given column.

For more information on the DBNULL= option, see Chapter 4, "SAS/ACCESS Data Set Options" .

DBTYPE=

Specifies the data type(s) to override default data type(s) when SAS creates a Teradata table.

Default value: See Table 1.5 on page 31 for a list of the default data types.

Syntax

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

column name
specifies a SAS variable name to generate a corresponding Teradata column.

DBMS-type
specifies a Teradata data type and attributes.

Details

This option is valid only when creating Teradata tables.

When SAS /ACCESS outputs a new table, it converts each SAS data type for a column to a default Teradata data type. For a list of the default data types, see Table 1.5 on page 31. However, you can use DBTYPE= to override the default if you need a different Teradata data type.

You can also use DBTYPE= to specify data attributes for a column. See your Teradata CREATE TABLE documentation for information on which data type attributes that you can specify.

If you specify DBNULL=NO for a column, do not also use NOT NULL with DBTYPE= to specify this attribute for that column. If you do, 'NOT NULL' is inserted twice in the column definition. This will cause Teradata to generate an error message.

For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options" .

Example 1: Specifying a Data Type

```
data mydblib.newdept(dbtype=(deptno='byteint' city='char(25)'));
set dept;
run;
```

Example 1 creates a new Teradata table, NEWDEPT, specifying the Teradata data types for the DEPTNO and CITY columns.

Example 2: Specifying a Data Type and Attributes

```
data mydblib.newemployees(dbtype= (empno="SMALLINT FORMAT '9(5)'
  CHECK (empno >= 100 AND empno <= 2000)"));
set employees;
run;
```

Example 2 creates a new Teradata table, NEWEMPLOYEES, and specifies a data type and attributes for the EMPNO column.

Note: Example 2 encloses the Teradata type and attribute information in double quotes. Single quotes conflict with the single quotes required by the Teradata FORMAT attribute. If you use single quotes, SAS returns syntax error messages. △

READ_ISOLATION_LEVEL=

Specifies the level of isolation from other table users that is required during SAS/ACCESS read operations.

Default value: None

Syntax

READ_ISOLATION_LEVEL=ACCESS | READ | WRITE

ACCESS

obtains an ACCESS lock by ignoring other users' ACCESS, READ, and WRITE locks.

Effect: Permits other users to obtain a lock on the table or view.

READ
obtains a READ lock if no other user holds a WRITE or EXCLUSIVE lock.
Effect: Prevents other users from being granted a WRITE or EXCLUSIVE lock. It does not prevent other users from reading the object.

WRITE
obtains a WRITE lock on the table or view if no other user has a READ, WRITE, or EXCLUSIVE lock on the resource. You cannot explicitly release a WRITE lock. It is released only when the table is closed.
Effect: Prevents other users from acquiring any lock but ACCESS.

Details Specifying ACCESS can return inconsistent or unusual results. And, specifying WRITE is unnecessarily restrictive, locking the entire table until the read operation is finished. You cannot explicitly release a WRITE lock. It is only released when the DBMS object on which it is applied is closed. Therefore, whenever possible specify READ; it is usually adequate for most SAS/ACCESS processing.

For more information, see the LOCKING modifier in your Teradata SQL Reference manual, as well as "Understanding the Scope of SAS/ACCESS Lock Options" on page 11.

READ_LOCK_TYPE=

Specifies and changes the scope of the Teradata lock during SAS/ACCESS read operations.

Default value: None

Syntax

READ_LOCK_TYPE=TABLE | VIEW

TABLE
locks the entire table during a read operation.

VIEW
locks the entire view during a read operation.

Details If you do not use any SAS/ACCESS read lock options, Teradata locks at either the row or table level. Since SAS/ACCESS does not currently support a ROW lock, specify TABLE or VIEW only when you must lock the entire table or view.

For more information, see the LOCKING modifier in your Teradata SQL Reference manual, as well as "Understanding the Scope of SAS/ACCESS Lock Options" on page 11.

READ_MODE_WAIT=

Specifies during SAS/ACCESS read operations whether Teradata should wait to acquire a lock or fail your request when the DBMS resource is locked by a different user.

Default value: None

Syntax

READ_MODE_WAIT=YES | NO

YES
specifies that SAS/ACCESS should wait to acquire a lock for read operations if a different user holds a restrictive on page 21 lock.

NO
specifies that SAS/ACCESS should not wait.

Details If you specify READ_MODE_WAIT=NO, and a different user holds a *restrictive* lock, then the executing SAS step will fail. SAS/ACCESS continues processing the job by executing the next step. If you specify READ_MODE_WAIT=YES, SAS/ACCESS waits indefinitely until it can acquire the lock.

