SAS/ACCESS® Interface to Teradata

The Power to Know.

A SAS White Paper

SAS/ACCESS® Interface to Teradata

The Power to Know®

About the White Paper

Audience

The audience for this white paper is SAS and Teradata relational database management system (hereafter, Teradata DBMS) users who need fast, seamless access to Teradata DBMS tables from the SAS System.

Purpose

This white paper provides an overall view of the SAS/ACCESS Interface to Teradata to relate the interface to adjunct SAS products, to acquaint the user with the product, and to showcase its features and capabilities.

This paper provides general and technical product information[1], focusing first on the methods to access Teradata DBMS data, and then comparing the product's functionality and performance to other Teradata data access alternatives. Comparisons are made using a pro-and-con approach, expounding on where the product shines and pointing out where the user can get better results by using Teradata utilities. This approach is deliberate. Frequently, client applications that are compared to the Teradata DBMS promise more than they deliver. The objective of this paper is to guide Teradata users or database administrators reliably in their efforts to match software or a product feature to a processing situation.

Throughout this white paper, tips are provided on how to optimize a SAS session or job so that the processing is performed by the SAS System or by the Teradata DBMS, whichever is more appropriate. The objective is for SAS to process, that is, to have the SAS System *manage* but not necessarily *perform*, the work**.** SAS processing, given good performance, is preferable to separating the work, performing some tasks in SAS and the rest outside (for example, preprocessing or post-processing with Teradata DBMS utilities). Additionally, this paper answers many technical FAQs received from customers who use the SAS/ACCESS Interface to Teradata. The answers emphasize 'best use' of the SAS/ACCESS Interface to Teradata product.

Content and Organization

The audience for this paper is as wide as the content, which ranges from rudimentary to complex. The main ideas are presented as topics. Within a topic, a technical discussion and/or code examples are often provided. At the end of a topic, a summary is sometimes provided with a small graphic (shown left) to highlight an important point. This organization and a Table of Contents permit rapid skimming of topics of interest.

[1] Our intent is not to repeat the SAS/ACCESS Interface to Teradata installation instructions or the user guide documentation. For this type of information, see the section "What other documentation is available for a SAS or Teradata DBMS user?" on page 34.

Table of Contents

What is the SAS/ACCESS Interface to Teradata?

Consider this: Your site uses Teradata, a relational DBMS that enjoys an enviable reputation for reliable management of terabytes of data, sophisticated hardware that facilitates cutting-edge parallel processing, and an operating system that supports both the hardware and software. Your site has enthusiastic Teradata users who want to use SAS software to mine, warehouse, and analyze the DBMS data, thereby acquiring an arsenal of SAS tools from basic data exploitation (reports and graphs) to the latest technologies.

You wonder: Can I bridge these powerhouses -- the SAS System and the Teradata DBMS? You can with the SAS/ACCESS Interface to Teradata, client/server software that enables SAS users to transparently access and manipulate Teradata DBMS data. Transparent access simply means that SAS users can read and write data to and from the Teradata DBMS using either Base SAS software or SAS Enterprise Guide software. At the heart of the SAS/ACCESS Interface to Teradata is engine technology, a SAS mechanism that enables users to read or write data directly in a specific data format -- in this case, the Teradata DBMS format.

How does the SAS/ACCESS Interface to Teradata relate to Base SAS software and other SAS products?

The product summaries that follow are deliberately skeletal to show the relationship between the SAS/ACCESS Interface to Teradata and adjunct SAS products. With the exception of the Scalable Performance Data Server, the adjunct SAS products use the SAS/ACCESS Interface to Teradata to access Teradata DBMS data.

Base SAS Software

To access the Teradata DBMS, the user must install Base SAS software and the SAS/ACCESS Interface to Teradata, along with NCR's Teradata CLIv2 libraries on the client machine. The user then invokes the Teradata engine in a SAS session or job by specifying TERADATA in a SAS LIBNAME statement. Alternatively, the user can invoke the engine using a PROC SQL statement. See the section "How does the SAS/ACCESS Interface to Teradata work? What are the methods to access a Teradata DBMS" on page 12 for more information.

The LIBNAME statement method includes, by extension, all functionality of Base SAS software. Thus, with the LIBNAME method, the user can surface and manage Teradata DBMS tables along with SAS data sets. SAS programmers like to use the LIBNAME method because, usually, they do not have to make changes to existing PROC or DATA steps to run them successfully against the Teradata DBMS.

The SAS/ACCESS Interface to Teradata supplements Base SAS functionality with Teradata-specific capabilities, enabling SAS users to supersede some Teradata DBMS default behaviors. For example, the interface enables users, via SAS data set options, to create DBMS table columns that will not accept NULL values or to override Teradata's default row-locking via SAS/ACCESS Interface to Teradata locking options. For inexperienced SAS users, the Teradata engine offers non-programming tools such as the graphical SAS Explorer and Query windows. The SAS Explorer, with the familiar split-window display of libraries and the library files, permits novice users to operate on Teradata DBMS databases and tables without having to

write SAS code. Similarly, the Query window furnishes a graphical capability to query table data without the need to know SQL.

Sites that have more than one DBMS and who license additional corresponding SAS/ACCESS DBMS interfaces enjoy simple interoperability between the several databases and servers. (Interoperability means that users can easily extract data from one DBMS and load it to a different DBMS or join the disparate DBMS tables in a single SAS session or job.)

In summary, the SAS/ACCESS Interface to Teradata enables SAS users to tap the SAS System's highly acclaimed analytic tools and data exploitation capabilities using Teradata DBMS tables, without requiring the users to learn much more than the Teradata engine options. Or, it permits novice SAS users, who might be unfamiliar with SAS programming, to operate on Teradata DBMS tables merely by pointing and clicking from the SAS Explorer[2] or Query window.

SAS/ACCESS Interface to ODBC

ODBC is an established protocol that facilitates communication between a DBMS and an application that complies with the ODBC standard. Earlier, it was explained that the SAS/ACCESS Interface to Teradata is a SAS engine. In the SAS System, the SAS/ACCESS Interface to ODBC is also implemented as a SAS engine.

What is the difference between the two engines? One difference is that the SAS/ACCESS Interface to Teradata engine communicates directly with the Teradata DBMS, calling the Teradata CLIv2 interface. In contrast, the SAS/ACCESS Interface to ODBC engine communicates indirectly with the Teradata DBMS via the Teradata ODBC driver. This added layer accounts for some differences in capabilities and performance between the products.

Prior to SAS Version 8, SAS users often accessed Teradata DBMS data using the SAS/ACCESS Interface to ODBC because the SAS/ACCESS Interface to Teradata native engine was not available . Because both products can access Teradata DBMS data, users are frequently confused about the products. To clear up the confusion, a comparison of the Teradata and SAS/ODBC engines and some functional and performance differences between the two are presented in the section "SAS/ACCESS Interface to ODBC" on page 29.

SAS/Warehouse Administrator

SAS/Warehouse Administrator software is the tool of choice to ETL (*Extract, Transform, and Load*) DBMS data and other file data into a logical data warehouse. Designed for IT professionals who create and manage data warehouse and data mart processes, SAS/Warehouse Administrator can be customized, and it provides a single point of control to respond to the changing requirements of a business community.

Increasingly, end users understand that business intelligence systems must be based on the solid foundation of a data warehouse. However, writing programs to perform the tedious chores of ETL to create a business intelligence repository is time-consuming and frequently overextends the most productive IT departments. With its graphical interface, SAS/Warehouse Administrator is great for IT professionals

2 The SAS Explorer potentially allows users to retrieve terabytes of table data. Thus, the end user or the system administrator must know the data and restrict access appropriately to Teradata DBMS tables and processing.

because it simplifies the visualization, navigation, and maintenance of a data warehouse; it also eliminates much of the coding needed to build and manage the data warehouse.

In brief, SAS/Warehouse Administrator offers adaptability and manageability by

- integrating ETL tools
- storing and managing the metadata required to efficiently manage a warehouse
- facilitating business subject definitions and uniform business rules
- scheduling warehouse maintenance and integrating the processes with decision-support tools to exploit the data warehouse effectively
- leveraging core strengths of Base SAS software to deliver a data warehouse faster.

Whether a data warehouse is used with the Teradata DBMS alone or in combination with other databases, SAS/Warehouse Administrator employs native (DBMS) utilities and SAS/ACCESS interfaces to deftly handle all databases supported by the SAS System.

Specifically, for a Teradata DBMS, SAS/Warehouse Administrator software uses Teradata's FASTLOAD and MULTILOAD utilities, along with the SAS/ACCESS Interface to Teradata, to extract and load (refresh and append) Teradata DBMS tables. (For more options that SAS users have to rapidly load and refresh Teradata DBMS table data in the warehouse, see the section "Alternatives for Loading/Refreshing Teradata DBMS Tables" on page 28.)

SAS Enterprise Guide

SAS Enterprise Guide provides a point-and-click interface for connecting to SAS servers and automates the task of performing SAS data processing and analytical tasks. A true SAS thin client for the Microsoft Windows environment, SAS Enterprise Guide uses the COM (Component Object Model) architecture from Microsoft, which defines a structure for building program routines (objects) that can be called up and executed in a Windows environment.

SAS Enterprise Guide, a stand-alone Windows application, does not require installation of any other SAS software on the client machine. Assuming that the SAS/ACCESS Interface to Teradata is installed on the SAS server (see the section "Network Overview: Teradata DBMS (Server) and Clients" on page 11), the SAS Enterprise Guide client user can operate on Teradata DBMS tables as if they were SAS data sets and the user does not have to know SAS programming.

SAS Scalable Performance Data Server

The Scalable Performance Data Server is the SAS solution for implementing data marts that are gigabytes in size. A highly parallel data server, the Scalable Performance Data Server stores and retrieves SAS data and runs on most UNIX SMP and Windows SMP platforms. Frequently, a data warehouse solution centers on a Teradata DBMS, as well as on Oracle or DB2 databases. In these scenarios, the Scalable Performance Data Server can replace Oracle or DB2 for SAS applications – this is a cost-effective replacement that provides enhanced performance.

The SAS Scalable Performance Data Server enhances performance by using parallel algorithms for data-intensive processing operations, such as table scans, sorts, indexed WHERE clause evaluations, SELECTs

with aggregate functions (for example, GROUP BY, AVG, etc.), table loads or copies, and index creation. An enhancement planned for the Scalable Performance Data Server is an intelligent hybrid index that eliminates the need for users to choose the best type of index to obtain rapid query responses.

Besides boosting performance, the SAS Scalable Performance Data Server provides row/column security; utilities for incremental table/file backups and restores; file encryption and compression; and support for ODBC, JDBC, and htmSQL.

