SAS/ACCESS® Interface to Teradata White Paper

December 15, 2000

By Donna R. Adler and Douglas J. Sedlak

About the White Paper

Audience

SAS® and Teradata® DBMS users who need fast, seamless access to Teradata DBMS tables from the SAS System.

Purpose

We offer an umbrella view of the SAS/ACCESS® Interface to Teradata, relating the interface to adjunct SAS products, to acquaint you with the product. Then, we showcase its features and capabilities. (Hereafter, SAS/ACCESS Interface to Teradata is referred to as S/A Interface to Teradata.)

In detailing[1] how S/A Interface to Teradata operates, the paper provides general and technical product information. The first half focuses on methods that the interface furnishes to access Teradata DBMS data. The second half discusses the functionality and performance of the interface when compared to other Teradata data access alternatives, information intended to guide you in matching software or a product feature to the processing situation.

Throughout we offer tips on how to optimize a SAS session or job so that the processing is performed where appropriate: by the SAS System or by the Teradata DBMS. Our objective is SAS-centered processing; that is, to have the SAS System *manage but not necessarily perform the work***.** SAS-centered processing, given good performance, is preferable to separating the work, performing some in SAS and the rest outside -- for example, pre-processing or post-processing with Teradata DBMS utilities.

Additionally, the paper answers technical FAQ's received from S/A Interface to Teradata customers. The replies emphasize 'best use' of the S/A Interface to Teradata product.

Content and Organization

Since our audience is wide, the content can range from rudimentary to complex. Sections dense with technical detail can be time-consuming to read. Therefore, we introduce each of the main ideas as topics. Within a topic you may find a technical discussion and/or code examples. And, at the end of a main idea there may be a summary with a small graphic reminder (shown left) to highlight an important point. The paper's organization and Table of Contents permit rapid skimming of topics of interest.

[1] Our intent is not to repeat SAS/ACCESS Interface to Teradata installation instructions or User Guide documentation. If you need this information, see "What other documentation is available for a SAS or Teradata DBMS end user?" on page 29.

Table of Contents

What is 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.

And, your site has enthusiastic SAS users who want to use SAS products to mine, warehouse, and analyze the DBMS data, thereby acquiring the SAS System's arsenal of 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 SAS/ACCESS Interface to Teradata (S/A 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 or the SAS Enterprise Guide software.

At the heart of the S/A Interface to Teradata product is engine technology, a SAS System mechanism that enables SAS users to read or write data directly in a specific data format -- in this case the Teradata DBMS format.

How Does S/A Interface to Teradata Relate to Base SAS and Other SAS Products?

The product summaries that follow are thumbnail sketches, deliberately skeletal to show the relationship between S/A Interface to Teradata and adjunct SAS products. (In the summary, we mention whether an adjunct product uses S/A Interface to Teradata.)

Base SAS â

To access the Teradata DBMS you must install base SAS and S/A Interface to Teradata, along with NCR's Teradata CLIv2 libraries, on the client machine. You then invoke the Teradata engine -- in a SAS session or job -- by specifying the name of the Teradata engine, TERADATA, in a SAS LIBNAME statement. Alternatively, you can invoke the engine using a PROC SQL statement. (See "How Does S/A Interface to Teradata Work: Methods to Access a Teradata DBMS" on page 10 for more information.)

By extension the LIBNAME statement method furnishes all functionality of base SAS. Thus, the LIBNAME method permits you to surface and manage Teradata DBMS tables along with SAS data sets. SAS programmers find the LiBNAME method attractive because usually they do not have to make changes to existing PROC or DATA steps to run them against the Teradata DBMS successfully.

S/A Interface to Teradata supplements base SAS functionality with Teradata-specific capabilities allowing SAS users to supersede some default behaviors of the Teradata DBMS. For example, the interface enables users to create Teradata DBMS columns that do not accept NULL values (using SAS data set options.) Or, to override Teradata's default row-locking with S/A Interface to Teradata locking options. For inexperienced SAS users, the Teradata engine offers, again by extension, non-programming graphical tools such as the SAS Explorer and the Query window. The SAS Explorer, a familiar split window that displays libraries and the library contents (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 having to know SQL.

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

In summary: S/A Interface to Teradata allows SAS users to tap the SAS System's highly-touted analytic tools and data exploitation capabilities for Teradata tables without requiring that they learn much more than several Teradata engine options. Or, it permits novice SAS users, unfamiliar with SAS programming, to operate on Teradata DBMS tables merely by pointing and clicking from the SAS Explorer or Query windows. (Potentially, transparent access using the SAS Explorer allows users to retrieve terabytes of table data. Thus, the end user or the administrator must know the data and restrict access appropriately to Teradata DBMS tables and processing.)

SAS/ACCESS â Interface to ODBC

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

What then is the difference between the two engines? One difference is the S/A Interface to Teradata engine communicates directly with the Teradata DBMS, calling the Teradata CLIv2 interface. In contrast, the S/A 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.

Historically, (prior to SAS Version 8) SAS users could access Teradata DBMS data only by using S/A Interface to ODBC; the S/A Interface to Teradata native engine was unavailable before then. Since both products can access Teradata DBMS data, users are frequently confused about the products. To clear up the confusion, we compare the Teradata and SAS/ODBC engines, identifying some functional and performance differences between the products (see "S/A Interface to ODBC" on page 25).

SAS/Warehouse Administrator â

SAS/Warehouse Administrator (SAS/WA) is the tool of choice for ETL (extract, transform, and load) of DBMS and other file data into a logical data warehouse. Designed for IT professionals, who are responsible for creating and managing data warehouse/data mart processes, SAS/WA is customizable and provides a single point of control for response to the changing requirements of a business community.

