



Global Business Intelligence and Data Integration Practice

Troubleshooting SAS and Teradata Query Performance Problems

Document Type: Best Practice

Date: January 14, 2010

Contact Information

Name: Jeffrey D. Bailey

Title: Database Technical Architect

Phone Number: (919) 531-6675

E-mail address: Jeff.Bailey@sas.com

**THE
POWER
TO KNOW®**

Revision History

Version	By	Date	Description
1.0	Jeffrey D. Bailey	March 14, 2008	Final Version
2.0	Jeffrey D. Bailey	January 8, 2010	Title changed Removed company confidential materials Removed extraneous materials Updated Client Screenshots to TTU 13.0 General edits

Table of Contents

1	Introduction	1
1.1	Purpose of the Paper	1
1.2	Target Audience	1
1.3	Overview	1
1.4	Assumptions	2
1.5	Technical Problem(s) Addressed	2
1.6	Environment Requirements	3
	1.6.1 Recommended knowledge and Training (Summary)	3
	1.6.2 Software and Hardware Configuration (Summary)	3
2	Troubleshooting SAS and Teradata Performance Problems	4
2.1	Verify that you can connect to Teradata Using SAS	4
	2.1.1 Connecting to Teradata via Teradata Tools and Utilities	4
2.2	Connecting to Teradata Using SAS® Foundation	7
2.3	Capture the SQL Statements that are Passed to Teradata	8
2.4	Find the Trouble Spots	15
2.5	Compare SAS runtimes with Teradata runtimes	16
	2.5.1 Executing a Query with Teradata BTEQ	17
	2.5.2 Executing a Query with Teradata SQL Assistant	18
2.6	See how Teradata is executing the SQL query	23
	2.6.1 The EXPLAIN SQL Statement	24
	2.6.2 Visual Explain	25
	2.6.3 Explain Plan Recommendations	31
2.7	Collect statistics	32
	2.7.1 Random AMP Sampling	32
	2.7.2 Full Statistics Collection	33
	2.7.3 Random sampling with the USING SAMPLE option	36
	2.7.3.1 USING SAMPLE option – Column Not Indexed	36
	2.7.3.2 USING SAMPLE option – Primary Index	36
	2.7.3.3 USING SAMPLE option – Secondary Index	36

2.7.4 Statistics Collection Recommendations	36
2.8 Hot AMPing	37
3 Summary	39
3.1 Summary of Lessons Learned	39
4 Bibliography	40
5 Credits and Acknowledgements	41

1 Introduction

1.1 Purpose of the Paper

Teradata is one of the most robust and best performing database management systems (DBMS) available. Its performance is legendary. That being said, there are going to be times when you – the SAS implementer – are going to face performance issues. That is where this paper comes in.

You don't have to be a Teradata database administrator (DBA) to troubleshoot performance issues. Using SAS and client tools supplied by Teradata, you can determine the root cause of many performance problems; fixing these issues might be well within your grasp. There are times that you will need a DBA to address performance issues, but after reading this document you will have the knowledge and skills required to guide them.

Performance issues are one of the leading causes of delay in implementing solutions that help our customers. Performance issues also cause untold frustration and delays in getting vital information to the decision makers in an organization.

1.2 Target Audience

The target audience for this paper is SAS Intelligence Platform administrators, SAS programmers and consultants who are experiencing performance issues when accessing Teradata data from SAS.

1.3 Overview

Troubleshooting performance issues is not magic; troubleshooting performance issues is not outside your abilities; Troubleshooting performance issues is simply using the tools that you have at your disposal to shine a light on the root cause of a problem. Even if you cannot fix the problem you will be able to present it to the DBA in a clear and concise manner. That ability will get your problems solved more quickly than you thought possible.

We will:

- Collect timing information that will help determine if a slow performing query is a database issue.
- Determine the SQL that is being passed from SAS to Teradata.
- Run the query in a tool provided by the database vendor.
- Determine how the database is executing the query.
- Provide the database with the information it needs to efficiently execute the query.

- Discuss Teradata “hot AMP’ing”.

1.4 Assumptions

In order to do many of the steps described above you will need:

- Access to a Teradata environment on which to execute queries.
- A user ID and password for the database in question.
- Privileges in the Teradata environment what will allow you to collect and drop statistics.
- Privileges in the Teradata environment that will allow you to generate explain plans on the queries in question.
- SAS 9.1.3 software installed (including the SAS/ACCESS Interface to Teradata).
- Your SAS environment properly configured so that it will attach to Teradata.
- Access to the Teradata Tools and Utilities (TTU).
- Access to the Teradata Analysis Pak.

1.5 Technical Problem(s) Addressed

The technical problems are not as simple as the business problem. That being said, the technical issues are going to fall into a couple of categories. These categories are:

- SAS is handling requests that are best dealt with by the database.
- Poor indexing in the database.
- Database optimizer cannot choose a good access path because of bad catalog statistics.
- Poorly written queries.
- Poorly designed architecture.
- Poorly designed database.
- Hardware not up for the task at hand.

This paper is going to focus on the top three. Over the years we have found that post-processing by SAS, poor indexing and bad catalog statistics cause many, if not most, of the problems faced by SAS consultants in the field.

Simply capturing good statistics or adding (or deleting) an index can yield huge performance boosts. I have seen situations where collecting statistics and adding an index has greatly improved

performance. In a couple of cases it has allowed queries to complete in minutes when they were taking over 24 hours. Granted, most improvement will not be that spectacular, but it could be.

1.6 Environment Requirements

In order to follow the troubleshooting guidelines we are recommending you will have:

- Access to a Teradata environment on which to execute queries.
- A userid and password for the database in question.
- Privileges in the Teradata environment what will allow you to collect and drop statistics.
- Privileges in the Teradata environment that will allow you to generate explain plans on the queries in question.
- SAS 9.2 (or SAS 9.2) software installed (including the SAS/ACCESS Interface to Teradata).
- Your SAS environment properly configured so that it will attach to Teradata.
- Access to the Teradata Tools and Utilities (TTU).

If you do not have access to everything listed above, do not despair. Understanding the contents of this paper will allow you to request that the DBA do some of this. Simply knowing about these techniques will aid you in speaking with the DBA.