SAS Enterprise Miner

SAS Enterprise Miner is a powerful data exploration tool that is used frequently for customer-targeting campaigns, credit scoring, churn analysis, and fraud detection. Combined with SAS data warehousing and OLAP technologies, SAS Enterprise Miner offers the full spectrum of knowledge discovery, including exploratory data analysis, predictive modeling, and model deployment. Designed to examine large quantities of data, SAS Enterprise Miner enables users to discover hidden relationships and patterns that help to make intelligent decisions.

Which clients are supported by the SAS/ACCESS Interface to Teradata?

Mainframe	UNIX	PC
OS/390 (MVS)	AIX HP-UX 11 Solaris SPARC UNIX MP-RAS	Microsoft Windows NT Microsoft Windows 9x Microsoft Windows 2000 Microsoft Windows XP

The table lists SAS 8.2 (32-bit) clients. For clients on later releases of SAS software, check your SAS documentation.

Which releases of the Teradata DBMS server are supported?

Teradata DBMS servers can run on

- UNIX MP-RAS, a version of UNIX developed by NCR
- Microsoft Windows NT, Windows 2000, and Windows XP

The SAS/ACCESS Interface to Teradata supports both platforms as long as the release of the Teradata DBMS server is Version 2, Release 2 or higher. Beginning with Version 8 of SAS, the SAS/ACCESS Interface to Teradata runs interchangeably with these Teradata DBMS versions. Please note that support for Teradata's TIME and TIMESTAMP data types is available beginning with the SAS/ACCESS 8.1 Interface to Teradata.

What SAS software is required for accessing the Teradata DBMS?

- Base SAS
- SAS/ACCESS Interface to Teradata

Where must the required SAS software be installed?

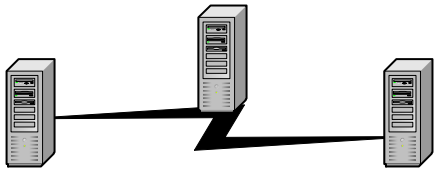
To access the Teradata DBMS, the user must install the required SAS software on the same machine where Teradata's TUF[3] client software, specifically the CLIv2 libraries, is installed. This machine is referred to as a SAS server and a DBMS client machine. (See the section "Network Overview: Teradata DBMS (Server) and Clients" on page 11 for examples of SAS servers.)

The SAS/ACCESS Interface to Teradata contacts the Teradata DBMS server by calling the Teradata CLIv2 library; the library, in turn, relies on standard Teradata supplied middleware.

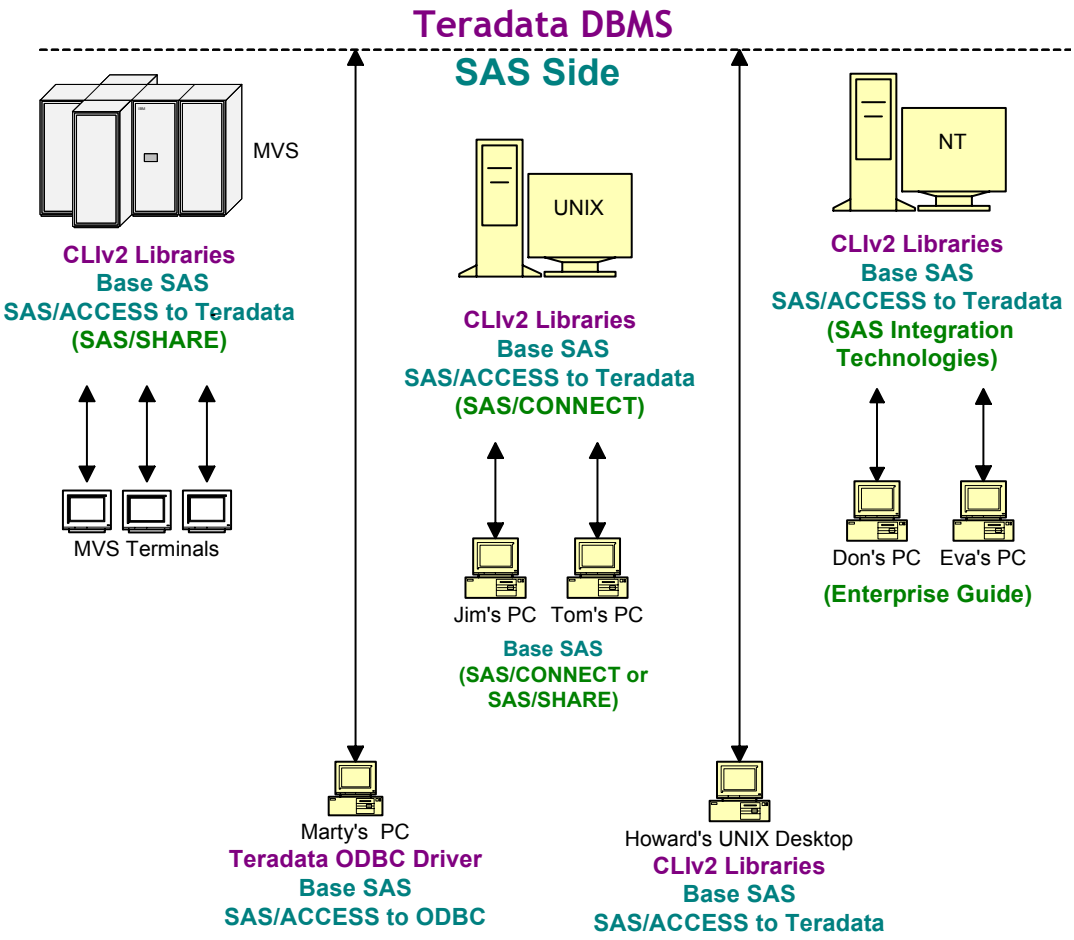
What additional software is required to set up an OS/390 client?

No additional software is required to set up an OS/390 client. The user needs only to have the CLIv2 libraries installed, specifically the APPLOAD load library. This library is provided with the TUF (Teradata Utilities Foundation) toolkit. Installation of the SAS/ACCESS Interface to Teradata to use under OS/390 is straightforward. The important thing to remember is to make sure that there is OS/390 to Teradata server connectivity, which enables BTEQ or another native Teradata DBMS utility to be run from OS/390. If you can run BTEQ, the SAS/ACCESS Interface to Teradata should work. (If you do not have Teradata software on your OS/390 system, contact NCR.)

[3] TUF software is the unified name for Teradata's client software. The TUF toolkit includes all the Teradata client software that you need to run the SAS/ACCESS Interface to Teradata, including the CLIv2 libraries and TDP/MOSI middleware.



MPP Cluster of UNIX MP-RAS SMP Boxes



Network Overview: Teradata DBMS (Server) and Clients

The Teradata DBMS (server) can run either on an MPP cluster of UNIX MP-RAS SMP boxes or on a Microsoft Windows NT SMP box. Most NCR customers run the more powerful MPP configuration. Typically, client applications connect to the DBMS from another machine[4]. OS/390, Windows NT, Solaris, AIX, and HP-UX seem to be more popular than MP-RAS for end-user applications, possibly because their operating systems do not have the two-gigabyte file size limit that MP-RAS imposes.

[4] In the figure shown here, the SAS/ACCESS Interface to Teradata is licensed on a single server node with multiple SAS clients accessing the Teradata DBMS through that SAS server. Optional SAS software that can be installed on the SAS server or PC clients is shown in parentheses. **For simplicity, the diagram does not show all possible configurations for either the SAS servers or the PC clients**.

Overriding the default response buffer length used by SAS

The response buffer length parameter is an internal setting that specifies the length of the buffer to hold data returned from the Teradata DBMS to SAS. For optimal performance, the parameter is set to 65535 for Version 2, Release 4 and above of the Teradata DBMS, and 32767 for prior versions. If there is a need to change the default response buffer length, update the response buffer length defaults for the Teradata client (the resp_buf_len field for Network Attached Clients and the IBCFBRL field for Channel Attached Clients) first. Refer to NCR documentation if you are not sure how to do this. Then, use the OVERRIDE_RESP_LEN option in a LIBNAME statement to indicate that the Teradata default for the response buffer length should be used instead of the value selected by SAS.

For example,

```
libname x teradata user=jo pw=XX server=dev override_resp_len=yes;
```

How does the SAS/ACCESS Interface to Teradata work? What are the methods to access a Teradata DBMS?

SAS/ACCESS Interface to Teradata offers three methods to access Teradata DBMS tables. Two of the methods, explicit and implicit SQL pass-through, are discussed under the topic “The SQL Pass-Through Method.” The third method is the SAS LIBNAME engine method and is discussed in the next section.

SQL is more familiar to Teradata DBMS users. For that reason, this paper elaborates on explicit and implicit SQL pass-through and provides mostly SQL examples. Keep in mind that the LIBNAME engine method provides interoperability and is a powerful feature of the SAS/ACCESS Interface to Teradata. Using the SAS LIBNAME engine method, SAS procedures and SAS DATA steps can seamlessly and simultaneously operate on Teradata DBMS tables. For more examples of the LIBNAME engine method than are given in this paper, see the Teradata chapter in SAS online documentation under the title, ”SAS/ACCESS for Relational Databases.”

The SAS LIBNAME Engine Method

With this method, you specify the Teradata engine name, TERADATA, along with connection options and other engine options, in a SAS LIBNAME statement. Once invoked, SAS/ACCESS Interface to Teradata generates SQL statements that equate to the SAS requests and submits the SQL that’s generated to the Teradata DBMS on your behalf. An example of a SAS LIBNAME statement follows:

```
/*Invokes the Teradata engine and connects to Teradata DBMS*/

libname trlib teradata user=testuser password=testpass;

/*Surfaces an existing Teradata DBMS table, EMP*/

proc print data=trlib.emp;
run;

/*Creates a Teradata DBMS table, NEWEMPLOYEES, that contains*/
/*employee numbers 7800 through 8000*/
```

```
data trlib.newemployees;
set trlib.emp;
where empno between 7800 and 8000;
run;
```

The SQL Pass-Through Method

SQL Pass-Through	User action in SAS/ACCESS Interface to Teradata	PROC SQL does …	DBMS Response is…
EXPLICIT	specifies Teradata specific SQL with PROC SQL.	passes the SQL exactly as written to the Teradata DBMS.	performs the SQL request if the syntax is correct; otherwise, the request fails.
IMPLICIT[5]	specifies SAS SQL with PROC SQL.	converts SAS SQL to Teradata specific SQL on your behalf and passes the SQL to the DBMS; executes the SQL portions of the queries, joins, etc. that cannot be converted in SAS.	performs the SQL request if the functionality is supported; otherwise, returns an error condition that triggers SAS processing.

Explicit Pass-Through Method

With this method, you invoke the SQL procedure and specify a statement that requests connection to the Teradata server, followed by SQL requests. PROC SQL submits your SQL statements to the Teradata DBMS. Assuming that your SQL syntax is correct, the Teradata DBMS performs the processing requested. Because you specify the precise SQL that the procedure passes to the DBMS, this method is known as explicit pass-through.