Increasingly, end users understand the benefits of building business intelligence systems on the solid foundation of a data warehouse. But, writing programs that perform the tedious chores of ETL to deliver a repository suitable for business intelligence is time-consuming and can overextend the most productive IT department. The SAS/WA graphical interface simplifies visualization, navigation and maintenance of a data warehouse for IT professionals and eliminates much of the coding required not just for building but for managing the warehouse.

In brief, SAS/WA offers adaptability and manageability by:

- Integrating ETL tools;
- Providing a metadata framework for effective warehouse management;
- Facilitating business subject definitions and uniform business rules;
- Scheduling warehouse maintenance processes and integrating them with decision-support tools to exploit a data warehouse effectively;
- Leveraging the core strengths of SAS software to deliver a data warehouse faster.

Specifically, for a Teradata DBMS, SAS/W/A employs Teradata's FastLoad and MultiLoad Utilities as well as S/A Interface to Teradata to extract and load (refresh and append) Teradata tables. (To learn more about options that SAS users have for rapidly loading and refreshing Teradata DBMS table data in the warehouse, see "Alternatives for Loading/Refreshing Teradata DBMS Tables" on page 24.)

Whether your data warehouse uses the Teradata DBMS alone, or in combination with other DBMSs, SAS/WA -- employing native (DBMS) utilities and SAS/ACCESS products -- is designed for all DBMSs supported by the SAS System.

Enterprise Guide â

Enterprise Guide (EG) provides a point-and-click interface for connection to SAS System servers and automates the task of performing SAS data processing and analytical tasks. A true SAS thin client for the Microsoft® Windows environment EG 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.

EG, a separate Windows application does not require installation of any other SAS software on the client machine. Assuming that S/A Interface to Teradata is installed on the SAS server (see "Network Overview: Teradata DBMS (Server) and Clients" on page 9), the EG client user can operate on Teradata DBMS tables as if they were SAS data sets without knowing SAS programming.

SAS/SPD Server â

SAS/SPD Server is the SAS solution for implementing data marts that are gigabytes in size. SAS/SPD Server, a highly parallel data server, stores and retrieves SAS data and runs on most UNIX SMP and Windows platforms.

Frequently, a data warehouse solution centers on a Teradata DBMS and includes Oracle® or DB2® databases. In these scenarios, SAS/SPD Server can replace Oracle or DB2 for SAS applications -- a substitution that can also increase performance.

SAS/SPD Server enhances performance by implementing parallel algorithms for many intensive data processing operations such as table scans, sorting, indexed WHERE clause evaluations, SELECTs with aggregate functions (that is, GROUP BY, AVG, etc.), table loads/copies, and index creation. Beyond boosting performance, the SAS/SPD Server furnishes row/column security, utilities for incremental table/file backups and restores, file encryption and compression, along with ODBC, JDBC, & htmSQL support.

Which S/A Interface to Teradata Clients are Supported?

- OS/390 (MVS)
- Solaris SPARC (32 bit)
- Microsoft Windows NT
- Microsoft Windows 9X
- Microsoft Windows 2000
- UNIX MP-RAS

Which Teradata DBMS Server Releases are Supported?

Teradata DBMS Servers:

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

S/A Interface to Teradata supports Teradata DBMS server releases V2R2, V2R3, and V2R4. All releases of S/A Interface to Teradata, version 8.0 and higher, run interchangeably with these Teradata DBMS versions. (Support for Teradata's TIME and TIMESTAMP data types is available with S/A Interface to Teradata version 8.1 and higher.)

Which SAS Products are Required to Access the Teradata DBMS?

- Base SAS
- SAS/ACCESS Interface to Teradata

Where Must the Required SAS Products Reside?

To access the Teradata DBMS, you must install the required SAS products on the same machine where the Teradata's TUF[2] client software, specifically the CLIv2 libraries, is installed. We refer to this machine as a SAS server and a DBMS client machine. (See "Network Overview: Teradata DBMS (Server) and Clients" on page 9 for several examples of SAS servers.)

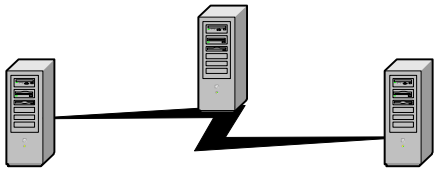
S/A Interface to Teradata contacts the Teradata DBMS server by calling the Teradata CLIv2 library; the library in turn relies on standard Teradata-supplied middle ware.

What Other Software is Required to Set-Up an OS/390 Client?