1.6.1 Recommended knowledge and Training (Summary)

It is recommended that you have a good understanding of SAS programming, the SAS/ACCESS Interface to Teradata, and SQL. If you don't, you might still find this document useful.

1.6.2 Software and Hardware Configuration (Summary)

In order to follow the steps documented in this paper you will need a Windows client machine, SAS Enterprise Guide or SAS Foundation, the SAS/ACCESS Interface to Teradata, a Teradata server, and the Teradata Tools and Utilities (TTU).

TTU must be configured so that you can connect to Teradata. The SAS software must be installed and configured such that you can use it to connect to Teradata.

2 Troubleshooting SAS and Teradata Performance Problems

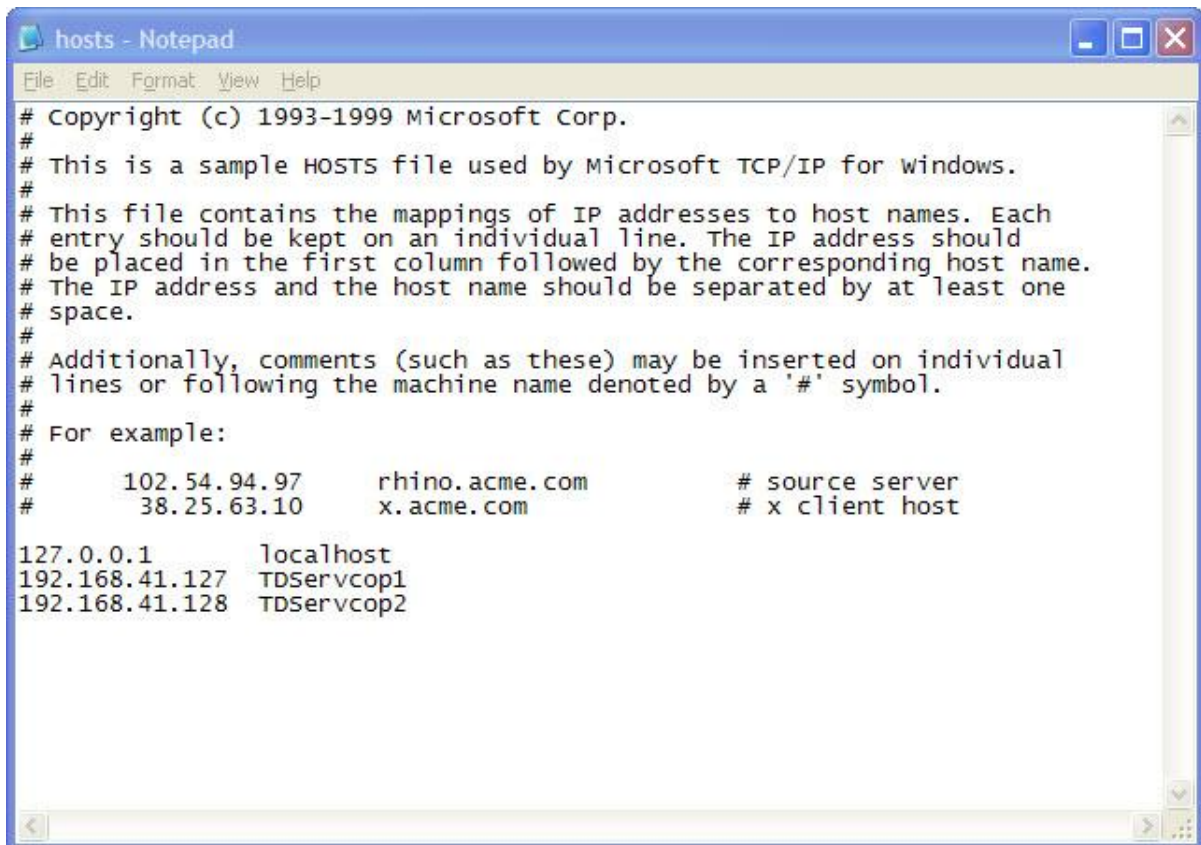
2.1 Verify that you can connect to Teradata Using SAS

There are many ways that you can connect to Teradata. You can use Teradata utilities, SAS Foundation, SAS Enterprise Guide (via a SAS Workspace Server or SAS Stored Process Server), the SAS Management Console (via a SAS Workspace Server). I always recommend that you start with the simplest.

Here are the scenarios and how you would test the connection:

2.1.1 Connecting to Teradata via Teradata Tools and Utilities

Teradata stores its connection information in the DNS server or the client machines hosts file. On Windows this is located in the C:\WINNT\system32\drivers\etc directory. Open the file in Notepad.



```

# Copyright (c) 1993-1999 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#       102.54.94.97       rhino.acme.com           # source server
#       38.25.63.10       x.acme.com               # x client host

127.0.0.1       localhost
192.168.41.127  TDServcop1
192.168.41.128  TDServcop2

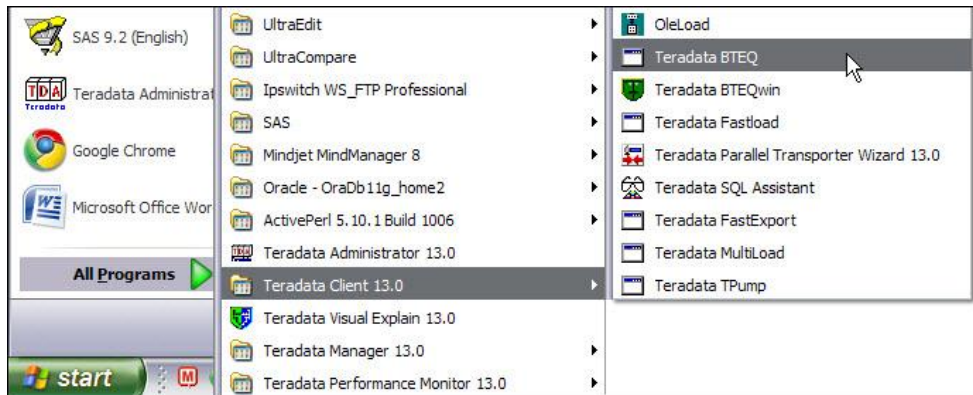
```

The Teradata server that we will be using for this discussion is TDServ. Notice that there are two references to this server. This is a multi-node Teradata environment. We don't need to use the "cop1" or "cop2" extensions when addressing the Teradata hostname. When we need to specify a Teradata server we will use TDServ.