A *restrictive* lock means that another user is holding a lock that prevents you from obtaining your desired lock. Until the other user releases the restrictive lock, you cannot obtain your lock.

For more information, see the LOCKING modifier in your Teradata SQL Reference manual, as well as “Understanding the Scope of SAS/ACCESS Lock Options” on page 11.

SASDATEFMT=

Changes the SAS format that is associated with a DBMS column.

Default value: None

Syntax

SASDATEFMT=*(<DBMS-date-col-1='SAS-date-format'> <...*
<DBMS-date-col-n='SAS-date-format'>>)

DBMS-date-col
specifies the name of a Teradata table column that is a date type.

SAS-date-format
specifies a SAS date format that has a like-named informat.

For example, DATETIME21.2 is both a SAS format and SAS informat. It is an example of a valid value for the *SAS-date-format* argument.

Details When a SAS date, datetime, or time variable type does not match the type of the corresponding Teradata column, you must use SASDATEFMT= to convert the SAS type to the Teradata type. If the SAS and Teradata type match, the option is unnecessary.

For more information on the SASDATEFMT= option, see Chapter 4, "SAS/ACCESS Data Set Options" .

Example 1: Converting a SAS Datetime Type to a Teradata Date Type

```
libname x teradata user=kamdar password=ellis;
proc sql noerrorstop;
 create table x.dateinfo ( date1 date );
 insert into x.dateinfo
   ( sasdatefmt=( date1='datetime21.') )
     values ( '31dec2000:01:02:30'dt );
```

In this example, SASDATEFMT= correctly converts DATE1, a SAS datetime value, to a Teradata date column named DATE1.

Example 2: Converting a Teradata Date Type to a SAS Datetime Type

```
libname x teradata user=kamdar password=ellis;
data sas_local;
format date1 datetime21.;
set x.dateinfo( sasdatefmt=( date1='datetime21.') );
run;
```

In this example, SASDATEFMT= correctly converts DATE1, a Teradata date column, to a SAS datetime type named DATE1.

SCHEMA=

Specifies an alternative database to use when referring to a DBMS object. DATABASE= or DB= are aliases.

Default value: If you do not specify SCHEMA=, the default is the value of the SCHEMA= LIBNAME option on page 9.

Syntax

SCHEMA=<'> *alternate database*<'>

<'> *alternate database*<'>
specifies a database name.

Details The SCHEMA= option enables you to view or modify a different user's DBMS tables or views, assuming that you have the requisite Teradata privileges to that user's tables and views. For example, to read a different user's tables, you must have the Teradata privilege SELECT for that user's tables.

Example: Accessing A Different User's Table

```
libname mydblib teradata user=kamdar pw=ellis;
proc print data=mydblib.employees(schema=donna);
run;
```

In this example, user KAMDAR prints the contents of the EMPLOYEES table which is located in the DONNA database.

UPDATE_ISOLATION_LEVEL=

Specifies the level of isolation from other table users that is required as SAS/ACCESS reads Teradata rows in preparation for updating the rows.

Default value: None

Syntax

UPDATE_ISOLATION_LEVEL=ACCESS | READ | WRITE

ACCESS
obtains an ACCESS lock by ignoring other users' ACCESS, READ and WRITE locks.

READ
obtains a READ lock if no other user holds a WRITE or EXCLUSIVE lock.
Effect: prevents other users from being granted a WRITE or EXCLUSIVE lock.

WRITE
obtains a WRITE lock on the table or view if no other user has a READ, WRITE, or EXCLUSIVE lock on the resource. You cannot explicitly release a WRITE lock. A WRITE lock is released only when the DBMS object on which it is applied is closed.
Effect: Prevents other users from obtaining any lock but ACCESS.

Details

The READ value locks the entire table or view, allowing other users to acquire READ locks. The READ value can lead to deadlock situations.

The WRITE value prevents other users, except those with ACCESS locks, from accessing the table. Although a WRITE value prevents the possibility of a deadlock, it limits concurrent use of the table.

Avoid an ACCESS lock for update if there is a possibility that a different user might update the table at the same time. In this situation, the data can be corrupted.

Use SAS/ACCESS locking options only when Teradata's standard locking is undesirable. For more information, see the LOCKING modifier in your Teradata SQL Reference manual, as well as "Understanding the Scope of SAS/ACCESS Lock Options" on page 11.

UPDATE_LOCK_TYPE=

Specifies and changes the scope of the Teradata lock during SAS/ACCESS update operations.

Default value: None

Syntax

UPDATE_LOCK_TYPE=TABLE | VIEW

TABLE
locks the entire Teradata DBMS table during an update operation.

VIEW
locks the entire Teradata DBMS view during an update operation.