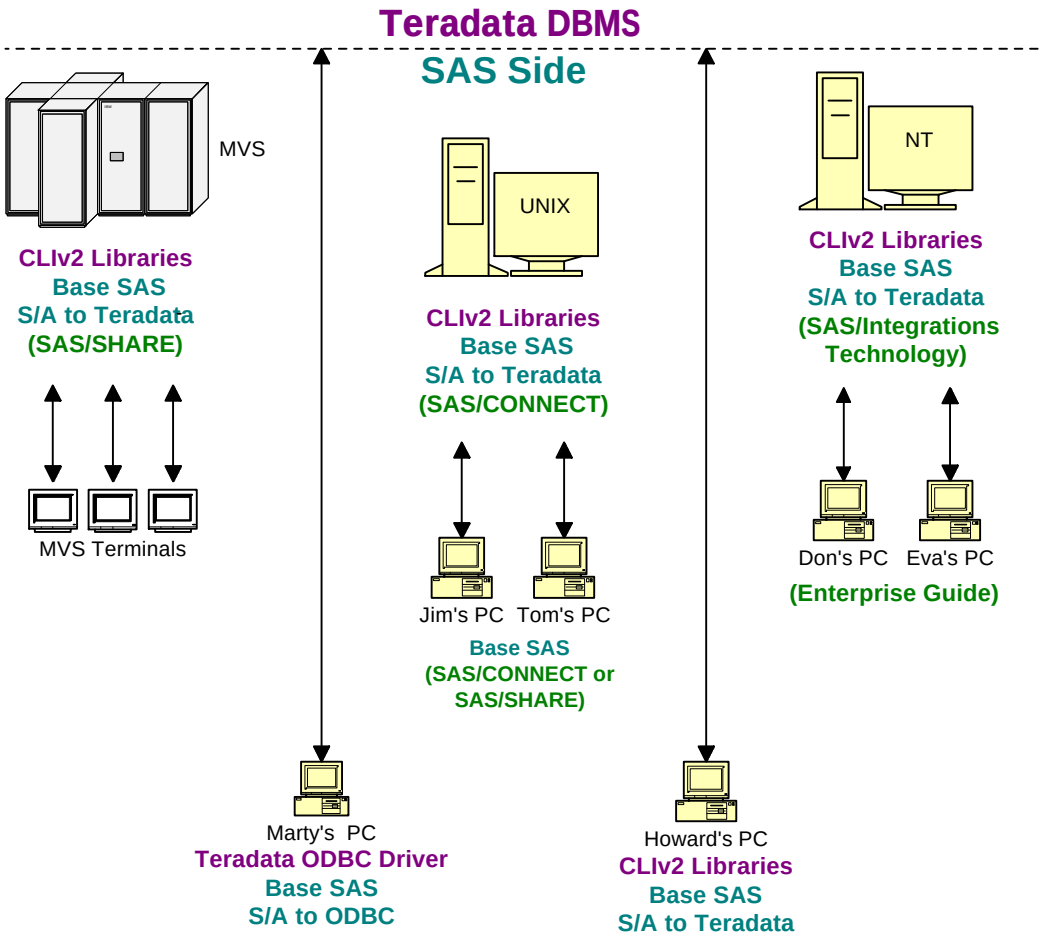
No other software is required. You need only the CLIv2 libraries, specifically the APPLOAD load library, to be installed. This library is supplied standard with the TUF toolkit. (Here at SAS, we use TUF 5.6.) Installation of S/A Interface to Teradata for MVS is then very straightforward. The important thing is to make sure that you already have MVS-to-Teradata server connectivity (that is, you can run BTEQ and other native Teradata DBMS utilities from MVS). If you can run BTEQ, S/A Interface to Teradata should work without a hitch. If you do not have the Teradata software on your MVS system, contact NCR.

[2] TUF or Teradata Utilities Foundation Software is the 'unified' name for Teradata's client software. The TUF toolkit includes all the Teradata client software that you need to run S/A Interface to Teradata -- the CLI libraries and TDP/MOSI middleware.

Network Overview: Teradata DBMS (Server) and Clients



MPP Cluster of MP-RAS UNIX SMP Boxes



The Teradata DBMS (server) can run either on an MPP cluster of MP-RAS UNIX 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.[3] MVS, Windows NT, and Solaris appear more popular than MP-RAS for end user applications, possibly because their operating systems do not have the 2-gigabyte file size limit that MP-RAS imposes.

[3] Here S/A to Teradata is licensed on a single server node with multiple SAS clients accessing the Teradata DBMS through that SAS server. Optional SAS products that can be installed on the SAS server/PC clients are shown in tandem in parentheses. **For simplicity, the diagram does not show all possible SAS configurations for either SAS servers or PC clients**.

How Does S/A Interface to Teradata Work: Methods to Access a Teradata DBMS

S/A 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, the SAS LIBNAME Engine method, is discussed first.

Because SQL is more familiar to Teradata DBMS users, our discussions tend to emphasize the SQL pass-through method. Thus, you will notice that we elaborate on explicit and implicit SQL pass-through at length and provide SQL examples. But, keep in mind, the interoperability furnished by the LIBNAME method is an extremely powerful feature of S/A Interface to Teradata product. When you use the SAS LIBNAME engine method, SAS procedures and SAS DATA steps seamlessly inter-operate on Teradata tables. If you need more examples of the LIBNAME method than we offer in this paper, you will find many in the S/A Interface to Teradata Chapter in SAS online documentation (see "What other documentation is available for a SAS or Teradata DBMS end user?" on page 29.)

The SAS LIBNAME Engine Method

With this method you specify the Teradata engine name, TERADATA, along with connection and other engine options, in a SAS LIBNAME statement. Once invoked, S/A Interface to Teradata generates SQL statements equivalent to the SAS requests and submits the SQL that is generated to the Teradata DBMS on your behalf.

Example of a SAS LIBNAME Statement

```
/* 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 S/A Interface to Teradata …	PROC SQL does …	DBMS Response …
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, fails the request.
IMPLICIT[4]	specifies SAS SQL with PROC SQL.	Converts SAS SQL to Teradata-specific SQL on your behalf and passes the SQL to the DBMS; but 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

With this method you invoke the SAS Procedure, PROC SQL, and specify a statement that requests connection to the Teradata server, followed by SQL requests. The PROC 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 PROC passes to the DBMS, this method is known as explicit pass-through.

Implicit Pass-Through

Alternatively -- similar to the LIBNAME method -- PROC SQL can generate SQL statements on your behalf, behind-the-scenes processing 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 Teradata DBMS. Implicit pass-through is subject to rules; we discuss the rules in the section, "Understanding the Rules of Implicit Pass-Through" on page 11.

Understanding the Rules of Implicit Pass-Through

The passing of a query to the Teradata DBMS via implicit pass-through is analogous to a horse that must pass hurdles to win a race. One hurdle: the query must have basic characteristics that make it eligible for implicit pass-through. Another hurdle: the query must not contain any elements that make PROC SQL disqualify it.

[4] Implicit pass-through is a series of performance enhancements to SAS PROC SQL For additional information on the subject, see "What other documentation is available for a SAS or Teradata DBMS end user?" on page 29.

Basic Eligibility

For basic eligibility any query or query part, must:

- Refer to a single S/A LIBNAME;
- Be legal according to the ANSI SQL2 standard
- Be recognized by PROC SQL.

Effect of DBMS SQL2 Conformance

The DBMS can reject a perfectly legal SQL2 query passed by SAS. The reason: the DBMS may not support all levels (3) of SQL2 conformance. Describing a legal SQL2 query for the Teradata DBMS is beyond the scope of this paper. Instead, we list SQL keywords that trigger PROC SQL to pass the query, or query part, to the Teradata DBMS and identify elements within the query that make PROC SQL disqualify it.

SQL Keywords that Trigger Implicit Pass-Through

The following SQL keywords[5] 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 the Query for Implicit Pass-Through

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

- CONNECTION TO
- Re-merging
- 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 the SAS data function list (see "Data Functions that are Passed to the Teradata DBMS" on page 13).

Thus, PROC SQL does not pass to Teradata a query that contains a WHERE clause specifying an unsupported SAS function, for example, the FUZZ[6] function. Instead, the PROC returns the query to SAS to process.

[5] This list is continually expanding. To obtain a complete list for Version 9 and higher consult your SAS documentation.

[6] The FUZZ: function returns the nearest integer if the argument is within 1E-12.

Data Functions that are Passed to the Teradata DBMS

Version 8.2 and higher[7] of S/A Interface to Teradata passes the following data functions to the Teradata DBMS for processing:

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

Example: Implicit Pass-Through of SAS TODAY Function

SAS WHERE Clause