Note: TDServ is the Teradata server that we will use in this discussion.

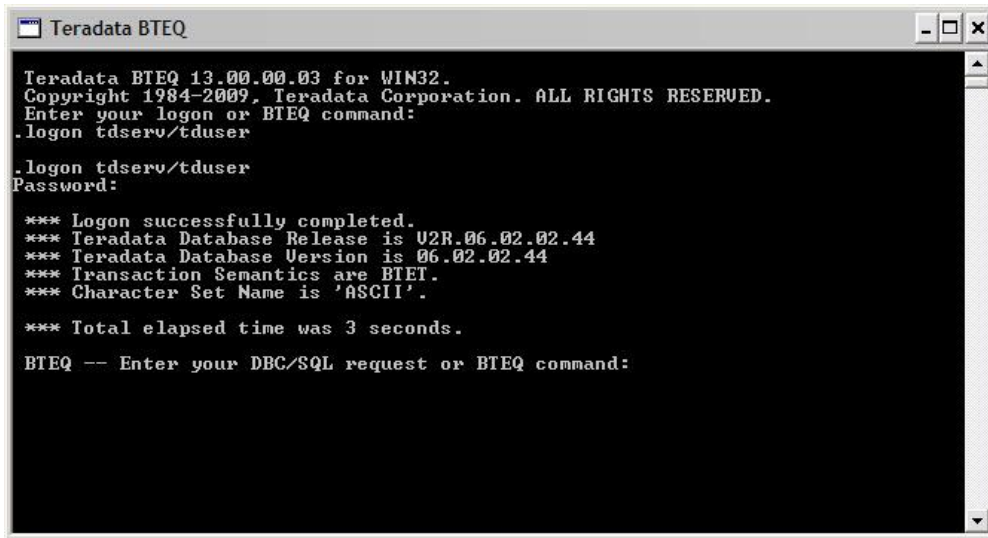
In this situation you want to ensure that the Teradata client is properly installed. This is the most basic connectivity test that you can do. When is it useful to do this? When you are dealing with a machine that has a new database client installed it is a good idea to test it. It is also a good idea to test with a client utility when you have added Teradata server references to the machine. You can use the Teradata utility (BTEQ or Teradata SQL Assistant) for this test. Here is an example.

1. Select **Teradata BTEQ** via the **Start Button**. Select **Start → All Programs → Teradata Client 13.0 → Teradata BTEQ**.



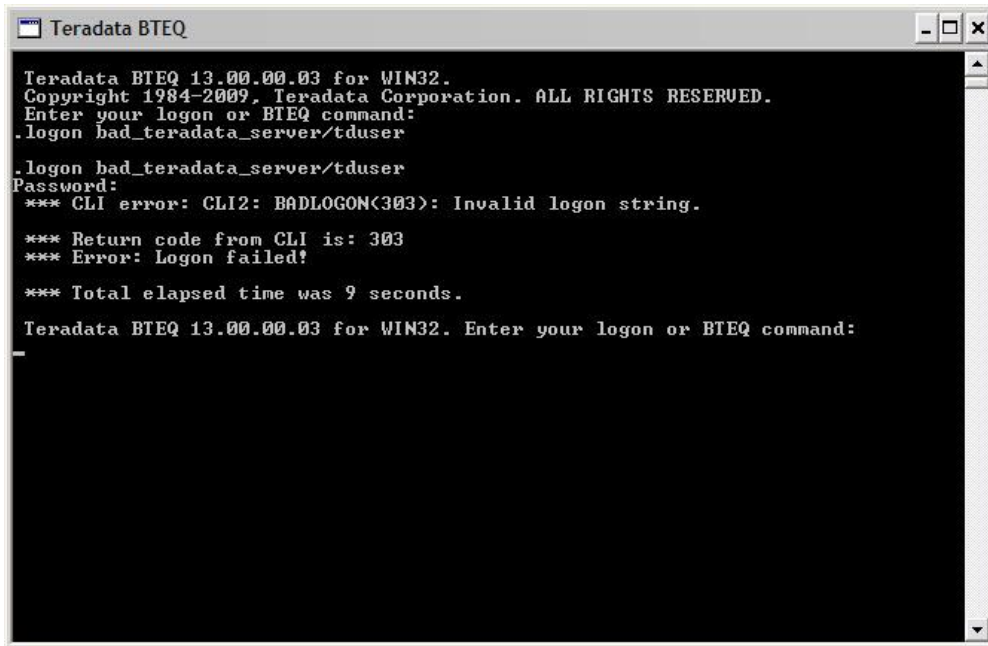
2. Next we are going to logon to Teradata. In order to do this you must have your Teradata client configured correctly and your DBA must have created a user ID and password for you. In this example the Teradata server name is TDServ and the user ID is tduser. The password is tdpaswd.

Here is an example of connecting to Teradata. Type **.logon tdserv/tduser** on the command line. Then **Enter**. Type **tduserpw** (the password). Then **Enter**.



```
Teradata BTEQ 13.00.00.03 for WIN32.  
Copyright 1984-2009, Teradata Corporation. ALL RIGHTS RESERVED.  
Enter your logon or BTEQ command:  
.logon tdserv/tduser  
.logon tdserv/tduser  
Password:  
*** Logon successfully completed.  
*** Teradata Database Release is U2R.06.02.02.44  
*** Teradata Database Version is 06.02.02.44  
*** Transaction Semantics are BTEI.  
*** Character Set Name is 'ASCII'.  
  
*** Total elapsed time was 3 seconds.  
BTEQ -- Enter your DBC/SQL request or BTEQ command:
```

This shows a successful connection. Here is an example of a failed connection. The Teradata server value is incorrect. Type **.logon bad_teradata_server/tduser** on the command line. Then **Enter**. Type **tduserpw** (the password). Then **Enter**.