Implicit Pass-Through Method

Similar to the LIBNAME engine method, the SQL procedure can generate SQL statements on your behalf. This behind-the-scenes processing is known as implicit pass-through. The purpose of implicit pass-through is to have SAS, via PROC SQL, pass as much work as possible to the underlying DBMS, in this case to Teradata. However, implicit pass-through is subject to rules that are discussed in the next section "Understanding the Rules of Implicit Pass-Through."

Understanding the Rules of Implicit Pass-Through

Passing a query to the Teradata DBMS via implicit pass-through is analogous to a horse jumping hurdles to win a race. One hurdle is that the query must have basic characteristics that make it eligible for implicit

[5] Implicit pass-through is actually a series of performance enhancements to SAS PROC SQL. For more on this subject and to learn the current rules for SAS System 9 and higher, see "What other documentation is available for a SAS or Teradata DBMS user?" on page 34.

pass-through. Another hurdle is that the query must not contain any elements that would cause PROC SQL to disqualify it.

Basic Eligibility

For basic eligibility, any query or part of a query must

- refer to a single SAS/ACCESS LIBNAME
- be legal according to the ANSI SQL2 standard
- be recognized by PROC SQL.

Effect of DBMS SQL2 Conformance

The DBMS can reject a legal SQL2 query passed by SAS because the DBMS might not support all three levels of SQL2 conformance. Describing a legal SQL2 query for the Teradata DBMS is beyond the scope of this paper. Instead, SQL keywords are listed that trigger PROC SQL to pass the query or query part to the Teradata DBMS, and elements within the query that would cause PROC SQL to disqualify it are identified.

SQL Keywords That Trigger Implicit Pass-Through

The following SQL keywords[6] trigger PROC SQL to pass the query to the Teradata DBMS.

- DISTINCT
- Aggregate functions
- count(*)
- count(x)
- freq(x)
- n(x)
- avg(x)
- mean(x)
- max(x)
- min(x)
- sum(x)
- JOIN
- UNION

Elements That Disqualify Implicit Pass-Through

PROC SQL disqualifies any query or query part that involves one or more of the following elements.

- CONNECTION TO
- Remerging
- Data set options
- One or more truncated comparisons
- INTO clause
- One or more ANSI MISS/NOMISS inner or outer joins
- A SAS function that is not in the aggregate function list above or in the SAS data function list (see the section "Data Functions That Are Passed to the Teradata DBMS" on page 15).

Thus, PROC SQL does not pass a query to the Teradata DBMS that contains a WHERE clause that specifies an unsupported SAS function, for example, the FUZZ[7] function. Instead, PROC SQL returns the query to SAS for processing.

[6] This list is continually expanding. To obtain a complete list for SAS 9 and higher, consult your SAS documentation.
[7] The FUZZ: function returns the nearest integer if the argument is within 1E-12.

Data Functions That Are Passed to the Teradata DBMS

Beginning with SAS 8.2, the SAS/ACCESS Interface to Teradata [8] passes the following data functions to the Teradata DBMS for processing:

- ABS
- EXP
- LOG
- LOG10
- SQRT
- LOWCASE
- SUBSTR
- TODAY/DATE
- UPCASE
- YEAR
- MONTH
- DAY

Example of Implicit Pass-Through of SAS TODAY Function

```
proc print data=trlib.tbl;
    where x=today();
    run;
```

The SAS TODAY function is equivalent to the Teradata DBMS CURRENT_DATE data function. Therefore, implicit pass-through in SAS 8.2 and higher generates the following Teradata specific SQL for the preceding WHERE clause code in SAS:

```
select "x" from "tbl" where ("x" = current_date )
```

Explicit and Implicit Pass-Through: Side-by-Side Code Examples

```
/*Setup: Create a tiny Teradata DBMS table for the example*/
libname trlib teradata user=testuser pw=testpass;
data trlib.customr16;
   input custname $ 1-10 custnum custcity $ 22-36;
   datalines;
Beach Land       16  Ocean City
Coast Shop        3  Myrtle Beach
Coast Shop        5  Myrtle Beach
Coast Shop       12  Virginia Beach
Coast Shop       14  Charleston
Del Mar           3  Folly Beach
Del Mar           8  Charleston
Del Mar          11  Charleston
New Waves         3  Ocean City
New Waves         6  Virginia Beach
Sea Sports        8  Charleston
Sea Sports       20  Virginia Beach
Surf Mart       101  Charleston
Surf Mart       118  Surfside
Surf Mart       127  Ocean Isle
Surf Mart       133  Charleston
run;
```

[8] Check your SAS documentation for updates to the list for SAS 9 and higher.

SQL Pass-Through Examples

<table>
<tr><th>Explicit SQL</th><th>Implicit SQL</th></tr>
<tr><td>

```
option sastrace=',,,d'
sastraceloc=saslog no$stsuffix;
title2 'Customer Cities';



proc sql noerrorstop;
  connect to teradata (user=testuser
  password=testpass);
  select * from connection to teradata
  (select distinct custcity from
  customr16);
quit;
```

</td><td>

```
Option sastrace=',,,d'
Sastraceloc=saslog no$stsuffix;
Title2 'Customer Cities';
Libname trlib teradata   user=testuser
  pw=testpass;

Proc sql noerrorstop;
  select distinct custcity from
  trlib.customr16;
quit;
```

</td></tr>
<tr><th colspan="2">SAS LOG</th></tr>
<tr><td>

```
1 option sastrace=',,,d'
2 sastraceloc=saslog no$stsuffix;
3 title2 'Customer Cities';
4 proc sql noerrorstop;
5 connect to teradata (user=testuser
  pw=XXXXXXX);
6 select * from connection to teradata
7 (select distinct custcity from
  customr16);
Prepare stmt: select distinct custcity
  from customr16
Prepare SQL(trprep): select distinct
  custcity from customr16
Trget: rows to fetch: 7
8 quit;
```

</td><td>

```
libname trlib teradata user=testuser
  pw=XXXXXXX;
NOTE: Libref TRLIB was successfully
  assigned as follows:
      Engine:        TERADATA
      Physical Name:
2 option sastrace=',,,d'
3 sastraceloc=saslog no$stsuffix;
4 title2 'Customer Cities';
5 proc sql noerrorstop;
6 select distinct custcity from
  trlib.customr16;
Prepare SQL(trprep): SELECT * FROM
  "customr16"
Prepare stmt:  select distinct
  "customr16"."custcity" from "customr16"
Prepare SQL(trprep):  select distinct
  "customr16"."custcity" from "customr16"
SQL Implicit Passthru stmt prepared is:
  select distinct "customr16"."custcity"
  from "customr16"
trget: rows to fetch: 7
7 quit;
```

</td></tr>
<tr><th colspan="2">SAS LST</th></tr>
<tr><td>

```
The SAS System
Customer Cities

custcity
---------------
Charleston
Folly Beach
Myrtle Beach
Ocean City
Ocean Isle
Surfside
Virginia Beach
```

</td><td>

```
The SAS System
Customer Cities

Custcity
---------------
Charleston
Folly Beach
Myrtle Beach
Ocean City
Ocean Isle
Surfside
Virginia Beach
```

</td></tr>
</table>

In the examples above, look at the SQL that is marked in bold. Notice that, with explicit pass-through, the SQL that is submitted to Teradata is exactly as specified; with implicit pass-through, the SAS/ACCESS Interface to Teradata prepares the SQL that is submitted on your behalf. (In this example, the SQL submitted to Teradata is essentially the same.)

The following section about performance discusses the LIBNAME engine (see the section "How does the SAS/ACCESS Interface to Teradata work? What are the methods to access a Teradata DBMS" on page 12) and SQL pass-through connection methods more fully, detailing how SQL pass-through can come into play. For now, remember that one or both connection methods can be used in a single job/session to enable any feature or function that the method supports.

Both SAS/ACCESS connection methods call the CLIv2 interface (the same interface that Teradata's native utilities: FASTLOAD, MULTILOAD, and BTEQ call). Beneath the CLIv2 layer is NCR middleware, for example, TDP on OS/390, or MTDP and MOSI on UNIX and Microsoft Windows.

Capturing Implicit and Explicit SQL Statements to the SAS Log

Often, users want to see the SQL that the engine generates or that they specify. The user can see the SQL that's generated by using the following option in the SAS session or program (the preceding example also uses this option).

```
option sastrace=',,,d'
sastraceloc=saslog no$stsuffix;
```

Once set, this option writes to the SAS log the SQL that the Teradata engine passes to the DBMS. Consequently, you can view the SQL that is generated and executed on your behalf when implicit pass-through is triggered via PROC SQL.

Processing Performance - Good, Better, and Best

Across the board, the SAS/ACCESS Interface to Teradata provides excellent read processing performance. Performance of non-read table operations varies dramatically but there are ways to enhance the default performance. In the next section, performance-enhancing alternatives for the following operations are presented:

- loading empty tables
- appending[9] to tables already containing rows
- updating rows
- deleting rows
- upserting[10]
- joining a small SAS table with a large Teradata DBMS table

In the introduction, it was stated that the goal of the writers, given good performance, is to have SAS manage the processing work. Therefore, the suggested alternatives for enhanced performance frequently use the SAS/ACCESS FASTLOAD option. When you specify FASTLOAD=YES, SAS/ACCESS directly and quickly loads data to Teradata DBMS tables.

SAS/ACCESS FASTLOAD shares some characteristics of the Teradata FASTLOAD utility.

---

[9] For a definition of an **append** operation, see the section "Table Append Operation" on page 19.

[10] For a definition of an **upsert** operation, see the section "Table Upsert Operation" on page 20.

Optimizing Large-Scale DBMS Table Operations: Some Code Examples

The SAS/ACCESS Interface to Teradata can insert, update, and delete rows seamlessly on behalf of the SAS user. But, when these operations involve **many** rows, the engine's default methods can be inefficient and resource-intensive. This section shows you how to optimize large-scale operations, as well as how to perform an upsert operation from SAS. Notes about related Teradata DBMS utilities are also provided. (To distinguish Teradata native utilities from SAS software, Teradata utility notes are italicized.)

How examples are set up

The examples shown here refer to two tables with identical column layouts. TeraMaster, the master table into which rows will be inserted, updated, or deleted, is stored in the Teradata DBMS. SASTrans, a SAS data set (SAS table) that supplies the insert and update transactions, is stored locally in your client SAS session. In these examples, the TeraMaster table is located in the PLAYAREA database.

It is assumed that the following SAS LIBNAME statement has been previously issued:

```
libname tra teradata user=testuser password=testpass
  database=playarea;
```

Table Load Operation

A table load consists of inserting rows into an empty table. By default, the SAS/ACCESS Interface to Teradata inserts one row at a time. However, when there are many rows, the default processing is slow (expensive). The performance-enhancing alternative is to enable SAS/ACCESS FASTLOAD and have SAS perform the loading.