Details If you do not use any SAS/ACCESS update locking options, Teradata locks at either the row or table level. Since SAS/ACCESS does not currently support row-level locking, specify TABLE or VIEW only when you *must* lock the entire table or view. Use SAS/ACCESS locking options only when Teradata's standard locking is undesirable. For more information, see the LOCKING modifier in your Teradata SQL Reference manual, as well as "Understanding the Scope of SAS/ACCESS Lock Options" on page 11.

UPDATE_MODE_WAIT=

Specifies during SAS/ACCESS update operations whether Teradata should wait to acquire a lock or fail your request when the DBMS resource is locked by a different user.

Default value: None

Syntax

UPDATE_MODE_WAIT=YES | NO

YES
specifies that SAS/ACCESS should wait to acquire a lock for an update operation if a different user holds a restrictive on page 21 lock.

NO
specifies that SAS/ACCESS should not wait.

Details If you specify UPDATE_MODE_WAIT=NO, and a different user holds a restrictive lock, then your SAS step will fail and SAS/ACCESS will continue the job by processing the next step. If you specify UPDATE_MODE_WAIT=YES, SAS/ACCESS waits indefinitely until it can acquire the lock.

A *restrictive* lock means that a different user is holding a lock that prevents you from obtaining your desired lock. Until the other user releases the restrictive lock, you cannot obtain your lock.

Use SAS/ACCESS locking options only when Teradata's standard locking is undesirable. For more information, see the LOCKING modifier in your Teradata SQL Reference manual, as well as "Understanding the Scope of SAS/ACCESS Lock Options" on page 11.

SQL Procedure Pass-Through Facility: Teradata Specifics

You can use the pass-through facility of PROC SQL to build your own Teradata SQL statements and then pass them to the Teradata server for execution. The PROC SQL CONNECT statement defines the connection between SAS and the Teradata DBMS. See Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software" .

The following section describes the DBMS-specific arguments that you use in the CONNECT statement to establish a connection with a Teradata database.

Arguments to Connect to Teradata

The SAS/ACCESS Interface to Teradata can connect to multiple Teradata servers and to multiple Teradata databases. However, if you use multiple, simultaneous connections, you must use an *alias* argument to identify each connection.

CAUTION:
Do not issue a Teradata DATABASE statement within the EXECUTE statement in PROC SQL. Use the SAS/ACCESS SCHEMA= option if you must change the default Teradata database. △

The CONNECT statement uses SAS/ACCESS connection options. For more information, see SCHEMA= on page 9 and "SAS/ACCESS Engine Connection Options" on page 7.
USER and **PASSWORD** are the only required options.

CONNECT TO TERADATA <AS *alias*> (USER=*TERADATA-user-name*
PASSWORD=*TERADATA-password*
<TDPID=*dbcname*
SCHEMA=*alternate-database*
ACCOUNT=*account_ID*>);

USER=<'>*Teradata-user-name*<'>
specifies a Teradata user name. You must also specify PASSWORD=.

PASSWORD= | PASS= | PW= <'>*Teradata-password*<'>
specifies the Teradata password that is associated with the Teradata user name. You must also specify USER=.

Pass-Through Examples

This section presents simple pass-through examples. If you need background information to understand any example, or need more sophisticated examples, see the PROC SQL documentation available from SAS Institute.

Example 1: Using the Alias DBCON for the Teradata Connection

```
proc sql;
   connect to teradata as dbcon
           (user=kamdar pass=ellis);
 quit;
```

In Example 1, SAS/ACCESS

- Connects to the Teradata DBMS using the alias **dbcon**.
- Performs no other work.

Example 2: Deleting and Recreating a Teradata Table

```
proc sql;
   connect to teradata as tera ( user=kamdar password=ellis );
   execute (drop table salary) by tera;
   execute (create table salary (current_salary float, name char(10)))
            by tera;
   execute (insert into salary values (35335.00, 'Dan J.')) by tera;
```

```
   execute (insert into salary values (40300.00, 'Irma L.')) by tera;
   disconnect from tera;
 quit;
```

In Example 2, SAS/ACCESS

- Connects to the Teradata DBMS using the alias **tera**.
- Drops the SALARY table.
- Recreates the SALARY table.
- Inserts two rows.
- Disconnects from the Teradata DBMS.

Example 3: Updating a Teradata Table

```
proc sql;
  connect to teradata as tera ( user=kamdar password=ellis );
  execute (update salary set current_salary=45000
           where (name='Irma L.')) by tera;
  disconnect from tera;
 quit;
```

In Example 3, SAS/ACCESS

- Connects to the Teradata DBMS using the alias **tera**.
- Updates the row for Irma L., changing her current salary to 45000.00.
- Disconnects from the Teradata DBMS.