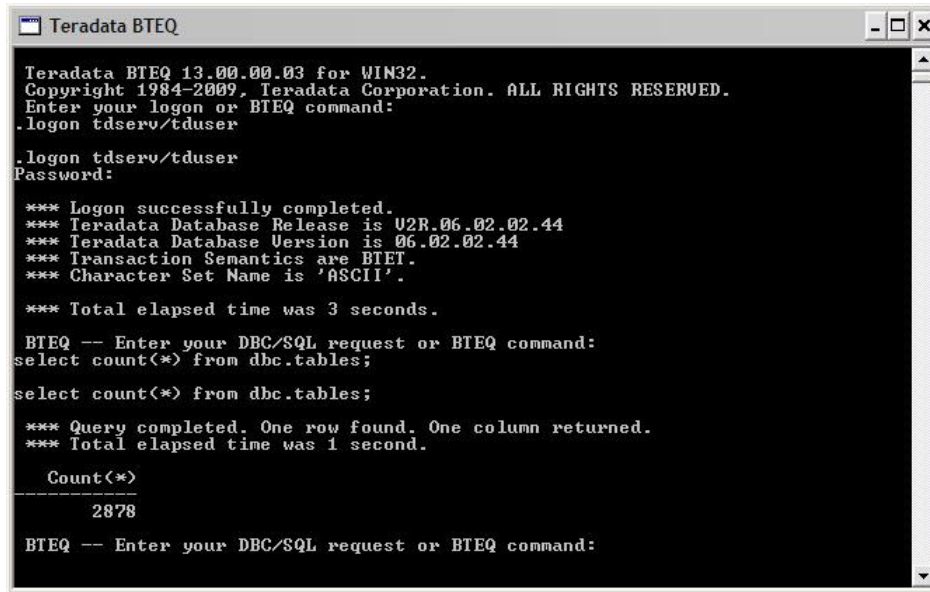


```
Teradata BTEQ 13.00.00.03 for WIN32.  
Copyright 1984-2009, Teradata Corporation. ALL RIGHTS RESERVED.  
Enter your logon or BTEQ command:  
.logon bad_teradata_server/tduser  
.logon bad_teradata_server/tduser  
Password:  
*** CLI error: CLI2: BADLOGON(303): Invalid logon string.  
  
*** Return code from CLI is: 303  
*** Error: Logon failed!  
  
*** Total elapsed time was 9 seconds.  
Teradata BTEQ 13.00.00.03 for WIN32. Enter your logon or BTEQ command:  
_
```

3. In order to fully test the connection you need to issue an SQL query. Here is an example showing the number of tables in the database. Logon the Terdata server using a valid account.

Enter the following query and press the Enter key:

Select count(*) from dbc.tables;



```

Teradata BTEQ
Teradata BTEQ 13.00.00.03 for WIN32.
Copyright 1984-2009. Teradata Corporation. ALL RIGHTS RESERVED.
Enter your logon or BTEQ command:
.logon tdservr/tduser

.logon tdservr/tduser
Password:

*** Logon successfully completed.
*** Teradata Database Release is U2R.06.02.02.44
*** Teradata Database Version is 06.02.02.44
*** Transaction Semantics are BIEI.
*** Character Set Name is 'ASCII'.

*** Total elapsed time was 3 seconds.

BTEQ -- Enter your DBC/SQL request or BTEQ command:
select count(*) from dbc.tables;
select count(*) from dbc.tables;

*** Query completed. One row found. One column returned.
*** Total elapsed time was 1 second.

      Count(*)
      -----
          2878

BTEQ -- Enter your DBC/SQL request or BTEQ command:

```

You can exit Teradata BTEQ by typing “.exit” and then pressing the Enter key. Notice that there is a period in front of exit.

2.2 Connecting to Teradata Using SAS® Foundation

I recommend testing the connection via SAS Foundation for new installs or when an upgrade has been performed. You can use SAS® Enterprise Guide® for this test if you are testing new work space or stored process servers. Our goal is to ensure that the SAS/ACCESS Interface to Teradata has been installed and configured properly.

We can do that by submitting code similar to this (note: TDPID= is an alias for SERVER=. It is commonly used in mainframe environments, but they can be used interchangeably).

```
LIBNAME mytera TERADATA USER=tduser PASSWORD=tdpasswd SERVER=TDServ;
```

You should see something similar to this in the SAS Log.

```

5  LIBNAME mytera TERADATA USER=tduser PASSWORD=XXXXXXXXX SERVER=TDServ;
NOTE: Libref MYTERA was successfully assigned as follows:
      Engine:          TERADATA
      Physical Name:   TDServ

```

If this test fails, you will want to test using Teradata BTEQ.

Creating a SAS Library for Teradata in Metadata

Once you have proven that you can connect to Teradata via Teradata BTEQ and SAS Foundation you might want to verify that you can create a SAS Library for your Teradata server in SAS Management Console. If you have a problem doing this, you will want to run through this test again. It is much easier to determine the problem via Teradata BTEQ or SAS Foundation than it is in SMC.

2.3 Capture the SQL Statements that are Passed to Teradata

In order to troubleshoot SQL problems we need to know one thing. What is the SQL that is being passed to the database? Before we discuss how this is done, let's talk about the two ways that SQL is passed to the database.

Explicit SQL Pass-Through

When you use explicit SQL Pass-Through you choose the SQL that is passed to the database. This can be vendor-specific SQL. In our case that SQL will be Teradata specific.

Let's say we want to use Teradata SQL to create a table and then immediately drop it. Here is a code example.

```
PROC SQL;
  CONNECT TO TERADATA (USER=tduser PW=tdpasswd SERVER=TDserv);
  EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
  EXECUTE ( CREATE TABLE tduser.TestTable
    ,FALLBACK
    ,NO BEFORE JOURNAL
    ,NO AFTER JOURNAL
    (
      Number    CHAR(10) TITLE 'Number' NOT NULL
    ,Name       CHAR(25) TITLE 'Name' NOT NULL
    )
    UNIQUE PRIMARY INDEX( Number ) ) BY TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
  EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
QUIT;
```

The Teradata SQL is contained within the parentheses in the EXECUTE statements. The interesting thing about this is that we can pass database-specific SQL commands to the database. This means you know exactly what SAS is asking the database to. There is no doubt.