```
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, version 8.2 and higher, generates the following Teradata-specific SQL for the preceding SAS WHERE clause code:

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

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

/* Setup: Create a Teradata DBMS table for the example */

```
libname trlib teradata user=testuser pw=testpass;
data trlib.customr16;
   input custname $ 1-10 custnum custcity $ 22-36;
   cards;
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;
```

[7] For updates to the list for Version 9.0 and higher, check your SAS documentation.

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 marked in bold. Notice, with explicit pass-through, the SQL that is submitted to Teradata is exactly as specified. In contrast, with implicit pass-through, S/A Interface to Teradata prepares the SQL that is submitted on your behalf.

In the performance section, we discuss the LIBNAME and SQL pass-through connection methods (see "How Does S/A Interface to Teradata Work: Methods to Access a Teradata DBMS" on page 10) more fully, detailing how implicit SQL pass-through can come into play. For now, just remember that you can use one or both connection methods in a single job/session to enable any feature or function that the method supports.

Both S/A 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 MVS 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. You can see the SQL generated using the following option in your SAS session or program (the preceding example uses the 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 also 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 S/A Interface to Teradata furnishes excellent read processing performance. Although performance of non-read table operations varies, there are ways to enhance the default performance. In the next section, we present performance-enhancing alternatives for the following operations:

- loads of empty tables
- appends[8] to tables already containing rows
- row updates
- row deletes
- upserts[9]

In the paper's introduction we stated that our goal, given good performance, is to have the SAS System manage the processing work. Therefore, the performance alternatives that we suggest frequently utilize S/A FastLoad, that is, fastloading Teradata table data from SAS. When you enable S/A FastLoad by specifying FASTLOAD=YES, S/A Fastload directly fastloads data to Teradata tables.

Although S/A Fastload does not use the Teradata FastLoad Utility, it shares some characteristics in common with that Utility. For example, like the Teradata Fastload Utility, it drops duplicate rows when fastloading tables. Additionally, when fastloading data to a temporary Teradata table (as we show in some examples), the temporary table will require additional DBMS space.

[8] For a definition of an **append** operation, see “Table Append Operation” on page 17.
[9] For a definition of an **upsert** operation, see “Table Upsert Operation” on page 17.

Optimizing Large-Scale DBMS Table Operations: Code Examples

S/A 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. In this section we show you how to optimize large-scale operations, as well as how to perform an upsert operation from SAS. We also furnish notes about related Teradata DBMS utilities. (To distinguish Teradata's native utilities from SAS software, the Teradata utility notes are in purple and italicized.)

Example Setup:
Our examples refer to two tables with the identical column layout. TeraMaster, the master table into which we want to insert, update, or delete rows, is stored in the Teradata DBMS. SASTrans, a SAS data set (SAS table) which supplies the insert and update transactions, is stored local to your client SAS session. In our examples, the TeraMaster table is located in the PLAYAREA database.

And, we assume 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** is insertion of rows into an empty table. By default S/A Interface to Teradata inserts a single row at a time. However, when there are many rows, this default processing is slow (expensive). The performance-enhancing alternative is to enable S/A FastLoad and have SAS perform fastloading.

Fastloading delivers very high performance with only two constraints: duplicate rows are dropped and error detection/correction is weak. (For complete details, see SAS and Teradata documentation on FastLoad and "How does FastLoad affect writes to Teradata DBMS tables?" on page 31.)

Load of Empty TeraMaster Table

Default Example: Load without S/A FastLoad

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

Enhanced-Performance Example: Load with S/A 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 tables. Performance of S/A FastLoad is equivalent to the Teradata FastLoad Utility.

Table Append Operation

A table **append** is insertion of rows into a non-empty table, that is, a table that already contains some rows. Since we have many rows to insert, we want to circumvent a row-at-a-time insert again. So we use S/A FastLoad to load an intermediary Teradata table. Next we pass explicit SQL statements to Teradata to append data from that intermediary table. Finally, we delete the intermediary table upon completion. Keep in mind that duplicate rows in the original "append set" will be removed as a result of the FastLoad step and thus are 'lost' to the target Teradata DBMS table. (See "How does FastLoad affect writes to Teradata DBMS tables?" on page 31.)