Example 4: Selecting and Displaying a Teradata Table

```
proc sql;
   connect to teradata as tera2 ( user=kamdar password=ellis );
   select * from connection to tera2 (select * from salary);
   disconnect from tera2;
  quit;
```

In Example 4, SAS/ACCESS

- Connects to the Teradata database using the alias **tera2**.
- Selects all rows in the SALARY table and displays them using PROC SQL.
- Disconnects from the Teradata database.

Data Object Naming Conventions

Teradata Conventions

The data objects that you can name in Teradata include tables, views, columns, indexes and macros. When naming a Teradata object, use the following conventions:

- A name must start with a letter unless you enclose it in double quotation marks.
- A name must be from 1 to 30 characters long.
- A name can contain the letters A through Z, the digits 0 through 9, the underscore (_), $, and #. A name in double quotation marks can contain any characters except double quotation marks.

- A name, even when enclosed in double quotation marks, is not case sensitive. For example, CUSTOMER is the same as customer.
- A name cannot be a Teradata reserved word.
- The name must be unique between objects. That is, a view and table in the same database cannot have the identical name.

SAS Naming Conventions

When naming a SAS object use the following conventions:

- A name must start with a letter or underscore.
- A name cannot be enclosed in double quotation marks.
- A name must be from 1 to 32 characters long.
- A name can contain the letters A through Z, the digits 0 through 9, and the underscore (_).
- A name is not case sensitive. For example, CUSTOMER is the same as customer.
- A name need not be unique between object types.

Naming Objects to Meet Teradata and SAS Conventions

To share objects easily between the DBMS and SAS, create names that meet both SAS and Teradata naming conventions. Make the name

- start with a letter
- include only letters, digits, and underscores
- have a length of 1 to 30 characters.

Accessing Teradata Objects That Do Not Meet SAS Naming Conventions

The following are SAS/ACCESS code examples to help you access Teradata objects (existing Teradata DBMS tables and columns) that have names that do not follow SAS naming conventions.

Example 1: Unusual Teradata Table Name

```
libname unusual teradata user=kamdar password=ellis;
proc sql dquote=ansi;
  create view myview as
  select * from unusual."More names";
proc print data=myview;run;
```

Example 2: Unusual Teradata Column Names

SAS/ACCESS automatically converts Teradata column names that are invalid for SAS, mapping any invalid characters to underscores. It also appends numeric suffixes to identical names to ensure that column names are unique.

```
create table unusual_names( Name$ char(20), Name# char(20),
                            "Other strange name" char(20))
```

In Example 2, SAS/ACCESS converts the spaces found in the Teradata column name, OTHER STRANGE NAME, to Other_strange_name. After the automatic conversion, SAS programs can then reference the table as usual, for example:

```
libname unusual teradata user=kamdar password=ellis;
proc print data=unusual.unusual_names; run;
```

Output 1.3 PROC PRINT Display

Name_	Name_0	Other_strange_name

Teradata Data Types

Every column in a table has a name and data type. The *data type* tells Teradata how much physical storage to set aside for the column, as well as the form in which to store the data. Teradata data types fall into categories: binary, character, date, and numeric data. Each type is described in the following sections.

Note: SAS/ACCESS Version 8 does not support the following Teradata data types: GRAPHIC, VARGRAPHIC and LONG VARGRAPHIC. △

Binary String Data

BYTE (*n*)
specifies a fixed-length column of length *n* for binary string data. The maximum for *n* is 64,000.

VARBYTE (*n*)
specifies a varying-length column of length *n* for binary string data. The maximum for *n* is 64,000.

Character String Data

CHAR (*n*)
specifies a fixed-length column of length *n* for character string data. The maximum for *n* is 64,000.

VARCHAR (*n*)
specifies a varying-length column of length *n* for character string data. The maximum for *n* is 64,000. VARCHAR is also known as CHARACTER VARYING.

LONG VARCHAR
specifies a varying-length column, of the maximum length, for character string data. LONG VARCHAR is equivalent to VARCHAR(32000) or VARCHAR(64000) depending on which Teradata version your server is running.

Date Data

The date type in Teradata is similar to the SAS date value. It is stored internally as a numeric value and is displayed in a site-defined format. The Teradata date type that SAS supports is listed here.

Note: Date type columns may contain Teradata values that are out of range for the SAS System, which handles dates from 1582 A.D. through 20,000 A.D. If SAS/ACCESS encounters an unsupported date, for example, a date earlier than 1582 A.D., it will return an error message and display the date as a missing value. △

DATE
specifies dates values in the default format YYYY-MM-DD. For example, January 25, 1989, is input as 1989-01-25. Values for this type can range from 0001-01-01 through 9999-12-31.