SAS/ACCESS FASTLOAD delivers high performance with only two constraints: duplicate rows are dropped and error detection and correction is weak. (For complete details, see the SAS and Teradata documentation on FASTLOAD and the section "How does SAS/ACCESS FASTLOAD affect writes to Teradata DBMS tables?" on page 35.)

Loading Empty TeraMaster Table

Default Example: Load without SAS/ACCESS FASTLOAD Enabled

```
proc sql;
insert into tra.TeraMaster  select * from SASTrans;
```

Enhanced Performance Example: Load with SAS/ACCESS FASTLOAD Enabled

```
proc sql;
insert into tra.TeraMaster(FASTLOAD=yes)  select * from
  SASTrans;
```

Teradata DBMS Utilities Note:
The Teradata FASTLOAD utility is the corresponding Teradata DBMS mechanism for high-volume data loading of Teradata DBMS tables. Performance of SAS/ACCESS FASTLOAD is equivalent to the Teradata FASTLOAD utility.

Table Append Operation

A table append consists of inserting rows into a non-empty table, which is a table that already contains some rows. Because there are many rows to insert, you want to circumvent a row-at-a-time insert. In this example, SAS/ACCESS FASTLOAD is used to load an intermediary Teradata DBMS table. Next, explicit SQL statements are passed to Teradata to append data from that intermediary table. Upon completion, the intermediary table is deleted. Keep in mind that duplicate rows in the original "append set" will be removed as a result of the FASTLOAD step and are lost to the target Teradata DBMS table. (See the section "How does SAS/ACCESS FASTLOAD affect writes to Teradata DBMS tables?" on page 35.)

Appending Non-Empty TeraMaster Table

Default Example: Append without SAS/ACCESS FASTLOAD Enabled

```
proc append data=SASTrans base=tra.TeraMaster; run;
```

Enhanced Performance Example: Two-Step Append with SAS/ACCESS FASTLOAD Enabled

```
proc sql;

/*Step 1*/

create table tra.Intermediary(FASTLOAD=yes) as select * from
  SASTrans;
connect to teradata (user=testuser password=testpass
  database=playarea);

/*Step 2*/

execute ( insert into TeraMaster select * from Intermediary ) by
  teradata;
execute ( drop table Intermediary ) by teradata;
execute( commit ) by teradata;
```

Limiting the number of insertion errors

The ERRLIMIT option in a LIBNAME statement causes load processing to terminate after the specified number of errors are reached. In the following example, there is a constraint violation. FASTLOAD terminates after the specified error limit is reached due to constraint violation.

```
libname mydblib teradata user=terauser pw=XXXXXX ERRLIMIT=10;
data tra.TeraFload(FASTLOAD=yes dbtype=(i='int check (i > 11)') );
  do i=1 to 50000;output;
  end;
run;
```

The error tables are locked when FASTLOAD is paused. The error tables can only be read by setting the READ_ISOLATION_LEVEL to ACCESS.

```
proc sql;
select ERRORCODE from tra.SAS_FASTLOAD_ERRS1_1379495443
  (read_isolation_level=access
  read_mode_wait=no
  read_lock_type=table);
quit;
```

The error code can be used to determine why FASTLOAD failed by checking Teradata documentation.

The target table for FASTLOAD must be deleted before attempting to reload.

Teradata DBMS Utilities Note:
The Teradata MULTILOAD utility is the corresponding Teradata DBMS mechanism for high-volume data appending. Its performance is comparable to the two-step process shown in the previous example. It will preserve duplicate rows - an important requirement in some processing situations.

Table Upsert Operation

A table upsert updates rows that match on a specified key. Rows in the transaction table that do not match a master table key are appended to the master. Most SAS/ACCESS interfaces support upsert capability through the SAS DATA step MODIFY clause. But, the SAS/ACCESS Interface to Teradata does not support the DATA step MODIFY clause. (For technical details, as well as an example of a one-step upsert operation, see the section "Can I perform upsert processing without using a SAS DATA step and the MODIFY clause?" on page 38.)

In the following example, an explicit SQL is used to perform an upsert, quickly loading the SAS intermediary table. Again, note that using FASTLOAD for the intermediary table will drop duplicate rows.

High Performance Example: Multi-Step Upsert with SAS/ACCESS FASTLOAD Enabled

```
proc sql;

/*Step 1*/

create table tra.Intermediary(FASTLOAD=yes) as select * from
  SASTrans;

/*Step 2: Update common rows (transaction record matches master
record)*/

connect to teradata (user=testuser password=testpass
  database=playarea);
  execute (update TeraMaster set LastPurchase =
  Intermediary.LastPurchase where TeraMaster.CustId =
  Intermediary.CustId) by teradata;

/*Step 3: Delete common rows from transaction table*/

execute (delete from Intermediary where Intermediary.CustId =
  TeraMaster.CustId) by teradata;

/*Step 4: Append remaining (new customers) transaction rows*/

execute (insert into TeraMaster select * from Intermediary) by
  teradata;
  execute ( drop table Intermediary ) by teradata;
execute (commit) by teradata;
quit;
```

Teradata DBMS Utilities Note:
The Teradata MULTILOAD utility is the corresponding Teradata DBMS mechanism for high-volume upsert operations. Its performance is somewhat better than the multi-step process shown above. It preserves duplicate rows in the transaction table - an important requirement in some processing situations.

Table Update and Delete Operations

A table update changes the value of one or more columns based on an optionally specified condition. A delete drops rows based on an optionally specified condition. The SAS/ACCESS Interface to Teradata updates and deletes one row at a time -- this[11] is inefficient and time-consuming when many rows must be

[11] This behavior is applicable to SAS releases 8.2 and earlier.

updated or deleted. An enhanced performance alternative is to pass an explicit Teradata specific update or delete statement to the Teradata DBMS, which performs these operations in parallel.

Default Delete and Update Example:

```
proc sql;
delete * from tra.TeraMaster where visits > 99999;
update tra.TeraMaster set visits=visits+1 where visits < 50000;
```

Enhanced Performance Example: Delete and Update

```
proc sql;
connect to teradata (user=testuser password=testpass
  database=playarea);
execute (delete from  TeraMaster where visits > 99999) by teradata;
execute (update TeraMaster set visits = visits +1 where visits <
  50000) by teradata;
execute( commit ) by teradata;
```

Teradata DBMS Utilities Note:

You can issue identical SQL as shown in the enhanced performance example with the Teradata BTEQ utility, thereby obtaining the same functionality. The performance of the code is identical to the SAS/ACCESS Interface to Teradata explicit SQL.

Table Read Operations

SAS System 9 supports NCR's FASTEXPORT utility for Teradata reads, which is the fastest way of reading large volumes of data. Additionally, a new option called threaded reads is supported that uses partitioning WHERE clauses to improve performance over single-threaded reads. SAS 9 supports threaded reads for Teradata on Windows only.

FASTEXPORT

The FASTEXPORT utility must be licensed separately from NCR to be used by SAS 9. FASTEXPORT is invoked by using the DBSLICEPARM=ALL option. SAS 9 automatically generates the specialized script required for the FASTEXPORT utility and retrieves the data back from FASTEXPORT on sockets. FASTEXPORT is supported on explicit SQL as well. Following is an example.

```
libname tra Teradata user=terauser pw=XXXXXX server=boom;
proc freq data=tra.big(dbsliceparm=all);
table x1-x3;
run;
```

Following is an example using explicit SQL.

```
proc sql;
  connect to teradata(user=terauser password=XXXXXX server=boom
  dbsliceparm=all);
select * from connection to teradata
  (select * from big);
quit;
```

If SAS 9 cannot find the FASTEXPORT utility in the environment's path, it automatically switches to threaded reads instead. With threaded reads, SAS generates partitioning WHERE clauses using the MOD function. If tracing is turned on, the following message can be seen in the SAS log when SAS cannot find FASTEXPORT and switches to threaded reads.

```
sasiotra/trautogn(): FASTEXPORT not found on your system 8 1379616470
no_name 0 REG
sasiotra/trautogn(): Cannot FASTEXPORT. Reverting to MOD slicing. 9
1379616470 no_name 0 REG
```

Coding tip:
Fully threaded applications, such as PROC REG, will always try to use FASTEXPORT and, if it is unavailable, will switch to threaded reads. It is not necessary to code DBSLICEPARM=ALL for such procs.

Debug tip:

If FASTEXPORT is available on UNIX or Windows and SAS cannot find it, check to see if FASTEXPORT is in the environment's path.

Performance tip:
Teradata has a limit on the number of Teradata utilities that can be run concurrently (the number varies from 5 to 15 depending on system settings). Every invocation of FASTLOAD and FASTEXPORT that uses SAS is counted in the number of concurrent Teradata utilities executing. Therefore, it is important to make judicious use of FASTLOAD and FASTEXPORT.

Coding tip:
If only some tables need to be read with FASTEXPORT, use the DBSLICEPARM=ALL data set option for the specific tables being read. If DBSLICEPARM=ALL is used in the LIBNAME statement, all tables will be read using FASTEXPORT.

Threaded read
SAS 9 supports a new performance option called threaded read. For Teradata, this option is supported on Windows platforms only. The threaded read option enables data to be read from Teradata in parallel on multiple threads. SAS does this by adding partitioning WHERE clauses using the MOD function to the query submitted to the DBMS. For example, instead of submitting a query such as "Select Name, SSN from Customer," the following queries will be submitted to partition the result set.

```
"Select Name, SSN from Customer where ABS( Age MOD 3)=0"
"Select Name, SSN from Customer where ABS( Age MOD 3)=1"
"Select Name, SSN from Customer where ABS( Age MOD 3)=2"
```

Each result set can now be retrieved in parallel on three different threads. Note that this depends on the existence of a suitable column (such as Age in this case) for applying the MOD function.

Use the DBSLICEPARM=DBI option to invoke threaded reads.

Performance tip:
DBSLICEPARM=ALL will perform threaded reads if FASTEXPORT is not available, but the option includes the overhead of checking for the existence of FASTEXPORT.

In the following example, the SQL submitted for threaded reads is shown.

```
libname tra teradata user=terauser pw=XXXXXX server=boom;
proc freq data=tra.big(dbsliceparm=dbi);
table x1-x3;
run;
```

The SAS log shows the following SQL.

```
SELECT "X1","X3","X2" FROM "big"  WHERE  ABS("X1" MOD 2)=0 67
1379617052 no_name 0 FREQ
SELECT "X1","X3","X2" FROM "big"  WHERE (ABS("X1" MOD 2)=1 OR "X1" IS
NULL) 71 1379617052 no_name 0 FREQ
```

In the following example, the SQL submitted for single-threaded reads is shown.

```
proc freq data=x.big;
table x1-x3;
run;
```

Single-threaded reads submit the following SQL.

```
SELECT "X1","X3","X2" FROM "big"  110 1379617571 no_name 0 FREQ
```

For SAS to automatically generate WHERE clauses, there must be existing columns in Teradata that are suitable for applying the MOD function.