Append of Non-empty TeraMaster Table

Default Example: Append without S/A FastLoad

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

Enhanced Performance Example: Two-Step Append with S/A 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;
```

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 our example. It also 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. SAS/ACCESS products support upsert capability through the SAS DATA step MODIFY clause. But, S/A Interface to Teradata does not support the DATA step MODIFY clause. (For technical details as well as an example of a one-step insert operation, see "Can I perform upsert processing without using a SAS DATA step and the MODIFY clause?" on page 34.)

In the example that follows, we use explicit SQL to perform an upsert, fastloading the SAS intermediary table. Again, note that use of FastLoad for the intermediary table will drop duplicate rows.

High Performance Example: Multi-Step Upsert Processing using S/A FastLoad

```
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: Update common rows (transaction record matches master record)*/
execute (update TeraMaster set LastPurchase =
Intermediary.LastPurchase where TeraMaster.CustId =
Intermediary.CustId) by teradata;
/* Step3: Delete common rows from transaction table. */
execute (delete from Intermediary where Intermediary.CustId =
TeraMaster.CustId) by teradata;
/* Step4: 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 also the corresponding Teradata DBMS mechanism for high-volume upsert operations. . Its performance is comparable the multi-step process shown above. It also 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. S/A Interface to Teradata updates and deletes one row at a time -- this is very inefficient and time consuming when many rows must be updated or deleted. The 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 the identical SQL shown in the enhanced-performance example with the Teradata BTEQ Utility, thereby obtaining the same functionality. The performance of the code is identical to S/A Interface to Teradata.

Assessing Reads of Gigabytes or Terabytes of Teradata DBMS Data into SAS

S/A Interface to Teradata is designed to read Teradata rows as fast as possible; no further tuning is required. Even for high volume reads, S/A Interface to Teradata will outperform most products -- with the exception of the Teradata FastExport Utility. The Teradata FastExport utility is a higher-performance read alternative, deriving its additional speed by opening multiple channels to the Teradata DBMS. Most products, S/A Interface to Teradata included, use a single channel. If you routinely read gigabytes or terabytes of data from Teradata DBMS tables into SAS -- and time is critical -- you might consider using FastExport.

If you decide to use the FastExport Utility, you must write a FastExport script. When invoked, the script submits your query to the Teradata DBMS and fetches data on multiple channels. The script outputs data to a 'flat file' on the client machine (that is, the machine where FastExport and SAS run). To read the flat file into SAS, you must write customized SAS code. To obtain a FastExport example -- beyond the scope of this paper -- see Teradata DBMS documentation.

We mention use of FastExport, in conjunction with SAS, to accentuate that this alternative requires customization. And to point out that it is a complex, multi-step process that conflicts with the seamless integration furnished by S/A Interface to Teradata. But, if performance is critical and the volume of data that you must pipe from Teradata into SAS is huge, you may need to consider alternatives.

Enhancing Performance: Ensuring Teradata DBMS Server-Processing

Situations That Cause Teradata DBMS Processing to Occur

Since processing is greatly enhanced when performed by the Teradata DBMS, SAS users are eager to know how to move work to the Teradata DBMS side. Below we recap two situations discussed previously that pass work to the DBMS (see, "The SQL Pass-Through Method" on page11):

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 contain operations such as joins, distinct processing, and SQL-defined aggregate functions.

There is an unmentioned situation that also triggers DBMS processing:

WHERE Clause Processing

SAS passes a WHERE clause to the DBMS if the WHERE clause can be processed by the DBMS.

Since most users understand WHERE clause processing, we expand discussion on SQL pass-through situations only.

Facilitating Performance: The Best Connection Method for the Situation

SQL Pass-Through Method

Explicit Pass-Through: Facilitates User-Written Teradata-Specific SQL

Using SQL syntax to effect DBMS processing can be an advantage to a Teradata DBMS user. In contrast, an experienced SAS System user, more familiar with SAS syntax, can find the use of SQL a disadvantage.

Beyond an ease-of-use consideration, explicit pass-through is the only connection method that allows you to code Teradata-specific SQL. Thus, explicit SQL pass-through offers the most flexibility. Specifying the correct Teradata-specific SQL against the Teradata DBMS, you can effect almost any functionality that you implement with Teradata's native tools. One simple example: you can set or retrieve DBMS table column constraints with SQL pass-through.

In contrast, using the S/A Interface to Teradata LIBNAME method, the Teradata engine generates SQL for you -- you cannot modify the SQL that the engine submits to the Teradata DBMS.

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

Implicit pass-through[10] passes a growing number of queries and joins to the Teradata DBMS instead of having SAS process them on the client. As of Release 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.

[10] Version 7 and higher of the SAS System.

Under the Covers: An Example of Implicit Pass-Through

Assume your site has two large DBMS tables named TABLE1 and TABLE2. The tables have the DEPTNO column in common. You want 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 (where the SAS *libref* references a database). Once detected, the Teradata engine 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 tables.

In our example, the query join follows the implicit pass-through rules that we discussed earlier (see "Understanding the Rules of Implicit Pass-Through" on page 11), allowing SAS PROC SQL to pass the request to Teradata. What happens if the query did not comply with the rules? PROC SQL simply passes the request back to SAS for processing.

Obviously, it's far more efficient for 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 a site can incur when performing work on the SAS side. For example, a site can have an MVS machine with attendant billing fees. Thus, when SAS, which resides on the MVS client machine, performs the processing instead of Teradata, the site incurs real dollar costs.

SAS Options to Manage Implicit Pass-through

Since implicit pass-through is subject to several variables, how do you know if it occurred? SAS provides two options to help you 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 you how your query was processed; this output also tells you whether implicit pass-through occurred.

In summary, you can obtain the best performance for queries and joins when you use explicit SQL pass-through, simply because you guarantee that the Teradata DBMS will perform the query or join processing. However, implicit pass-through furnishes a second opportunity to have SAS generate Teradata-specific SQL.