Numeric Data

Note: When reading Teradata data, SAS/ACCESS converts all Teradata numeric data types to the SAS System internal format, floating-point. △

BYTEINT
specifies a single–byte signed binary integer. Values can range from -128 to +127.

DECIMAL(*n*,*m*)
specifies a packed-decimal number. *n* is the total number of digits (precision). *m* is the number of digits to the right of the decimal point (scale). The range for precision is 1 through 18. The range for scale is 0 through *n*.

If *m* is omitted, 0 is assigned and *n* can also be omitted. Omitting both *n* and *m* results in the default DECIMAL(5,0). DECIMAL is also known as NUMERIC.

CAUTION:
Because SAS stores numbers in floating-point format, a Teradata DECIMAL number with very large precision can lose precision. For example, when SAS/ACCESS running on a UNIX MP-RAS client reads a Teradata column specified as DECIMAL (18,18), it maintains only 13 digits of precision. This can cause problems. A large DECIMAL number can cause the WHERE clause that SAS/ACCESS generates to perform improperly (fail to select the expected rows). There are other potential problems. For this reason, *use carefully* large precision DECIMAL data types for Teradata columns that SAS/ACCESS will access. △

FLOAT
specifies a 64-bit IEEE floating-point number in sign-and-magnitude form. Values can range from approximately 2.226×10^{-308} to 1.797×10^{308}. FLOAT is also known as REAL or DOUBLE PRECISION.

Note: When the SAS/ACCESS client internal floating point format is IEEE, Teradata FLOAT numbers convert precisely to SAS numbers. Exact conversion applies to SAS/ACCESS for Teradata running under UNIX MP-RAS. However, if

you are running SAS/ACCESS for Teradata under OS/390, there can be minor precision and magnitude discrepancies. △

INTEGER
specifies a large integer. Values can range from –2,147,483,648 through +2,147,483,647.

SMALLINT
specifies a small integer. Values can range from –32,768 through +32,767.

NULL and NOT NULL Values

The *NULL* value means an absence of information in Teradata and is analogous to the SAS System's *missing value*. Therefore, when SAS/ACCESS reads a Teradata NULL value, it generates a SAS missing value.

By default, Teradata columns accept NULL values. But, you can define columns so that they will not contain NULL values. To create Teradata columns that disallow NULL values, use the DBNULL= data set option. For example, when creating a SALES table, define the CUSTOMER column NOT NULL, telling Teradata not to add a row to the table unless the CUSTOMER column for the row has a value. For more information, as well as the syntax, see "DBNULL=" on page 18.

LIBNAME Statement Data Conversions

When you *read* a Teradata table, SAS/ACCESS assigns default SAS data types and formats to the Teradata table columns. In assigning the defaults SAS/ACCESS does not use Teradata's column format information. The Table 1.4 on page 30 shows the default SAS System formats that the LIBNAME statement assigns to the Teradata data types.

Note: The Teradata data types, GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC, are not supported by Version 8 of SAS/ACCESS. △

When you *create* Teradata tables, the default Teradata columns that SAS/ACCESS creates are based on the type and format of the SAS column. The Default Output Teradata Data TypesTable 1.5 on page 31 table shows the default Teradata data types that the LIBNAME statement assigns to the SAS System formats during output processing.

To override any default output type, use the data set option "DBTYPE=" on page 18.

Table 1.4 Default SAS Formats

Teradata Data Type	Default SAS Format
CHAR(*n*)	$*n*. (*n*<= 32,767)
CHAR(*n*)	$32767.(*n*>32,767) [1]
VARCHAR(*n*)	$*n*. (*n*<= 32,767)
VARCHAR(*n*)	$32767.(*n*> 32,767) [1]
LONG VARCHAR(*n*)	$32767. [1]
BYTE(*n*)	$HEX*n*. (*n*<= 32,767)
BYTE(*n*)[1]	$HEX32767.(*n*> 32,767)

Teradata Data Type	Default SAS Format
VARBYTE(*n*)	$HEX*n*. (*n*<= 32,767)
VARBYTE(*n*)	$HEX32767.(*n*> 32,767)
INTEGER	11.0
SMALLINT	6.0
BYTEINT	4.0
DECIMAL(*n*, *m*)[2]	(*n*+2).(*m*)
FLOAT	none
DATE[3]	DATE9.

1 When reading Teradata data into SAS, DBMS columns that exceed 32,767 bytes are truncated. The maximum size for a SAS character column is 32,767 bytes.
2 If the DECIMAL number is extremely large, SAS can lose precision. For details, see "Numeric Data."
3 See "Date Data" for how SAS/ACCESS handles dates that are outside the valid SAS date range.