Implicit Pass-Through

Implicit SQL Pass-Through is transparent, it happens behind the scenes. This is ANSI SQL that is generated by the SAS/ACCESS Interface to Teradata engine. An engine is an executable code

module. It is a dynamic link library (DLL) on Windows or a shared library in UNIX. The engine tries to construct the SQL in such a way that as much work as possible is done by the database. The work that cannot be performed by Teradata is completed by SAS. For example, joins could be performed by SAS which means passing lots of data, unnecessarily, from Teradata to SAS. We want Teradata to do as much of the processing as possible.

SASTRACE= and SASTRACELOC=

You are probably expecting the worst, but good fortune is with us. It is easy to see the SQL that is being passed to Teradata. All it takes is a couple of SAS options. The options are SASTRACE= and SASTRACELOC=

SASTRACE=

This is a SAS/ACCESS debugging option that will display SQL being passed to the database in the SAS log. It is one of the most powerful troubleshooting tools available for database work. In addition to the SQL which is being passed, you can also have timing information, DBMS calls (API and Client calls) and connection information sent to the SAS log.

SASTRACELOC=

This tells you where to write the SASTRACE= output. Specify SASLOG for this.

NOSTSUFFIX

One of the difficulties in using SASTRACE= is the quantity of information that is written to the SAS log. Fortunately, you can limit some of that by specifying NOSTSUFFIX.

Let us use our previous example to see the exact SQL statements that are being passed to Teradata. I am adding an options statement to the top of this code.

```
OPTIONS SASTRACE=' , , ,d' SASTRACELOC=saslog;
PROC SQL ;
  CONNECT TO TERADATA (USER=tduser PW=tdpasswd SERVER=TDserv) ;
  EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
  EXECUTE ( CREATE TABLE tduser.TestTable
    ,FALLBACK
    ,NO BEFORE JOURNAL
    ,NO AFTER JOURNAL
    (
      Number   CHAR(10) TITLE 'Number' NOT NULL
    ,Name      CHAR(25) TITLE 'Name' NOT NULL
    )
    UNIQUE PRIMARY INDEX( Number ) ) BY TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
  EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;
  EXECUTE (COMMIT) BY TERADATA;
QUIT;
```

Here is the output. I have made the SQL statements that are being passed much easier to read by making the font red. I have also included a Teradata error in this log. See if you can find it.

```

169  OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog;
170
171  PROC SQL;
172      CONNECT TO TERADATA (USER=tduser PW=XXXXXXXXX SERVER=TDServ);
173
174      EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;
      78 1518123619 no_name 0 SQL
TERADATA_12: Executed: 79 1518123619 no_name 0 SQL
DROP TABLE tduser.TestTable 80 1518123619 no_name 0 SQL
      81 1518123619 no_name 0 SQL
ERROR: Teradata execute: Object 'tduser.TestTable' does not exist.
175
176      EXECUTE (COMMIT) BY TERADATA;
      82 1518123619 no_name 0 SQL
TERADATA_13: Executed: 83 1518123619 no_name 0 SQL
COMMIT 84 1518123619 no_name 0 SQL
      85 1518123619 no_name 0 SQL
177
178      EXECUTE ( CREATE TABLE tduser.TestTable
179                  ,FALLBACK
180                  ,NO BEFORE JOURNAL
181                  ,NO AFTER JOURNAL
182                  (
183                      Number      CHAR(10) TITLE 'Number' NOT NULL
184                      ,Name        CHAR(25) TITLE 'Name' NOT NULL
185                  )
186                  UNIQUE PRIMARY INDEX( Number ) ) BY TERADATA;
      86 1518123619 no_name 0 SQL
TERADATA_14: Executed: 87 1518123619 no_name 0 SQL
CREATE TABLE tduser.TestTable ,FALLBACK ,NO BEFORE JOURNAL ,NO AFTER
JOURNAL ( Number CHAR(10)
TITLE 'Number' NOT NULL ,Name CHAR(25) TITLE 'Name' NOT NULL )
UNIQUE PRIMARY INDEX( Number ) 88
1518123619 no_name 0 SQL
      89 1518123619 no_name 0 SQL
187
188      EXECUTE (COMMIT) BY TERADATA;
      90 1518123619 no_name 0 SQL
TERADATA_15: Executed: 91 1518123619 no_name 0 SQL
COMMIT 92 1518123619 no_name 0 SQL
      93 1518123619 no_name 0 SQL
189
190      EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;
      94 1518123619 no_name 0 SQL
TERADATA_16: Executed: 95 1518123619 no_name 0 SQL
DROP TABLE tduser.TestTable 96 1518123619 no_name 0 SQL

```

```

97 1518123619 no_name 0 SQL
191
192      EXECUTE (COMMIT) BY TERADATA;
98 1518123619 no_name 0 SQL
TERADATA_17: Executed: 99 1518123619 no_name 0 SQL
COMMIT 100 1518123619 no_name 0 SQL
101 1518123619 no_name 0 SQL
193
194  QUIT;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE SQL used (Total process time):
      real time          0.42 seconds
      cpu time           0.06 seconds

```

I think that you can see how valuable this information is. Believe me; you cannot effectively work with databases without knowing this. That being said, there is a lot of stuff in there that doesn't really help. Let's get rid of it with NOSTSUFFIX (NOSTSUFFIX is an alias).

Were you able to find the error? The problem is that the code drops a table that does not exist.

We only need to change the OPTION statement. Let's use this one.

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog NOSTSUFFIX;
```

Here is the SAS Log. I have made the SQL statements that are being passed much easier to read by making the font red. I have also included a Teradata error in this log. See if you can find it.