Understanding implicit pass-through and managing the facility can help you to take better advantage of the transparent processing. Since it is automatic, it relieves you from having to know or learn Teradata-specific SQL to effect DBMS requests. But, implicit pass-through code has an added benefit -- it is portable. As a consequence, you can use and port implicit pass-through code from one SAS ACCESS engine to another.

In brief: rely on implicit pass-through whenever possible and reserve use of explicit SQL pass-through for processing situations that require your intervention to ensure optimal performance.

Explicit Pass-Through: Enables Basic Engine Options

Explicit SQL pass-through supports S/A Interface to Teradata's connection and database options. These options include:

- USER=
- PASSWORD=
- TDPID=[11]
- ACCOUNT=
- DATABASE=[12]

Thus, if you want to employ functionality that is furnished by 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, this method offers the product's most sophisticated features including:

- PreFetch

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

  - PREFETCH=

- S/A 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

[11] The alias for this option is SERVER=.

[12] The alias for this option is SCHEMA=.

- S/A FastLoad: A FastLoad Capability Without User Scripts

  S/A Interface to Teradata supports the fastloading of data to Teradata DBMS tables with the option:

  - FASTLOAD= (alias BULKLOAD=)[13]

Measuring the Performance of the SQL Generated

Earlier, when discussing Teradata engine connection methods, we pointed out that the LIBNAME engine method generates SQL for you. In this section, we broadly describe that SQL and identify SAS options that enable you to view the SQL that is generated, as well as the processing time. This information will allow you to benchmark and fine-tune processing performance.

A Broad Overview of the SQL that the Engine Generates

A DBMS table and a traditional SAS data set have many 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 products. This portable or shared code translates DBMS operations such as 'open', 'get', 'put', 'delete', 'does table exist', etc. into the required, DBMS-specific SQL. Since joins are critical to performance, SAS employs a separate code layer to optimize query joins, attempting whenever possible to pass the join processing to the DBMS. In brief the job of S/A Interface to Teradata -- similar to any SAS/ACCESS engine -- is to emit correct and optimized DBMS-specific SQL.

Capturing SQL and Timer Information to the SAS Log

In version 8.2 and higher, the following option statement logs the SQL statements submitted to Teradata and the time that Teradata took to perform the request. The timer statistics tell you where time is spent during a processing operation, helping you isolate a performance bottleneck. (In essence, you learn 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 to select from when loading and refreshing DBMS table data. The following table characterizes some of the alternatives.

[13] You can enable the FASTLOAD= option globally as a LIBNAME option, allowing 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. Note: 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
S/A FastLoad	SAS	• Very fast • Executes from within SAS • Must be enabled (by default, this option is off) • Populates empty Teradata DBMS tables • Permits *'closed loop'* processing[14]
SAS/WA Loader Add-In	SAS	• Requires the SAS/WA product • Populates empty SAS/WA 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 • Executes outside of SAS • Appends data to existing Teradata DBMS tables
S/A Interface to Teradata Two Step Append Using S/A FastLoad	SAS	• Fast • Executes from within SAS • Appends data to existing Teradata DBMS tables • Uses S/A Interface to Teradata's FastLoad in conjunction with explicit pass-through SQL
SAS/WA Loader Add-In	SAS	• Requires the SAS/WA product • Appends data to existing SAS/WA Teradata DBMS tables • Generates scripts that invoke the Teradata MultiLoad Utility

[14] **Closed Loop**: Here we mean that 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 Performance and Functionality of S/A Interface to Teradata to:

S/A Interface to ODBC

Functionality

Capability	S/A Interface to Teradata	S/A 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; you can select Teradata Mode and override the default. S/A Interface to Teradata runs in ANSI mode, you can specify Teradata mode only when using SQL explicit pass-through (see "Can I open Teradata Mode sessions using S/A Interface to Teradata?" on page 33.)

** SAS/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 using S/A 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 you run the product from;
- the version -- older or newer -- of the Teradata-supplied ODBC driver that you use;
- the hardware and network configurations that you set 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.	With various Teradata ODBC drivers, S/A Interface to ODBC takes 30% or more time than S/A Interface to Teradata.
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 S/A FastLoad[15] is enabled, S/A Interface to Teradata is several orders of magnitude faster.

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

Teradata BTEQ

Functionality

Capability	S/A 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. S/A Interface to Teradata defaults to ANSI mode (see "Can I open Teradata Mode sessions using S/A Interface to Teradata?" on page 33 to override the default).

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

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

You do not need COMMITs when using implicit SQL or the LIBNAME engine methods because in these situations the Teradata engine issues the COMMIT on your behalf. But, when you write explicit SQL, you must issue an explicit COMMIT statement along with any SQL request that modifies the database (SQL statements that create, update, modify, and drop Teradata tables.) The COMMIT statement is unnecessary for read requests (SELECTs). In the explicit SQL examples that follow, we embed comments showing you when and where an explicit COMMIT statement is required.

ANSI Semantics Mode: SQL Examples That Show Required COMMIT Statements

Example1

```
 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;
```

Example 2

```
 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;
```

Teradata FastLoad Utility

Functionality

Capability	S/A FastLoad	Teradata FastLoad Utility
Employs parallel processing	Y	Y
Requires scripts to execute	N	Y
Eliminates duplicate records	Y	Y
Reduces error checking	Y	Y
Writes a single entry to the Teradata transaction log (if transaction logging is enabled)	Y	Y

Performance

The performance of high-volume inserts into Teradata DBMS tables by S/A FastLoad is equivalent to that of the Teradata FastLoad Utility.

FAQ's (grouped by subject)

General

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

A S/A Interface to Teradata requires that the CLIv2 libraries, part of the TUF Toolkit, be installed on the client machine where base SAS is installed. If you have installed the Teradata CLIv2 libraries on the client machine and the native Teradata BTEQ facility functions correctly, S/A 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 access a Teradata DBMS server?