Default Output Teradata Data Types

Table 1.5 Default Output Teradata Data Types

SAS Data Type	SAS Format	Teradata Data Type
Character	$*w*.	CHAR[*w*]
	$CHAR*w*.	
	$VARYING*w*.	
Character	$HEX*w*.	BYTE[*w*]
Numeric	A date format	DATE
Numeric	A time format[1]	FLOAT
Numeric	A datetime format[1]	FLOAT
Numeric	*w*.(*w*≤2)	BYTEINT
Numeric	*w*.(3≤*w*≤4)	SMALLINT
Numeric	*w*.(5≤*w*≤9)	INTEGER
Numeric	*w*.(*w*≥10)	FLOAT
Numeric	*w*.*d*	DECIMAL(*w*-1,*d*)
Numeric	All other numeric formats	FLOAT

1 FLOAT is the default Teradata output type for SAS time and datetime values. If you want SAS/ACCESS to display Teradata columns that contain SAS time and datetimes properly, you must explicitly assign the appropriate SAS time or datetime display format to the column. If you do not assign a format, SAS/ACCESS displays the value as a simple floating point number; it does not look like a time or datetime value.

Example: Output of a Teradata Table with Date and Datetime Columns

```
/* Outputs a Teradata table named sasdt        */
/* The dt and t columns default to Teradata    */
```

```
  /* FLOAT values.                                  */
libname tra teradata user=kamdar password=ellis;
data tra.sasdt;
  format dt datetime26.6;
         dt = '04jul1976:12:00:00.00000'dt;
  format t time16.6;
         t = '12:32:59.67765't;
run;
```

The PROC print statements that follow display the Teradata table. The first PROC PRINT statement displays the DT and T columns as FLOAT values. This is not very useful. The second PROC PRINT statement assigns formats to the Teradata DT and T columns. As a result, SAS displays the columns properly, as SAS date and datetime values.

```
 /*PROC PRINT Statement With No Assigned Formats*/
proc print data=tra.sasdt;
run;
```

Output 1.4 Sample PROC PRINT Display 1

Obs	dt	t
1	520948800	45179.68

```
 /*PROC PRINT Statement with Explicit Date/Datetime Formats*/
proc print data=tra.sasdt;
 format dt datetime26.6 t time16.6;
run;
```

Output 1.5 Sample PROC PRINT Display 2

```
Obs                          dt                          t

 1              04JUL1976:12:00:00.000000         12:32:59.677650
```

Maximizing Teradata Performance

A major objective of SAS/ACCESS when reading DBMS tables is to take advantage of Teradata's rate of data transfer. This section describes the actions that you can take to ensure that SAS/ACCESS delivers optimal read performance including

- "Enabling PreFetch" on page 33. For usage examples, see
  - "Using PreFetch as a LIBNAME Option" on page 35
  - "Using Prefetch as a Global Option" on page 36.
- "Matching Teradata Data Types to More Efficient SAS Formats" on page 37.

About the PreFetch Facility

PreFetch is a SAS/ACCESS for Teradata facility that speeds up a SAS job by exploiting the parallel processing capability of Teradata. To obtain benefit from the facility, your SAS job must run more than once and have the following characteristics:

- use SAS/ACCESS to query Teradata DBMS tables
- should *not* contain SAS statements that create, update, or delete Teradata DBMS tables
- run SAS code that changes infrequently or not at all.

In brief, the ideal job is a stable read-only SAS job.

Enabling PreFetch

Use of PreFetch is optional. To use the facility, you must explicitly enable it with the LIBNAME option PREFETCH= on page 9.

How Prefetch Works

When reading DBMS tables, SAS/ACCESS submits SQL statements on your behalf to Teradata. Each SQL statement that is submitted has an execution cost: the amount of time Teradata spends processing the statement before it returns the requested data to SAS/ACCESS.

When PreFetch is enabled, the first time you run your SAS job, SAS/ACCESS identifies and selects statements with a high execution cost. SAS/ACCESS then stores (caches) the selected SQL statements to one or more Teradata macros that it creates.

On subsequent runs of the job, when PreFetch is enabled, SAS/ACCESS extracts statements from the cache and submits them to Teradata in advance. The rows selected by these SQL statements are immediately available to SAS/ACCESS because Teradata 'prefetches' them. Your SAS job runs faster because PreFetch reduces the wait for SQL statements with a high execution cost. However, Prefetch improves elapsed time only on subsequent runs of a SAS job. During the first run, SAS/ACCESS only creates the SQL cache and stores selected SQL statements; no prefetching is performed.