```

221  OPTIONS SASTRACE=',,,d' SASTRACELOC=saslog NOSTSUFFIX;
222
223  PROC SQL;
224      CONNECT TO TERADATA (USER=tduser PW=XXXXXXXXX SERVER=TDServ);
225
226      EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;

```

```

TERADATA_24: Executed:
DROP TABLE tduser.TestTable

```

```

ERROR: Teradata execute: Object 'tduser.TestTable' does not exist.
227
228      EXECUTE (COMMIT) BY TERADATA;

```

```

TERADATA_25: Executed:
COMMIT

```

```

229
230      EXECUTE ( CREATE TABLE tduser.TestTable
231                  ,FALLBACK
232                  ,NO BEFORE JOURNAL
233                  ,NO AFTER JOURNAL
234                  (

```

```

235             Number      CHAR(10) TITLE 'Number' NOT NULL
236             ,Name       CHAR(25) TITLE 'Name' NOT NULL
237         )
238         UNIQUE PRIMARY INDEX( Number ) ) BY TERADATA;

TERADATA_26: Executed:
CREATE TABLE tduser.TestTable ,FALLBACK ,NO BEFORE JOURNAL ,NO AFTER
JOURNAL ( Number CHAR(10)
TITLE 'Number' NOT NULL ,Name CHAR(25) TITLE 'Name' NOT NULL )
UNIQUE PRIMARY INDEX( Number )

239
240     EXECUTE (COMMIT) BY TERADATA;

TERADATA_27: Executed:
COMMIT

241
242     EXECUTE (DROP TABLE tduser.TestTable) BY TERADATA;

TERADATA_28: Executed:
DROP TABLE tduser.TestTable

243
244     EXECUTE (COMMIT) BY TERADATA;

TERADATA_29: Executed:
COMMIT

245
246 QUIT;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE SQL used (Total process time):
      real time          0.40 seconds
      cpu time           0.03 seconds

```

The previous examples have all used explicit SQL Pass-Through. We told SAS the “exact” SQL statement to pass along to Teradata. That won’t always be the case. In many situations – DATA step, PROC SQL, IMS, etc. - SAS will generate SQL and then implicitly pass it to the database.

This error is the same as the previous one. The code drops a table that does not exist.

Let’s look at an implicit SQL Pass-Through. When we execute a DATA step the SAS/ACCESS engine will convert the SAS calls to SQL statements. In this case a SELECT statement. SAS will hand those statements to the database for execution. Here is the code that we will run.

```

OPTIONS SASTRACE=' , , ,d' SASTRACELOC=saslog NOSTSUFFIX;

LIBNAME mytera TERADATA USER=tduser PASSWORD=tdpasswd TDPID=TDserv;

DATA work.test;

```

```

SET mytera.TestTable;
WHERE Name = 'Christine';
RUN;

```

Here is the SAS log. Take a look and see if the WHERE clause was passed to Teradata. You really want the WHERE clause passed to Teradata.

```

TERADATA_10: Prepared:
SELECT * FROM "TestTable"

```

```

427 DATA work.test;
428     SET mytera.TestTable;
429     WHERE Name = 'Christine';
430 RUN;

```

```

TERADATA: trqacol- Casting on. Raw row size=29, Casted size=33,
CAST_OVERHEAD_MAXPERCENT=20%

```

```

TERADATA_11: Prepared:
SELECT CAST("Number" AS FLOAT),"Name" FROM "TestTable" WHERE
("Name" = 'Christine' )

```

```

TERADATA_12: Executed:
SELECT CAST("Number" AS FLOAT),"Name" FROM "TestTable" WHERE
("Name" = 'Christine' )

```

```

TERADATA: trget - rows to fetch: 1
NOTE: There were 1 observations read from the data set
MYTERA.TestTable.
      WHERE Name='Christine';
NOTE: The data set WORK.TEST has 1 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time          0.20 seconds
      cpu time           0.03 seconds

```

You can see that the entire WHERE clause is being passed to Teradata. Passing the WHERE clause to the database is good because it means that data transfer is being kept to a minimum. Some of the prepared statements that are displayed are issued by SAS in order to determine the columns that are in the table. The CAST() function call is necessary due to data type differences between SAS and Teradata.

SASTRACE=

Valid values for the SASTRACE= option are:

- '.,d'

specifies that all SQL statements sent to the DBMS are sent to the log. These statements include the following:

SELECT	DELETE
CREATE	SYSTEM CATALOG
DROP	COMMIT
INSERT	ROLLBACK
UPDATE	

For those engines that do not generate SQL statements, the API calls, including all parameters, are sent to the log.

- '.,d,'

specifies that all routine calls are sent to the log. When this option is selected, all function enters and exits, as well as pertinent parameters and return codes, are traced. The information, however, will vary from engine to engine.

This option is most useful if you are having a problem and need to send a SAS log to technical support for troubleshooting.

- 'd,'

specifies that all DBMS calls, such as API and Client calls, connection information, column bindings, column error information, and row processing are sent to the log. However, this information will vary from engine to engine.

This option is most useful if you are having a problem and need to send a SAS log to technical support for troubleshooting.

- '.,.,s'

specifies that a summary of timing information for calls made to the DBMS is sent to the log.

- '.,.,sa'

specifies that timing information for each call made to the DMBS, along with a summary, is sent to the log.

- '.,t,'

specifies that all threading information is sent to the log. This information includes:

- The number of threads spawned
- The number of observations each thread contains
- The exit code of the thread, should it fail.

You can turn off SASTRACE by issuing this command in your SAS code.

```
OPTIONS SASTRACE=off;
```

2.4 Find the Trouble Spots

When you are dealing with databases you need to be prepared to deal with performance problems. That is a fact; there is no way around it. The database is a big, monolithic, entity that can make or break your consulting projects. You need efficiently performing SQL so that you can meet the constraints of the ETL batch window. You know the one; it keeps getting tighter and tighter. One day, it will certainly be a problem.

That is all-well-and-good, but how do you know when you are having a database-specific performance issue? It isn't too difficult. First, if your programs are taking hours to run and they are accessing data that lives in a database, then you should consider it. Second, if your programs are returning in minutes and you believe that they should not take that long, then you should look at it.

It is always a good idea to start with the PROC or DATA steps that are taking the most time. For the purpose of this document we are going to assume that these long running processes are accessing data which is stored in a DBMS. Do not forget to look at the queries that are taking minutes. If that code is running often during your job stream it could add up to hours. Prioritize, leave no stone unturned, and plan to look at everything.

Hopefully, you have followed the recommendations made in the previous section of this document and turned on FULLSTIMER=, SASTRACE= and SASTRACELOC. You won't be able to effectively troubleshoot issues without that information.

The most important option for finding problem queries is FULLSTIMER=. Consider some real output. In the following example the SAS Library materb actually points to a Teradata database.