The following data types are suitable.

Byteint
Smallint
Integer
Date
Decimal (Integral decimal columns only)

If SAS is unable to automatically generate the partitioning WHERE clauses, or if the user is familiar with the table data, the DBSLICE= option may be used to specify the WHERE clauses needed for partitioning the Teradata DBMS for threaded reads.

```
data tra.minor;
set lib.customer(DBSLICE=("AGE<18" "AGE>=18"));
where cash='Y';
run;
```

The SAS log shows that the following SQL was submitted to the DBMS.

```
SELECT "age","cash" FROM "customer"  WHERE  ("cash" = 'Y' ) AND  AGE<18
12 1380570347 no_name 0 DATASTEP
SELECT "age","cash" FROM "customer"  WHERE  ("cash" = 'Y' ) AND
AGE>=18 16 1380570347 no_name 0 DATASTEP
```

Debug tip:
If threaded reads are requested on UNIX and OS/390, SAS will revert to single-threaded reads automatically because threaded reads are not supported on those platforms. The following message is displayed in the SAS log if tracing is turned on.

```
sasiotra/trautogn(): MOD slicing is not supported on OS/390 and UNIX.
```

Joining a SAS Data Set with a Teradata DBMS Table

When you use PROC SQL to join a SAS table with a Teradata DBMS table, SAS pulls the Teradata DBMS table into SAS for the join. If the Teradata DBMS table is much larger than the SAS table (which is often the case), this operation is resource-intensive. In this situation, consider customizing the code to load the SAS table into Teradata (use FASTLOAD if the SAS table is more than a few hundred rows); then use implicit or explicit SQL pass-through to perform the join in Teradata, returning only the result set to SAS.

Enhancing Performance: Ensuring Teradata DBMS Server Processing

Situations That Cause Teradata DBMS Processing to Occur

Because processing can be enhanced when performed by the Teradata DBMS, SAS users are eager to know how to move processing to the Teradata DBMS side. In the next section is a review of two situations discussed previously that pass work to the DBMS (for more detail, see the section "
The SQL Pass-Through Method" on page13).

Explicit SQL Pass-Through

PROC SQL passes user-supplied Teradata-DBMS-specific SQL to the Teradata DBMS server.

Implicit SQL Pass-Through
PROC SQL transparently generates SQL for Teradata queries that contains operations such as joins, distinct processing, and SQL-defined aggregate functions.

There is a third, not yet mentioned, situation that triggers DBMS processing: WHERE clause processing.

WHERE Clause Processing
SAS passes a WHERE clause to the DBMS whenever the WHERE clause can be processed by the DBMS.

Because most users understand WHERE clause processing, discussion here is limited to SQL pass-through situations only.

Facilitating Performance: The Best Connection Method for the Situation

SQL Pass-Through Method

Explicit Pass-Through: Enables User-Written, Teradata Specific SQL

The use of SQL syntax to affect DBMS processing is an advantage to a Teradata DBMS user. However, an experienced SAS user who is more familiar with SAS syntax might consider using SQL a disadvantage.

Beyond an ease-of-use consideration, explicit pass-through is the only method that allows you to code Teradata specific SQL. For this reason, explicit SQL pass-through offers you the most flexibility. Specifying the correct Teradata specific SQL against the Teradata DBMS, you can affect almost any functionality that can be implemented with Teradata native tools. A simple example is setting or retrieving DBMS table column constraints with SQL pass-through.

In contrast, using the SAS/ACCESS Interface to Teradata LIBNAME method, the Teradata engine generates the SQL for you. For tasks oriented in SAS, the LIBNAME method is sufficient -- you do not need the flexibility of explicit SQL for this processing.

Implicit Pass-Through: Optimizes Many Data Queries, Joins, and Data Functions

Implicit pass-through[12] passes a growing number of queries and joins to the Teradata DBMS, instead of having SAS process them on the client. Beginning with SAS 8.2, it passes many data functions as well.

Prior to implicit pass-through, SAS processed a SQL query involving one or more DBMS tables as if they were separate SAS data files. This meant that the SAS SQL procedure fetched all rows from each DBMS table and performed the join within SAS.

Example of Implicit Pass-Through

In the following example, the site has two large DBMS tables named TABLE1 and TABLE2. The tables have the DEPTNO column in common. The user wants to perform an inner join of the tables, thereby retrieving rows where the DEPTNO value in TABLE1 equals the DEPTNO value in TABLE2.

```
proc sql;
    select tab1.deptno, dname from
       dblib.table1 tab1
       dblib.table2 tab2
       where tabl.deptno = tab2.deptno
 quit;
```

When implicit pass-through is enabled (the default), the SQL procedure detects a join between two tables within the same library (when the SAS *libref* references a database). Once detected, the SAS/ACCESS Interface to Teradata passes the join request directly to the Teradata DBMS. Assuming that the DBMS processes the inner join, Teradata returns only the result row set to SAS. The example processes an inner join. However, implicit pass-through supports both inner and outer joins between two or more Teradata DBMS tables.

In the example, the query join follows the implicit pass-through rules that were discussed earlier (see the section ”Understanding the Rules of Implicit Pass-Through“ on page 13), allowing PROC SQL to pass the request to Teradata. What happens if the query did not comply with the rules? PROC SQL passes the request back to SAS for processing.

Obviously, it is more efficient when you have large tables to have Teradata perform the join and return only the result set. This reduces network traffic, enhances processing efficiency, and eliminates any dollar penalty that a site can incur by performing work on the SAS side. For example, a site with an OS/390 machine can have attendant billing fees. When SAS, which resides on the OS/390 client, performs the processing instead of Teradata, the site incurs real dollar costs.

SAS Options to Manage Implicit Pass-Through

Because implicit pass-through is subject to several variables, how does the user know if it occurred? SAS provides two options to help manage implicit pass-through: DIRECT_SQL (LIBNAME option) and _METHOD (PROC SQL option). The first option, DIRECT_SQL, toggles implicit pass-through on or off. The second option, _METHOD, generates output that tells the user how the query was processed and whether implicit pass-through occurred.

[12]SAS, Version 7 and higher

The user can obtain the best possible performance for queries and joins when explicit SQL pass-through is used because it is guaranteed that the Teradata DBMS will perform the query or join processing. However, implicit pass-through provides another opportunity to have SAS generate Teradata specific SQL.

Understanding and managing implicit pass-through will help the user take better advantage of this transparent processing. Because it is automatic, it relieves the user of having to know or learn Teradata specific SQL to affect DBMS requests. But, implicit pass-through code has an added benefit; it is portable. When implicit pass-through code is used, it can be ported easily from one SAS/ACCESS engine to another.

Summary: Whenever possible, rely on implicit pass-through, reserving use of explicit SQL pass-through for processing situations that require user intervention to ensure optimal performance.

Explicit SQL Pass-Through: Enables Basic Engine Options

Explicit SQL pass-through supports the SAS/ACCESS Interface to Teradata connection and database options. These options include the following:

- USER=
- PASSWORD=
- TDPID=[13]
- ACCOUNT=
- DATABASE=[14]

To employ functionality available with other Teradata engine options, you must use the LIBNAME engine method.

LIBNAME Engine Method: Enables Advanced Engine Features

Because the LIBNAME engine connection method supports all the Teradata engine options, it provides the software's most sophisticated features.

- ## PREFETCH

  This option, when applicable, submits multiple SQL queries for the Teradata DBMS to process in parallel.

  ```
  PREFETCH=
  ```

- ## SAS/ACCESS Interface to Teradata Locking Options

  The following Teradata engine locking options override the default locking provided by the Teradata DBMS.

  ```
  READ_ISOLATION_LEVEL
  READ_MODE_WAIT
  READ_LOCK_TYPE
  UPDATE_ISOLATION_LEVEL
  UPDATE_MODE_WAIT
  UPDATE_LOCK_TYPE
  ```

[13] The alias for this option is SERVER=.
[14] The alias for this option is SCHEMA=.

- SAS/ACCESS FASTLOAD: A Load Capability without User Scripts

  The SAS/ACCESS Interface to Teradata supports the loading of data to Teradata DBMS tables with the option

  ```
  FASTLOAD= (alias BULKLOAD=)
  ```
  [15]

Measuring the Performance of the SQL Generated

Earlier, when discussing Teradata engine connection methods, it was pointed out that the LIBNAME engine method generates SQL for the user. In this section, that SQL is broadly described, and SAS options are identified that enable the user to view the SQL that is generated, as well as to find out the processing time. This information will enable the user to benchmark and tune processing performance.

A Broad Overview of the SQL That the LIBNAME Engine Generates

A DBMS table and a traditional SAS data set have features in common. SAS variables are equivalent to DBMS columns; similarly, SAS observations are equivalent to DBMS rows. SAS engine technology defines which operations are supported on a data set/table and then calls an established set of routines to obtain support, alternating the call depending on the data source type.

In turn, the operations call code that is shared by all SAS/ACCESS interfaces. This portable or shared code translates SAS operations such as open, get, put, delete, does table exist, etc. into the required, DBMS-specific SQL. Because joins are critical to performance, SAS uses a separate code layer to optimize query joins, attempting whenever possible to pass the join processing to the DBMS. In brief, the job of the SAS/ACCESS Interface to Teradata (similar to any SAS/ACCESS engine) is to issue correct and optimized DBMS-specific SQL.

Capturing SQL and Timer Information to the SAS Log

Beginning in SAS 8.2, the following option statement logs the SQL statements that are submitted to Teradata and the time that the DBMS took to perform the request. The timer statistics can tell the user where time is spent during a processing operation, helping isolate a performance bottleneck. In essence, the user learns which SQL processing is expensive and thus a good candidate for optimization.

```
option debug=dbms_timers
sastrace=',,,d'
sastraceloc=saslog no$stsuffix;
```

Rapidly Loading Table Data

Loading huge amounts of data to a Teradata DBMS table requires a rapid data load or refresh mechanism. Fortunately, Teradata DBMS users have alternatives when loading and refreshing DBMS table data. The following table characterizes some of the alternatives.

[15] You can enable the FASTLOAD= option globally in a LIBNAME statement, enabling you to apply the feature to an entire SAS session or job. We refer to this feature as FASTLOAD, rather than by its alias BULKLOAD, because the FASTLOAD name is familiar to Teradata users. Please note that other SAS/ACCESS DBMS interfaces call this feature BULKLOAD.