A Again, if you can access Teradata already from your client machine using Teradata utilities such as BTEQ, you do not need to make any other system changes to use S/A Interface to Teradata. If you cannot access the Teradata DBMS, then read further to understand connectivity between your S/A Interface to Teradata client and the Teradata server.

Simple Scenario: A Single Teradata DBMS server

Most Teradata shops run only a single Teradata server which houses all their Teradata tables. In this scenario, PC and UNIX client machines typically have set up a single entry, DBCCOP1, in their HOSTS file, that tells 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.)

Normally, on UNIX systems the HOSTS file is located in /etc/hosts; on Windows 95, 98, and Millennium systems in c:\windows; and on Windows NT and 2000 systems, in c:\winnt\system32\drivers\etc\Hosts.

HOSTS Example File for a Single Teradata DBMS Server

```
11.25.20.34     dbccop1       # default dbcname
```

LIBNAME Statement Examples to Test a Client Connection

When the HOSTS file of a PC and UNIX client has the above entry (see preceding example), BTEQ should work. Given a valid username, password, and database, S/A Interface to Teradata should also connect to the Teradata DBMS. To test a S/A Interface to Teradata connection, bring up SAS and issue the following LIBNAME statement (you can omit the TDPID=[16] option if the default server name, DBCCOP1, is set up in the HOSTS file):

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

OS/390 (MVS) client machines set up a single TDP task, typically called TDP0, that addresses the Teradata server. Under OS/390, once your administrator has started TDP0, S/A Interface to Teradata should connect to the Teradata DBMS server. To test an MVS 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):

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

Complex Scenario: Multiple Teradata DBMS Servers

Your site might have more than one Teradata DBMS server running. For example, in upgrading the Teradata server version, you can have a production system running day-to-day operations and a test system validating the latest Teradata release that you just received. 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, providing each server's name and IP address.

HOSTS Example File for Multiple Teradata DBMS Servers

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

LIBNAME Statement Examples Using the TDPID= Option to Connect to a Specific Teradata Server

On PC and UNIX systems, 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=test;
```

On OS/390 (MVS) 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;
```

[16] Or, the alias for the option,SERVER=.

Below is more **background** on connectivity than we presented above.

S/A Interface to Teradata uses a database connection name (*dbcname*), in conjunction with the connection option TDPID= to access a Teradata database. The content of the *dbcname* depends on how the client machine is attached to a Teradata DBMS server, that is, whether the machine is network-attached (PC and UNIX) or channel-attached (MVS).

Network Attached Systems (PC and UNIX)

You must make a *dbcname* (database connection name) entry in your client HOSTS file for every database server, thereby providing Teradata with an IP address. Each *dbcname* must have the suffix COPx, where x is a number. If you make a single entry, x must be 1. The default *dbcname* that S/A Interface to Teradata expects is DBCCOP1. If you make a DBCCOP1 entry in your HOSTS file—and run only one Teradata server—you do not need thereafter 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 can override the default *dbcname*, DBCCOP1, by specifying TDPID= and the appropriate dbcname for the server you want to connect to. Note: when making multiple entries for the same dbcname (a name identical up to the number), x must begin with 1 and increment sequentially.

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 @. By default, the Teradata client software for MVS is installed with a subsystem name of TDP0. 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.

And, for more examples, and a complete discussion of TDPID=, see your SAS Online Documentation.

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

A Additional sources below are grouped by subject:

S/A Interface to Teradata

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

SAS Online Doc is shipped with the S/A Interface to Teradata product; it is updated with each new release of the product.

Implicit Pass-Through

- SUGI White paper 51, "Performance Enhancements to PROC SQL in Version 7 of the SAS® System"
  by Lewis Church, Jr., SAS Institute
  http://www.sas.com/usergroups/sugi/sugi24/sipapers/p51-24.pdf

We have borrowed liberally from this paper. Although the content is dated, it gives the history of implicit pass-through and details use of the PROC SQL _**METHOD** option.

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

This paper describes a few processing situations in which you can obtain different results when the WHERE processing is performed by the DBMS instead of SAS.

Teradata DBMS

http://www.info.ncr.com

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.

Data Types

Where can I learn more about data types?

A The "*Data Types*" section of the S/A Interface to Teradata Chapter, mentioned above, describes the Teradata DBMS data types that the interface supports and details how they are converted when the engine reads them from or writes them to the Teradata DBMS.

Can I emit a native Teradata DBMS data type, such as a Timestamp and a Date, without writing explicit SQL?

A Yes. The examples below show two ways to do this. For more, *see "Using TIME and TIMESTAMP*" in the S/A Interface to Teradata Chapter. Specifically, look for examples that issue a LIBNAME statement (use LIBNAME connection method) and assert a SAS format or specify DBTYPE= to create the Teradata DBMS type.

Example: 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;
```

Example: Using DBTYPE= To Create a Teradata Timestamp

You can also use the Teradata engine data set option, DBTYPE=, to emit 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;
```

PreFetch

Where can I learn more about PreFetch?

A The S/A Interface to Teradata Chapter listed above 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 for PreFetch(3), your limit is three sessions. In actuality if you have multiple LIBNAMES open simultaneously **and** set the option PreFetch globally -- that is, use PreFetch for every LIBNAME -- the engine actually supports three sessions **per** LIBNAME.

Where are the macros created by the PreFetch facility stored?

A S/A Interface to Teradata writes the PreFetch macro(s) 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 to your default[17] database (the database that you connect to automatically in the absence of the DATABASE= specification.)

What happens if I don't have permission to create macros in the database that I accessed?

A S/A Interface to Teradata will still issue a CREATE MACRO request and the Teradata DBMS will return an error message. S/A Interface to Teradata passes the message back and continues processing. Since the macro is not created, there is no processing advantage when you next run the job with PreFetch enabled.

FastLoad

Can S/A Interface to Teradata append rapidly to DBMS tables without a MultiLoad capability?