```
118 proc sql;
119   select count(*)
120       from materb.contacts c
121           ,materb.cm_party p
122       where c.party_rk = p.party_rk
123   ;
```

<SASTRACE Messages Removed>

```
NOTE: PROCEDURE SQL used (Total process time):
      real time             17:08.69
      user cpu time         0.07 seconds
      system cpu time       0.42 seconds
      Memory                586k
```

This PROC SQL statement took 17 minutes and 8.69 seconds to execute. As far as database queries go, this is not bad. You will likely see much worse. Sometimes much worse is better because it is obvious that the query needs work.

Examine the “user cpu time” and “system cpu time.” Add these two values together and you get less than half a second. This means that most of the time spent running the query was not spent by SAS – it was spent in Teradata. This is a good candidate for further research.

Another good way to determine if the time is being spent in the database is comparing SAS runtimes with Teradata runtimes.

2.5 Compare SAS runtimes with Teradata runtimes

When you approach a Teradata DBA and make the statement, “I have a slow SQL query in my SAS job and need your help to fix it.” They will most likely respond, “The problem is not Teradata. The problem is SAS!”

How do you address this situation? Capture the SQL that is being passed to Teradata. You will have access to that since you have SASTRACE= and SASTRACELOC= set to the appropriate values.

Here is an example.

```
118 proc sql;
119   select count(*)
120     from materb.contacts c
121           ,materb.cm_party p
122   where c.party_rk = p.party_rk
123   ;
```

TERADATA_96: Prepared:

SELECT * FROM MA2."contacts" ← This statement is used to get a list of columns

TERADATA_97: Prepared:

SELECT * FROM MA2."cm_party" ← This statement is used to get a list of columns

TERADATA_98: Prepared:

select COUNT(*) from "MA2"."contacts" c, "MA2"."cm_party" p where c."PARTY_RK" = p."PARTY_RK"

ACCESS ENGINE: SQL statement was passed to the DBMS for fetching data.

TERADATA_99: Executed:

```
select COUNT(*) from "MA2"."contacts" c, "MA2"."cm_party" p where
c."PARTY_RK" = p."PARTY_RK" ← this is the statement executed by
Teradata
```

```
TERADATA: trget - rows to fetch: 1
124 quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time          17:08.69
      user cpu time      0.07 seconds
      system cpu time    0.42 seconds
      Memory              586k
```

That last highlighted SQL statement is the one that we want. It is the statement being passed to Teradata.

```
select COUNT(*) from "MA2"."contacts" c, "MA2"."cm_party" p where
c."PARTY_RK" = p."PARTY_RK"
```

We will take this SQL statement and execute it in either Teradata BTEQ or Teradata SQL Assistant. We will note the time it takes to execute and then compare it to the time it takes SAS to execute the query.

2.5.1 Executing a Query with Teradata BTEQ

Executing a Query with Teradata BTEQ

Teradata BTEQ is a command line interface application that is shipped as part of the Teradata Tools and Utilities package.



```
Command Prompt - bteq
BTEQ -- Enter your DBC/SQL request or BTEQ command:
select COUNT(*) from "MA2"."contacts" c, "MA2"."cm_party" p where c."PARTY_RK" =
p."PARTY_RK"
;
select COUNT(*) from "MA2"."contacts" c, "MA2"."cm_party" p where c."PARTY_
RK" = p."PARTY_RK"
;
*** Query completed. One row found. One column returned.
*** Total elapsed time was 17 minutes and 5 seconds.