The PreFetch Option Arguments

unique_storename As mentioned, when PreFetch is enabled, SAS/ACCESS creates one or more Teradata macros to store the selected SQL statements that PreFetch caches. You can easily distinguish a PreFetch macro from other Teradata macros. The PreFetch Teradata macro contains a comment that is prefaced with the text, **"SAS/ACCESS PreFetch Cache"**.

The name that the PreFetch facility assigns the macro is the value that you enter for the *unique_storename* argument. The *unique_storename* name must be unique. Do not specify a name that exists in the Teradata DBMS already for a DBMS table, view or macro. Also, do not enter a name that exists already in another SAS job that employs the Prefetch facility.

#sessions This argument specifies how many cached SQL statements SAS/ACCESS submits in parallel to Teradata. In general, your SAS job completes faster if you increase the number of statements that Teradata works on in advance. However, a large number (too many sessions) can strain client and server resources. A valid value is 1 through 9. If you do not specify a value for this argument, the default is 3.

In addition to the specified number of sessions, SAS/ACCESS adds an additional session for submitting SQL statements that are not stored in the PreFetch cache. Thus, if the default is 3, SAS/ACCESS actually opens up to 4 sessions on the Teradata server.

algorithm This argument is present to handle future enhancements. Currently PreFetch only supports one algorithm, SEQUENTIAL.

When and Why Use PreFetch

If you have a read-only SAS job that runs frequently, this is an ideal candidate for PreFetch. For example, a daily job that extracts data from Teradata tables. To help you decide when to use PreFetch, consider the following daily jobs:

- *Job 1*

  Reads and collects data from the Teradata DBMS.

- *Job 2*

  Contains a WHERE clause that reads in values from an external, variable data source. As a result, the SQL code that the job submits through a Teradata LIBNAME statement or PROC SQL changes from run to run.

In these examples, Job 1 is an excellent candidate for the facility. In contrast, Job 2 is not. Using PreFetch with Job 2 does not return incorrect results but can impose a performance penalty. PreFetch uses stored SQL statements. Thus, Job 2 is not a good candidate because the SQL statements that the job generates with the WHERE clause change each time the job is run. Consequently, the SQL statements that the job generates never match the statements that are stored.

The impact of Prefetch on processing performance varies by SAS job. Some jobs improve elapsed time 5% or less; others improve elapsed time 25% or more.

Possible Unexpected Results

It is unlikely, but possible, to write a SAS job that delivers unexpected or incorrect results. This can occur if the job contains code that waits on some Teradata or system

event before proceeding. For example, SAS code that pauses the SAS job until another user updates a given data item in a Teradata table. Or, SAS code that pauses the SAS job until a given time; for example, 5:00 p.m. In both cases, PreFetch would generate SQL statements in advance. But, table results from these SQL statements would not reflect data changes that will be made by the scheduled Teradata or system event.

PreFetch Processing of Unusual Conditions

PreFetch Facility is designed to handle unusual conditions gracefully. Some of these unusual conditions include:

Condition: Your job contains SAS code that creates updates, or deletes Teradata tables.
PreFetch is designed only for read operations and is disabled when it encounters a non-read operation. The facility will return a performance benefit up to the point where the first non-read operation is encountered. After that, SAS/ACCESS will disable the PreFetch facility and continue processing.

Condition: Your SQL cache name (*unique_storename* value) is identical to the name of a Teradata table.
PreFetch issues a warning message. SAS/ACCESS will disable the PreFetch facility and will continue processing.

Condition: You change your SAS code for a job that has PreFetch enabled.
PreFetch detects that the SQL statements for the job changed and deletes the cache. SAS/ACCESS will disable Prefetch and will continue processing. The next time that you run the job, PreFetch creates a fresh cache.

Condition: Your SAS job encounters a PreFetch cache that was created by a different SAS job.
PreFetch deletes the cache. SAS/ACCESS will disable Prefetch and will continue processing. The next time that you run the job, PreFetch creates a fresh cache.

Condition: You remove the PreFetch option from an existing job.
Prefetch is disabled. Even if the SQL cache (Teradata macro) still exists in your database, SAS/ACCESS ignores it.

Condition: You accidentally delete the SQL cache (the Teradata macro created by PreFetch) for a SAS job that has PreFetch enabled
SAS/ACCESS simply rebuilds the cache on the next run of the job. In subsequent job runs, PreFetch continues to enhance performance

Using PreFetch as a LIBNAME Option

If you specify the PreFetch= option in a LIBNAME statement, PreFetch applies the option to tables read by the libref.