A As of version 8.2, S/A Interface to Teradata does not have a MultiLoad feature. However, you can still perform very rapid DBMS table appends if you employ a multi-step process. The multi-step processing is much, much faster than a normal SAS row-by-row append and permits you to complete all work within the SAS session or job. Just another reminder: duplicates rows are dropped. (For details, see "Enhanced Performance Example: Two-Step Append with S/A FastLoad Enabled" on page 17.)

How does FastLoad affect writes to Teradata DBMS tables?

A When FastLoad is not enabled (the default), S/A 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 also passes, for each and every row, an INSERT statement, along with the binary data to the Teradata DBMS.

[17] Typically, your default database matches your logon name.

In turn the Teradata DBMS re-parses the INSERT statement. Thus, re-parsing is required for each and every INSERT statement.

FastLoad -- whether performed by S/A FastLoad or the Teradata FastLoad Utility -- eliminates re-parsing of the INSERT statement. But, the inherent nature of fastloading causes 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 tremendous speed that FastLoad delivers.

When S/A FastLoad is enabled, you obtain a vast increase in speed because it opens multiple sessions to the Teradata DBMS. As a consequence, it pumps data to the DBMS in parallel. (The Teradata FastLoad Utility and the Teradata MultiLoad Utility also perform parallel processing.)

How many Teradata sessions does S/A FastLoad use?

A S/A FastLoad allocates as many sessions as there are AMPs (see "Glossary" on page 36) on the Teradata DBMS server. If you want to limit the number of sessions that are allocated, you can use the undocumented option, SESSIONS=.

Example of SESSIONS= Option

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

Does S/A FastLoad support checkpointing?

A Yes, this feature is available. For more information see the FASTLOAD option, CHECKPOINT=, which is described in the S/A Interface to Teradata Chapter of the SAS online documentation (see "What other documentation is available for a SAS or Teradata DBMS end user?" on page29.)

Transaction Semantics: ANSI Mode versus Teradata Mode

What are the effects of having the client session set to ANSI rather than Teradata Mode?

A Below we table some differences. To obtain all the rules for ANSI or Teradata Mode, see the TERADATA RDBMS SQL Reference manual.

	ANSI Mode	TERADATA Mode
Transaction initiation	Always implicit	Implicit or explicit
Default for character comparisons	CASESPECIFIC	NOT CASESPECIFIC
Explicit COMMIT WORK statement required after every DDL (DBMS) statement	Y	N
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 S/A Interface to Teradata?

A Yes. Release 8.2 and higher furnishes limited support of Teradata Mode using PROC SQL and an explicit pass-through option, MODE=Teradata. (No further support of Teradata Mode is planned.)

The example that follows shows use of the option, MODE=Teradata, to obtain the case insensitive behavior of Teradata mode. (Note: because we turn on Teradata Mode, SQL COMMIT statements are not required.)

Example: The SQL Pass-Through option, MODE=Teradata

```
/* 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 NOT CASESPECIFIC behavior even when the Teradata engine has set my session to ANSI mode?

A Yes. To mimic the "NOT CASESPECIFIC" functionality of BTEQ, which defaults to Teradata mode, you can 'cast' your WHERE clause arguments. Thus, in the example below, instead of using:

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

you would use the following:

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

Use of CAST is a 'portable' way to effect NOT CASESPECIFIC functionality. That is, the CAST example statement will work in either mode: ANSI or Teradata. 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 a smart technique to employ whenever possible. (You can also use CAST with S/A 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 previously the SASTRACE option, used with either the LIBNAME or SQL pass-through connection method, shows you the SQL that the engine emits. When used with SQL pass-through, the option also tells you which side -- SAS or Teradata -- performed the work.

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

Another PROC SQL option, _METHOD, generates log output that identifies which query work actually passes to the Teradata DBMS. With this debug information, you can create faster, more efficient queries -- queries that make the Teradata DBMS do the subset processing and use SAS, via PROC SQL, to merely display the result set returned. The _METHOD option also can help you avoid a common query pitfall: the SQL that your query generates requests Teradata to perform full table scans -- the WHERE clause and functions are not passed and thus are not performed by the Teradata DBMS. Below is an example using the _METHOD option. (Keep in mind, this option returns output for both explicit and implicit SQL pass-through processing.)

Example using the _METHOD Option

```
Proc sql _method

Select *
   from yourlib.table1;
```

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

A The SAS DATA step MODIFY clause, which reads an observation from the transaction data set, and uses dynamic WHERE processing to locate the matching observation in the master data set, will not work when run against Teradata. Thus, it does not work when executed from S/A Interface to Teradata or S/A Interface to ODBC. If you run the DATA step Modify code below with S/A 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."

```
/ *DATA Step with MODIFY */
libname trlib teradata user=testuser password=testpass
reread_exposure=yes;
proc delete data=trlib.master1;
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;
```

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

Workaround
A workaround to this MODIFY problem – that also furnishes upsert capability – is to use explicit SQL pass-through.

MODIFY Workaround and Example of Upsert Processing (Two Teradata DBMS Tables)

In this example, we are matching customer records stored in a transaction table against the customer master table. If a match is found, we want to update the existing master record. If no match is found, we 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 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.

You can determine the source of a definition by looking at the label at the top of the column. Definitions in the NCR column repeat, in full or part, information 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 our commentary or a repeat of 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.	

Term	NCR Definition	SAS Definition
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. (See "SAS Options to Manage Implicit Pass-through" on page 21 and "Example using the _METHOD Option" on page 34).
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 S/A Interface to Teradata option that when enabled fastloads empty tables on the Teradata DBMS and does not require the Teradata FastLoad Utility.
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)	
Clique	A logical group of nodes on an NCR system 5100M that shares access to disk storage.	
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)	

Term	NCR Definition	SAS Definition
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 the ones 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.