Alternatives for Loading/Refreshing Teradata DBMS Tables

Alternative	Vendor	Characteristics
Writes Data to an Empty DBMS Table (a FASTLOAD Candidate)		
Teradata FASTLOAD Utility	NCR	• Very fast • Populates empty Teradata DBMS tables • Executes outside of SAS
SAS/ACCESS FASTLOAD	SAS	• Very fast • Populates empty Teradata DBMS tables • Executes from within SAS • Must be enabled (by default, this option is off) • Permits *'closed loop'* processing[16]
SAS/Warehouse Administrator Loader Add-In	SAS	• Requires SAS/Warehouse Administrator software • Populates empty SAS/Warehouse Administrator Teradata DBMS tables • Generates scripts that invoke the Teradata FASTLOAD utility
Appends Data to a DBMS Table with One or More Rows (a MULTILOAD Candidate)		
Teradata MULTILOAD Utlility	NCR	• Fast • Appends data to existing Teradata DBMS tables • Executes outside of SAS
SAS/ACCESS Interface to Teradata Two-Step Append Using SAS/ACCESS FASTLOAD	SAS	• Fast • Appends data to existing Teradata DBMS tables • Executes from within SAS • Uses the SAS/ACCESS Interface to Teradata FASTLOAD option in conjunction with explicit pass-through SQL • Drops duplicate rows
SAS/Warehouse Administrator Loader Add-In	SAS	• Requires SAS/Warehouse Administrator software • Appends data to existing SAS/Warehouse Administrator Teradata DBMS tables • Generates scripts that invoke the Teradata MULTILOAD utility

[16] **Closed Loop**: No intermediary processing steps are required, such as writing SAS data to a flat file then using the Teradata FastLoad Utility to process the file.

Comparing Functionality and Performance of the SAS/ACCESS Interface to Teradata and SAS/ACCESS Interface to ODBC

Functionality

Capability	SAS/ACCESS Interface to Teradata	SAS/ACCESS Interface to ODBC
Transaction Semantics	ANSI Mode or Teradata Mode*	ANSI Mode or Teradata Mode*
Can override Teradata DBMS locking defaults	Yes	No
FASTLOAD capability	Yes	No
PREFETCH Feature	Yes	No
SAS Technical Support	Yes	Partial**
Read/Insert	Yes	Yes
Default Update/Delete	Yes	No***

* By default, the ODBC driver is set to ANSI mode, but the user can select Teradata mode and override the default. The SAS/ACCESS Interface to Teradata runs in ANSI mode. The user can specify Teradata mode only by using SQL explicit pass-through (see the section "Can I open Teradata mode sessions using the SAS/ACCESS Interface to Teradata?" on page 37.)

** The SAS/ACCESS Interface to ODBC depends on the ODBC driver that is supplied by NCR. Therefore, any ODBC driver problems must be relayed to, and resolved by, NCR.

*** These SAS operations can be accomplished by using the SAS/ACCESS Interface to ODBC only when the user writes Teradata specific SQL using explicit SQL pass-through.

Performance

Relative performance of both SAS engines is subject to several variables, including

- the client platform that the user runs the product from
- the version, older or newer, of the NCR-supplied Teradata ODBC driver that is being used
- the hardware and network configurations that the user sets up.

Because of the variables involved, we supply a general percentage (if possible) when comparing performance of the following processing operations.

Operation	Definition	Performance
Raw read performance	How fast the product reads a large numbers of rows	The SAS/ACCESS Interface to ODBC takes 30% or more time using various NCR ODBC drivers than the SAS/ACCESS Interface to Teradata takes.
Raw write performance	How fast the product performs high-volume inserts into Teradata DBMS tables	The default insert performance is comparable between the engines. However, when SAS/ACCESS FASTLOAD[17] is enabled, the SAS/ACCESS Interface to Teradata is faster.

[17] To enable the option, specify FASTLOAD=YES, either as a LIBNAME or data set option.

Comparing Functionality of the SAS/ACCESS Interface to Teradata and Teradata BTEQ

Functionality

Capability	SAS/ACCESS Interface to Teradata	BTEQ
Teradata Mode Available	YES	YES

ANSI Mode Requires the COMMIT Statement with Explicit Pass-Through

The Teradata DBMS supports two transaction semantics modes, Teradata and ANSI. The SAS/ACCESS Interface to Teradata defaults to ANSI mode (see the section "Can I open Teradata mode sessions using the SAS/ACCESS Interface to Teradata?" on page 37 to override the default).

Because of the ANSI setting, the interface can behave differently from other products that the user is familiar with; for example, Teradata's BTEQ utility (which defaults to Teradata mode). In the FAQ section, key differences between the Teradata and ANSI modes are highlighted. (The Teradata DBMS documentation explains the differences in greater detail.)

When using the SAS/ACCESS Interface to Teradata and running in ANSI mode, the most important difference to remember is that the user must issue the SQL COMMIT statement in order to commit transactions to the Teradata DBMS.

The user does not need to use COMMIT statements when using the implicit SQL pass-through or the LIBNAME engine methods because, in these situations, the SAS/ACCESS Interface to Teradata issues the COMMIT on the user's behalf. But, when users write explicit SQL, they must issue an explicit COMMIT statement along with any SQL request that modifies the database (SQL statements that create, update, modify, and drop Teradata DBMS tables). The COMMIT statement is unnecessary for read requests (for example, SELECT). In the explicit SQL examples that follow, comments are embedded to show when and where an explicit COMMIT statement is required.

ANSI Semantics Mode: Two SQL Examples That Show Required COMMIT Statements

```
proc sql;
  connect to &engine( user=testuser password=testpass
  database=testdbase );
  execute (drop table test ) by teradata;

/*Drop Table is a DDL statement and requires a COMMIT*/

  execute ( commit )  by teradata;
  execute ( create table test (counter int) )  by teradata;

/* Create Table is a DDL statement and requires a COMMIT*/

  execute ( commit )  by teradata;
  execute ( insert into test values( 1 ) ) by teradata;
  execute ( insert into test values( 2 ) ) by teradata;

/*Without a COMMIT, the inserts are rolled back*/

  execute ( commit )  by teradata;
  quit;
```

```
proc sql;
  connect to &engine( user=testuser password=testpass
  database=testdbase );
  select * from connection to teradata( select * from test);

/*COMMIT unnecessary because this is a read-only operation*/

  quit;
```

Comparing Performance and Functionality of the SAS/ACCESS Interface to Teradata and the Teradata FASTLOAD Utility

Functionality

Capability	SAS/ACCESS FASTLOAD	Teradata FASTLOAD Utility
Uses parallel processing	Yes	Yes
Requires scripts to execute	No	Yes
Eliminates duplicate records	Yes	Yes
Reduces error checking	Yes	Yes
Writes a single entry to the Teradata transaction log (if logging is enabled)	Yes	Yes
Restarts	No	Yes

Performance

The performance of high-volume inserts into Teradata DBMS tables by SAS/ACCESS FASTLOAD is equivalent to the Teradata FASTLOAD utility.

FAQs by subject

Client Setup

How do I know if the SAS client machine is set up correctly to access the Teradata DBMS?

A The SAS/ACCESS Interface to Teradata requires CLIv2 libraries, part of the TUF Toolkit, to be installed on the client machine where Base SAS software is installed. If the Teradata CLIv2 libraries are installed on the client and the native Teradata BTEQ facility works, the SAS/ACCESS Interface to Teradata is set up correctly and will work properly on all platforms, including OS/390.

What must I do to the client machine to connect to a Teradata server?

A If you can access Teradata from your client machine by using Teradata utilities such as BTEQ, you do not need to make any other system changes to use the SAS/ACCESS Interface to Teradata. If you cannot access the Teradata DBMS, read further to understand the connection between the SAS/ACCESS Interface to Teradata client and the Teradata DBMS server.

Simple Scenario: A Single Teradata DBMS Server

Most Teradata sites run a single Teradata server that stores all their Teradata DBMS tables. In this scenario, PC and UNIX clients typically have a single entry, DBCCOP1, set in their HOSTS file. The entry provides the IP address of the Teradata server. (In the examples that follow, we assume the TCP/IP address of the Teradata server is 11.25.20.34.) Usually, on UNIX systems, the HOSTS file is located in */etc/hosts*; on Windows 95, 98, and Millennium systems, the HOSTS file is located in c:\windows; and on Windows NT and 2000 systems, the HOSTS file is located in *c:\winnt\system32\drivers\etc\Hosts*.

HOSTS File: Example Entry for a Single Teradata DBMS Server

```
11.25.20.34       dbccop1    # default dbcname
```

LIBNAME Statement: Example for a Single Teradata Server

When the HOSTS file of a PC or a UNIX client contains the above entry, BTEQ should work. And, given a valid user name, password, and database, the SAS/ACCESS Interface to Teradata should connect to the Teradata DBMS.

To test a SAS/ACCESS connection, bring up SAS and issue the following LIBNAME statement. (You can omit the TDPID=[18] option if the default server name, DBCCOP1, is set up in the HOSTS file and you run a single Teradata server.)

```
libname test teradata user=TESTUSER password=TESTPASS
database=TESTUSER tdpid=dbc;
```

OS/390 client machines set up a single TDP task, usually called TDP0, which addresses the Teradata server. Under OS/390, after your administrator has started TDP0, the SAS/ACCESS Interface to Teradata should connect to the Teradata DBMS server. To test an OS/390 connection, bring up SAS and issue the following LIBNAME statement. (You can omit the TDPID= option if your administrator has set TDP0 as the default in the HSISPB and HSHSPB modules, and you run a single Teradata server.)

```
libname test teradata user=TESTUSER password=TESTPASS
database=TESTUSER tdpid=tdp0;
```

Complex Scenario: Multiple Teradata DBMS Servers

Your site can have more than one Teradata DBMS server running. For example, when upgrading the Teradata server version, a production system could be running day-to-day operations while a test system is validating the latest Teradata release. To access multiple Teradata DBMS servers on PC and UNIX systems, there must be multiple entries (one for each database server in the client's HOSTS file) that provide the name and IP address of each server.

HOSTS File: Example Entries for Multiple Teradata DBMS Servers

```
11.25.20.34    dbccop1    # Teradata production system
12.33.19.58    testcop1   # Teradata test system (new release)
```

LIBNAME Statements: Example for Multiple Teradata Servers

On PC and UNIX systems, to access either system, you must use the TDPID= option as follows.

```
libname test teradata user=TESTUSER password=TESTPASS
 database=TESTUSER tdpid=test;

libname prod teradata user=TESTUSER password=TESTPASS
 database=TESTUSER tdpid=dbc;
```

On OS/390 systems, there will be more than one TDP task running. The TDP task for the test system is likely to be named TDP1. You can access the production system as before, but to get to the test system, you must use the TDPID= option as follows.

```
libname test teradata user=TESTUSER password=TESTPASS
 database=TESTUSER tdpid=tdp1;
```

* CAUTION: If you run multiple Teradata servers and you want to connect to more than a single server in the same job or session, you must use the TDPID= specifier in each LIBNAME or SQL connection statement.