Note: If you have more than one LIBNAME in your SAS job, and you specify PreFetch for each LIBNAME, remember to make the SQL cache name for each LIBNAME unique. △

Example 1: Applying PreFetch to One of Two LIBNAMES

```
libname one teradata user=kamdar password=ellis
 prefetch='pf_store1';
libname two teradata user=larry password=riley;
```

In Example 1, you apply PreFetch to one of two LIBNAMES. During the first job run, PreFetch stores SQL statements for tables referenced by the LIBNAME ONE in a Teradata macro named PF_STORE1 for reuse later.

Example 2: Applying PreFetch to Multiple LIBNAMES

```
libname emp teradata user=kamdar password=ellis
 prefetch='emp_sas_macro';
libname sale teradata user=larry password=riley
 prefetch='sale_sas_macro';
```

In Example 2, you apply PreFetch to multiple LIBNAMES. During the first job run, PreFetch stores SQL statements for tables referenced by LIBNAME EMP to a Teradata macro named EMP_SAS_MACRO and SQL statements for tables referenced by LIBNAME SALE to a Teradata macro named SALE_SAS_MACRO.

Using Prefetch as a Global Option

Unlike other Teradata LIBNAME options, you can also invoke PreFetch globally for a SAS job. To do this, place the OPTION DEBUG= statement in your SAS program before all LIBNAME statements and PROC SQL steps. If your job contains multiple LIBNAME statements, the global PreFetch invocation creates a uniquely named SQL cache name for each of the LIBNAMES.

Note: Do not be confused by the DEBUG option here. It is merely a mechanism to deliver the PreFetch capability globally. PreFetch is not for debugging; it is a supported feature of SAS/ACCESS for Teradata. △

Example 1: Using PreFetch Globally with Multiple LIBNAMES

```
option debug="PREFETCH(unique_mac,2,SEQUENTIAL)";
libname one teradata user=kamdar password=ellis;
libname two teradata user=kamdar password=ellis
                           database=larry;
libname three teradata user=kamdar password=ellis
                             database=wayne;
proc print data=one.kamdar_goods;
run;
proc print data=two.larry_services;
run;
proc print data=three.wayne_miscellaneous;
run;
```

In Example 1, the first time you run the job with Prefetch enabled, the facility creates 3 Teradata macros: UNIQUE_MAC1, UNIQUE_MAC2, and UNIQUE_MAC3. In subsequent runs of the job, PreFetch extracts SQL statements from these Teradata macros, enhancing the job performance across all three LIBNAMES referenced by the job.

Example 2: PreFetch Selects the Algorithm

```
option debug='prefetch(pf_unique_sas,3)';
```

In Example 2, PreFetch selects the algorithm, that is, the order of the SQL statements. (The option debug statement must the first statement in your SAS job.)

Example 3: User Specifies the Sequential Algorithm

```
option debug='prefetch(sas_pf_store,3,sequential)';
```

In Example 3, the user specifies for PreFetch to use the SEQUENTIAL algorithm. (The option debug statement must the first statement in your SAS job.)

Matching Teradata Data Types to More Efficient SAS Formats

You can further optimize read operations, with or without use of the PreFetch facility, by matching the DBMS data types to more efficient SAS formats. Internally, SAS employs floating point numbers and fixed-length character strings. The corresponding Teradata data types are FLOAT and CHAR. When SAS/ACCESS issues SELECT statements on your behalf, it can optimize read processing if the SELECT statement requests FLOAT and CHAR columns from Teradata. In this case, SAS/ ACCESS can access the data directly, reading it from the buffers that Teradata controls on the client (your PC, workstation or terminal). SAS/ACCESS then avoids having to move each column from Teradata's buffers into buffers maintained by SAS/ACCESS. Reading data from Teradata's client buffers improves performance. This is true especially when SAS/ACCESS reads a large amount of Teradata data.

In order to take advantage of the optimization just described, you may want to create new tables. For the new tables, where possible define FLOAT columns in place of DECIMAL, INTEGER, SMALLINT, and BYTEINT numeric Teradata columns. Similarly, where possible define CHAR columns in place of VARCHAR and LONG VARCHAR character Teradata columns. These data type changes can speed up processing. But, only when you read the data using SAS/ACCESS. Therefore, create a new table only if you intend to access the table primarily through SAS/ACCESS.

When creating a table, remember that changing other numeric types to FLOAT automatically increases the size of a Teradata table. Other numeric types require less physical storage than does the FLOAT data type. Therefore, before creating a table, consider the overall table design. Additionally, consider running benchmarks with SAS/ ACCESS.

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

SAS/ACCESS® Software for Relational Databases: Reference, Version 8 (Teradata Chapter)

ISBN 1-58025-551-5

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

1st printing, October 1999