[18] The alias for this option, SERVER=.

The following is more background on connectivity.

The SAS/ACCESS Interface to Teradata uses a database connection name, DBCNAME, in conjunction with the connection option TDPID= to access a Teradata database. The content of DBCNAME depends on how the client machine is attached to a Teradata DBMS server, whether it is network-attached (PC and UNIX) or channel-attached (OS/390).

Network-Attached Systems (PC and UNIX)

You must specify a DBCNAME entry in your client HOSTS file for every database server, thereby providing Teradata with an IP address. Each DBCNAME must have the suffix COP*x*, where *x* is a number. If you make a single entry, *x* must be 1. The default DBCNAME that the SAS/ACCESS Interface to Teradata expects is DBCCOP1. If you make a DBCCOP1 entry in your HOSTS file and run a single Teradata server, you do not need to specify the Teradata engine connection option TDPID=.

If you run more than one Teradata server, you must enter a DBCNAME (eight characters or less, excluding the suffix) for every server. Subsequently, you must specify TDPID= and the appropriate DBCNAME for the server you want to connect to. Note that when making multiple entries for the same DBCNAME (a name identical except for the number), *x* must begin with 1 and increment sequentially.

Channel-Attached Systems (OS/390)

TDPID= specifies the subsystem name, which must be TDP*x*, where *x* can be 0-9, A-Z (not case-sensitive), or one of the following special characters: $, #, or @. By default, the Teradata client software for OS/390 is installed with a subsystem name of TDP0. If you have a single Teradata server and your OS/390 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 OS/390. For more examples, see the TDPID= connection option in your SAS online documentation.

What can I expect when upgrading the SAS client with newer TUF software?

A You can expect an easy transition. At SAS, TUF 5.6 on OS/390 and TUF 6.0 on other clients is used. In the past, earlier TUF releases were used. Because of the architecture of the SAS/ACCESS Interface to Teradata and TUF, no problems were encountered using any TUF release.

Data Types

Where can I learn more about data types?

A The *"Data Types"* section in the SAS/ACCESS Interface to Teradata chapter of your SAS online documentation describes Teradata DBMS data types that the interface supports, and details how the data types are converted when the engine reads them from or writes them to the Teradata DBMS.

Can I issue a native Teradata DBMS data type, such as TIMESTAMP and DATE, without writing explicit SQL?

A Yes, the examples below show two ways to do this. For more information, see the section *"Using TIME and TIMESTAMP*" in the Teradata chapter in the SAS documentation titled “SAS/ACCESS for Relational Databases”. Specifically, look for examples that issue a LIBNAME statement (for example, use the LIBNAME connection method) and assert a SAS format or specify DBTYPE= to create the Teradata DBMS type.

Asserting a SAS Format to Create a Teradata TIMESTAMP

```
libname trlib teradata user=testuser password=testpass;
data trlib.stamps;
format stamp0 datetime25.0
stamp0 = '13apr1961:12:30:55.123456'dt;
run;
```

Using DBTYPE= to Create a Teradata TIMESTAMP

You can use the Teradata engine data set option, DBTYPE=, to issue a native data type.

```
libname trlib teradata user=testuser password=testpass;
data trlib.stamps(dbtype=(stamp1='TIMESTAMP(2)' ));
stamp1 = '12dec1912:12:12:12'dt;
run;
```

Documentation

What other documentation is available for a SAS or Teradata DBMS user?

A Additional sources are grouped by subject.

SAS/ACCESS Interface to Teradata

- Installation Instructions
- SAS Online Documentation (User Guide)
  ⇒ SAS/ACCESS Software for Relational Databases, Teradata Chapter

Because the SAS online documentation that is shipped with the SAS/ACCESS Interface to Teradata is updated with each new release, this information can change.

Implicit Pass-Through

- SUGI White paper 51, "Performance Enhancements to PROC SQL in Version 7 of the SAS System"
  Lewis Church, Jr., SAS Institute
  http://www2.sas.com/proceedings/sugi24/Advtutor/p51-24.pdf

The writers of this document borrowed liberally from this paper. Although the content is dated, it gives the history of implicit pass-through and details use of the _METHOD option in PROC SQL.

- "Potential Result Set Differences between Relational DBMSs and the SAS System"
  Fred Levine, SAS Institute
  http://www.sas.com/rnd/warehousing/papers/resultsets.pdf

This paper describes a few processing situations (for example, the presence of NULL values) in which you can obtain different results when the WHERE processing is performed by the DBMS instead of SAS.

Teradata DBMS

- NCR Documentation
  http://www.ncr.com/library/library.htm

NCR provides complete Teradata documentation at this URL. The files, available in PDF format, can be easily downloaded or read online with Adobe Acrobat Reader.

DBMS Drivers and Utilities

BTEQ Utility

Do I need BTEQ to use the SAS/ACCESS Interface to Teradata?

A No. We suggest verifying Teradata DBMS connectivity with BTEQ because Teradata DBMS users are familiar with that utility. The SAS/ACCESS Interface to Teradata requires only NCR's CLIv2 and SAS to be loaded on the client. BTEQ, a separate NCR product, can be on the client but is not required or used by the product.

FASTLOAD Utility

How does SAS/ACCESS FASTLOAD affect writes to Teradata DBMS tables?

A When FASTLOAD is not enabled (the default), SAS/ACCESS Interface to Teradata feeds rows one at a time to the Teradata DBMS. Comprehensive error checking, referential integrity, etc. is in place on both the sending SAS side and the receiving Teradata DBMS side. The engine passes an INSERT statement to the Teradata DBMS for each row, along with the binary data. In turn, the Teradata DBMS re-parses the INSERT statement. Thus, re-parsing is required for each INSERT statement.

FASTLOAD, whether performed by SAS/ACCESS FASTLOAD or the Teradata FASTLOAD utility, eliminates re-parsing of the INSERT statement. But, the inherent nature of FASTLOAD results in a loss of granularity in error detection and subsequent error recovery. (Detailed error information is not returned to the calling application and, as a result, cannot be passed back to the user.) In most cases, the loss of error detail is acceptable given the user's requirements and the speed that FASTLOAD delivers.

When SAS/ACCESS FASTLOAD is enabled, you obtain a vast increase in speed because the facility opens multiple sessions to the Teradata DBMS. As a consequence, SAS/ACCESS FASTLOAD can pump data to the DBMS in parallel. (The Teradata FASTLOAD and MULTILOAD utilities also perform parallel processing.)

If FASTLOAD error detection is weak, how do I know if my operation succeeded?

A Look at your SAS log, specifically, at messages that follow the FASTLOAD step. If the messages reference duplicate rows or error tables, you know something went wrong. Keep in mind that you will see the normal SAS note about 'variables and observations written.' This does not tell the whole story. If there are no additional messages, FASTLOAD was successful. If there are additional messages specific to FASTLOAD, FASTLOAD was not successful.

Does SAS/ACCESS FASTLOAD support checkpointing and automatic restarting?

A Support for checkpointing is available. For information about the capability, see the BULKLOAD= (FASTLOAD=) and CHECKPOINT= options described in the Teradata chapter in "SAS/ACCESS for Relational Databases" in the SAS online documentation.

SAS/ACCESS FASTLOAD does not support automatic restart of SAS/ACCESS FASTLOAD after a DBMS restart. Additionally, it does not support the Teradata DBMS 'tenacity' mechanism that re-attempts a FASTLOAD after the DBMS has exceeded the maximum number of concurrent FASTLOAD operations.

How many Teradata sessions does SAS/ACCESS FASTLOAD use?

A SAS/ACCESS FASTLOAD allocates as many sessions as there are AMPs on the Teradata DBMS server. If you want to limit the number of sessions that are allocated, use the undocumented option, SESSIONS=. For example:

```
data trlib.testload (FASTLOAD=YES Sessions= 4);
```

MULTILOAD Utility

The SAS/ACCESS Interface to Teradata does not support MULTILOAD; can I still append to DBMS tables rapidly?

A Yes, if you are willing to use a multi-step process. The multi-step processing suggested earlier is much faster than the normal SAS row-by-row append. Remember that duplicate rows are dropped. (For details, see the section "Enhanced Performance Example: Two-Step Append with SAS/ACCESS FASTLOAD Enabled" on page 19.)

ODBC Driver

Does the SAS/ACCESS Interface to Teradata require a Teradata ODBC driver?

A No. The SAS/ACCESS Interface to Teradata is a CLI interface and does not use the ODBC protocol to connect to the Teradata DBMS. Thus, the presence or absence of the Teradata ODBC driver on the client machine has no effect on functionality.

PREFETCH

Where can I learn more about PREFETCH?

A The SAS/ACCESS Interface to Teradata chapter (see the section "What other documentation is available for a SAS or Teradata DBMS user?" on page 34) explains the PREFETCH capability and describes how to use it.

What is the actual PREFETCH sessions limit?

A The SESSIONS parameter for PREFETCH is global, implying that if you use the default PREFETCH(3), your limit is three sessions. However, if you have multiple LIBNAMES open simultaneously **and** set the option PREFETCH globally (use PREFETCH for every LIBNAME), the engine supports three sessions **per** LIBNAME.

Where are the macros created by the PREFETCH facility stored?

A The PREFETCH macro(s) are written to the database that you connect to in your LIBNAME statement. If you use the DATABASE= option in your LIBNAME statement, it writes the macro(s) to that database; otherwise, it writes the macro(s) to your default[19] database (the database that you automatically connect to in the absence of the DATABASE= specification).

What happens if I do not have permission to create macros in the database that I accessed?

A SAS/ACCESS Interface to Teradata will issue a CREATE MACRO request and the Teradata DBMS will return an error message. SAS/ACCESS Interface to Teradata passes the error message back and continues processing. Because the macro is not created, you obtain no processing advantage when you run the job with PREFETCH enabled.

Teradata DBMS Table Creation and the Critical PRIMARY Index

What determines which column is used for the PRIMARY index when a table is created?

A ExperiencedTeradata DBMS users know the following, but it is important to repeat. The Teradata DBMS uses a PRIMARY index, created as the table is created, as a hashing index to distribute table data across the server's available AMPs. Therefore, the PRIMARY index is critical for row distribution and efficient access to a DBMS table. "If you do not assign a PRIMARY index explicitly when you create a table, one is assigned automatically to the first column defined for the table" [20]. Consequently, when SAS issues a create table on your behalf -- and you do not specify a PRIMARY index with explicit SQL pass-through or the DB_CREATE_TABLE OPTS data set option -- implicitly you obtain a PRIMARY index on the first column.

Without intervention, creation of secondary tables generates default PRIMARY indexes. For example, with SAS you create a new secondary DBMS table using a table that was created with an explicit PRIMARY index as your source (you deliberately asserted a PRIMARY index on a column). When the new secondary table is created, your assertion of PRIMARY index does not persist; the new table acquires a default PRIMARY index on its first column.

Default PRIMARY indexes can occur whenever you create and load new tables (this means FASTLOAD operations are affected). An infrequent side effect can be out-of-disk-space error messages. The reason for the out-of-disk-space condition is subtle: column values for the default PRIMARY index skew the data distribution for the new table unequally among the AMPs, exhausting the disk space allotment of one or more AMPs. Users of the SAS/ACCESS Interface to Teradata, as with other application software, must be aware of Teradata's implicit PRIMARY index assignment.

[19] Usually, your default database matches your logon name.
[20] *Teradata RDBMS SQL Reference - Volume 1 Fundamentals*, page 2 through 25.

Teradata Mode versus ANSI Mode

What are the effects of setting the client session to ANSI mode instead of Teradata mode?

A The following table shows some differences. For the complete list, refer to the *Teradata RDBMS SQL Reference* manual.

	ANSI Mode	Teradata Mode
Transaction initiation	Always implicit	Implicit or explicit
Default for character comparisons	Case-specific	Not case-specific
Explicit COMMIT WORK statement required after every DDL (DBMS) statement	Yes	No
Default for CREATE TABLE statement	MULTISET (allows duplicate rows)	SET (does not allow duplicate rows)
Default for TRIM function	Trims leading/trailing blanks	Trims only trailing blanks

Can I open Teradata mode sessions using the SAS/ACCESS Interface to Teradata?

A Yes. Beginning with SAS 8.2, SAS provides limited support of Teradata mode using PROC SQL through the use of the explicit pass-through option MODE=Teradata. There is no further support of Teradata mode planned.

The following example shows the use of the MODE=Teradata option to obtain this mode's case-insensitive behavior. Note that because we turn on Teradata mode, the SQL COMMIT statements are not required because they are in ANSI mode.

```
/*Create and populate table in Teradata Mode (case-
insensitive)*/

proc sql;
connect to teradata (user=testuser pass=testpass
                     mode=teradata);
execute(create table casetest(x char(28)) ) by teradata;
execute(insert into casetest values('Case Insensitivity
Desired')) by teradata;
quit;

/*Query table in Teradata Mode (for case-insensitive match)*/

proc sql;
connect to teradata (user=testuser pass=testpass
                     mode=teradata);
select * from connection to teradata  (select * from casetest
where x='case insensitivity desired');
quit;
```

Can I obtain non-case-specific behavior even though the Teradata engine has set my session to ANSI mode?

A Yes. To mimic the non-case-specific functionality of BTEQ, which defaults to Teradata mode, you can CAST your WHERE clause arguments. Thus, in the simple example below, instead of using

```
where col='D_aw_bcall_200007DB'
```

use the following:

```
where col=cast('D_aw_bcall_200007DB' as not casespecific )
```

Use of CAST is a portable way to effect not case-specific functionality. That is, the CAST example statement will work in either mode, ANSI or Teradata. And, when coded into a native Teradata view, CAST works with any product that you use to access the Teradata DBMS, whether the product runs in Teradata mode or ANSI mode. For this reason, it is the smartest technique to use whenever possible. (You can also use CAST with the SAS/ACCESS Interface to Teradata explicit SQL.)

SQL Pass-Through

Can I determine whether the Teradata DBMS or SAS is performing the query?

A Yes. As mentioned earlier, the SASTRACE option, used with either the LIBNAME or SQL pass-through method, shows the SQL that the engine issues. When used with SQL pass-through, the option tells you which side, SAS or Teradata, performed the work.

```
option sastrace=',,,d'
sastraceloc=saslog no$stsuffix ;
```

Another PROC SQL option, _METHOD, generates output to the SAS log that identifies which query work actually passed to the Teradata DBMS. With this debug information, you can create faster, more efficient queries, that is, make the Teradata DBMS perform the subset processing and have SAS, via PROC SQL, merely display the result set returned. The _METHOD option can help you avoid a possible query pitfall: your query generates SQL that requests Teradata to perform full table scans; the WHERE clause and functions are not passed and therefore are not performed by the DBMS. The example below uses the _METHOD option. (Note that this option is available with explicit and implicit SQL pass-through processing.)

```
proc sql _method
select * from yourlib.table1;
```

Upsert Processing

Can I perform upsert processing without using a SAS DATA step and the MODIFY clause?

A Yes. But before explaining how, here is added technical detail. The SAS DATA step MODIFY clause reads an observation from the transaction data set; then uses dynamic WHERE processing to locate a matching observation in the master data set. The Teradata DBMS does not support this processing; consequently, the following example will not work when executed from the SAS/ACCESS Interface to Teradata or from the SAS/ACCESS Interface to ODBC, when run against the Teradata DBMS.

```
/*Example: DATA step with MODIFY*/

libname trlib teradata user=testuser password=testpass
reread_exposure=yes;

data trlib.master1;
do i=1 to 10;
x=1;
output;
end;
```

```
data extra;
do i=8 to 12; x=5;
output;
end;

data trlib.master1;
set extra;
modify trlib.master1(cntllev=rec dbkey=i ) keyÛkey;
if (_iorc_ = %sysrc(_sok)) then replace;
 else output;
run;
```

If you run the DATA step MODIFY code with the SAS/ACCESS Interface to Teradata, you obtain the following error message:

```
"ERROR: Teradata update: Parcel kind or ordering is invalid.
SQL statement was: USING (“i” FLOAT,”. Insert substitution
values are not shown.”
```

When the SAS/ACCESS Interface to ODBC executes the code, instead of returning a Teradata DBMS error message about unsupported functionality, it processes the MODIFY statement and returns incorrect results.

MODIFY Work-around: Example of Upsert Processing Using Two Teradata DBMS Tables

A work-around for the MODIFY problem that also provides upsert capability is to use explicit SQL pass-through. In the following example, customer records stored in a transaction table are matched against the customer master table. If a match is found, you want to update the existing master record. If no match is found, you want to append the transaction record (a new customer) to the master table.

```
 /*Create a simulated DBMS master table*/

libname trlib teradata user=testuser pass=testpass;
data trlib.master1;
  do i=1 to 10;
    x=1;
    output;
    end;

/*Create a simulated DBMS transaction table*/

data trlib.teratrans;
  do i=8 to 12;
    x=5;
    output;
    end;

/*Specify explicit SQL pass-through, via the EXECUTE statement,
to perform the upsert*/

proc sql;
connect to teradata (user=testuser pass=testpass);

/*Update common rows (where transaction record matches master
record)*/

execute (update master1 set x = teratrans.x where master1.i =
teratrans.i) by teradata;
```

```
/*Delete common rows from transaction table*/

execute (delete from teratrans where teratrans.i = master1.i) by
teradata;

/*Append remaining transaction rows (the new customers)*/

execute (insert into master1 select * from teratrans) by
teradata;

/******COMMIT is required because session is set to ANSI
mode******/

execute (commit) by teradata;
quit;

/*Use PROC PRINT to verify that the updates were successful*/

proc print data=trlib.master1;
run;
```

Glossary

The table below contains terms that users might find confusing or vague. Some are specific to the Teradata DBMS; others are general database or computer terms that are common to both SAS and Teradata. The source for a definition is at the top of the column. NCR definitions are extracts from NCR's Teradata DBMS documentation. (When not otherwise noted, the NCR definition is taken from "*Introduction to Teradata RDBMS.*") Definitions in the SAS column are either commentary or an extract from SAS documentation.

Term	NCR Definition	SAS Definition
AMP (Access Module Process)	An instance (virtual processor) of database management data (tables, rows, indices) with their associated data manipulation processes and their data context (Transaction in Progress table, lock information, disk access information).	
BTEQ (Basic Teradata Query)	A host-resident application program that enables a user to execute a series of Teradata SQL requests in either batch or interactive mode. BTEQ can read from or write to host data sets and use more than one Teradata session.	
BYNET	The dual interconnection network that allows high-speed communications between the nodes of an NCR System 4700, 5100M, and 5150.	
Clique	A logical group of nodes on an NCR system 5100M that shares access to disk storage.	
EXPLAIN Modifier	Reports a summary of the plan generated by the SQL query optimizer: the steps the Teradata RDBMS would perform to solve a request. The request itself is not processed. (Teradata RDBMS SQL Reference, Volume 6, V2R4, 3-92)	Similar in function, the option _METHOD generates query processing information to the SAS log.
FASTLOAD	Fast Data Load utility A program that loads empty tables on the Teradata RDBMS with data from a network-attached or channel-attached client. (Terabuilder Reference)	A SAS/ACCESS Interface to Teradata option that, when enabled, loads empty tables to the Teradata DBMS.
Full Table Scans	A full table scan is a retrieval mechanism in which all rows in a table are touched. These occur when you retrieve all rows from a table in a SELECT statement. (Teradata SQL Reference, Volume 1, V2R4.0, 2-36)	

Term	NCR Definition	SAS Definition
Join	In Teradata SQL, a select operation that combines information from two or more tables to produce a result.	Note: A SAS PROC SQL join is equivalent to a Teradata SQL join. It is *not the same* as a SAS DATA step MERGE statement. A MERGE statement combines the table rows differently.* Also, a PROC SQL join, unlike a MERGE statement, does not require: • Same-named columns in join expressions • Equality in join expressions. * Note: see SAS documentation for details.
Journal	JOURNAL provides data protection through system-generated before- and after-journals that contain the data that changed as a result of any insert, update, or delete. These journals are used either to restore a table or to reverse the changes that were made to a table. (Teradata SQL Reference, Volume 4, V2R4.0, 1-144)	
Macro (Teradata SQL Macro)	A macro consists of one or more statements that can be executed by performing a single statement. Each time the macro is performed, one or more rows of data can be returned. Performing a macro is similar to performing a multi-statement request.	SAS Macro Facility: A tool for extending and customizing the SAS System and for reducing the amount of text you must enter to do common tasks. The macro facility allows you to package small or large amounts of text into units that have names. From that point on, you can work with the names rather than with the text itself. When you use a macro facility name in a SAS program, the macro facility generates SAS statements and commands as needed. The rest of the SAS System receives these statements and commands and uses them in the same way it uses statements or commands that you enter.
MPP System (Massively Parallel Processing System)	Multiple SMP nodes that are connected by the BYNET to form a larger configuration.	
Skewed queries		A query condition(s) that causes the DBMS processing to be limited to a number of AMPs, rather than to be distributed among all available AMPs.

World Headquarters
and SAS Americas
SAS Campus Drive
Cary, NC 27513 USA
Tel: (919) 677 8000
Fax: (919) 677 4444
U.S. & Canada sales:
(800) 727 0025

SAS International
PO Box 10 53 40
Neuenheimer Landstr. 28-30
D-69043 Heidelberg, Germany
Tel: (49) 6221 4160
Fax: (49) 6221 474850

www.sas.com

 260871US.1103