Count(*)
-----
11589690

BTEQ -- Enter your DBC/SQL request or BTEQ command:
```

This is a good result. The majority of the time spent executing your query is being spent in Teradata. The problem, if there is one, is in Teradata and not SAS. Take these results to your DBA and hopefully, they will help you.

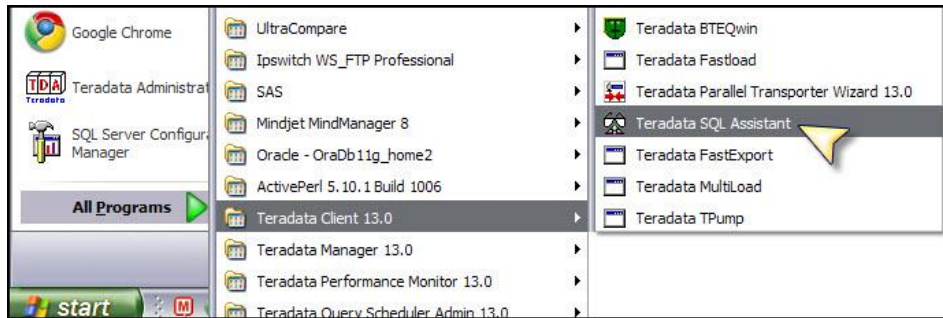
2.5.2 Executing a Query with Teradata SQL Assistant

Teradata SQL Assistant is a graphical user interface (GUI) application that is designed to execute SQL statements. Notice that I did not say “execute Teradata SQL Statements.” That was intentional. You can use this application to query many ODBC compliant databases. I have used it with Microsoft SQL Server and Oracle with no problem. It is a very versatile tool.

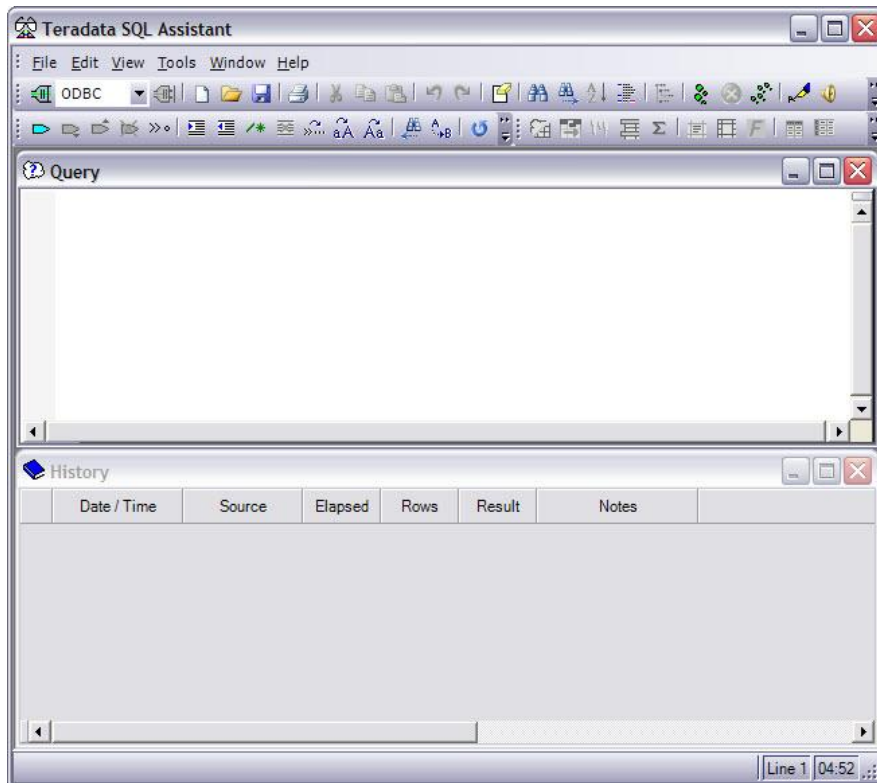
In this example I am using Teradata SQL Assistant 7.2. Your version number can differ.

Check the SQL execution time with Teradata SQL Assistant.


1. Invoke Teradata SQL Assistant via the Start button. **Start → All Programs → Teradata Client 13.0 → Teradata SQL Assistant.**

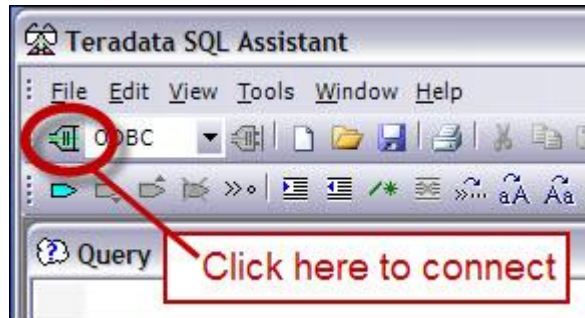


2. There is an introductory window which comes up, but it goes away so quickly that I can't capture the screen shot. When you see it you can ignore it.
3. When Teradata SQL Assistant comes up this is window that you will see.

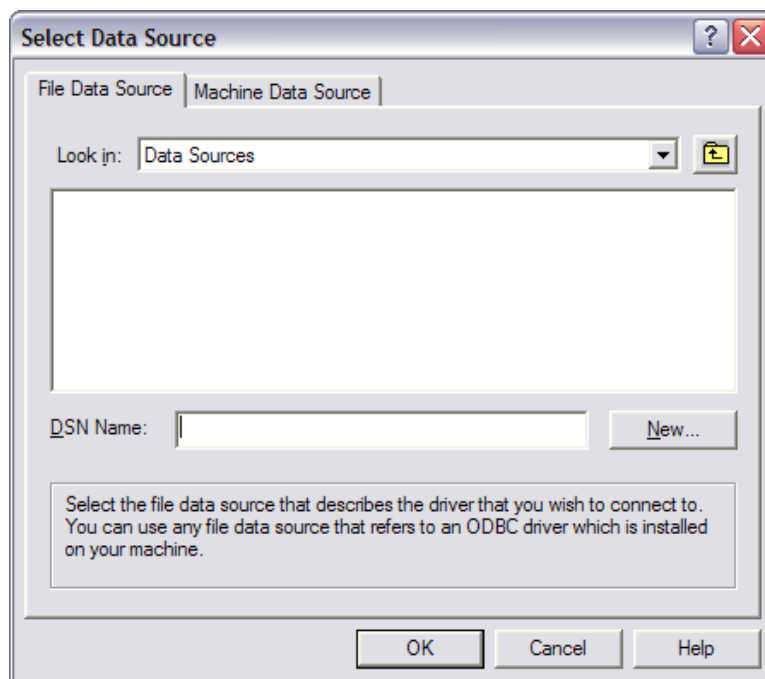


It is important to note that you are not connected to a database at this point.

4. Connect to the Teradata database by clicking .

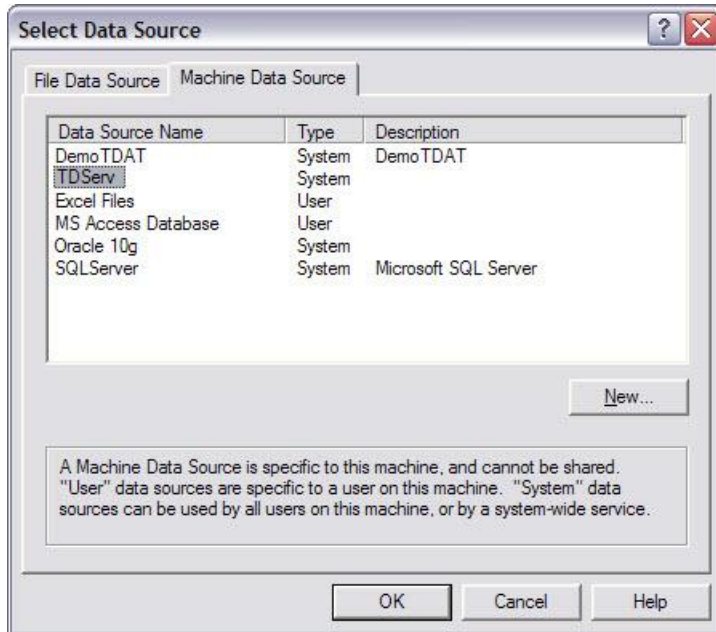


5. Typically, your ODBC data sources are Machine Data Sources. Select Machine Data Source.

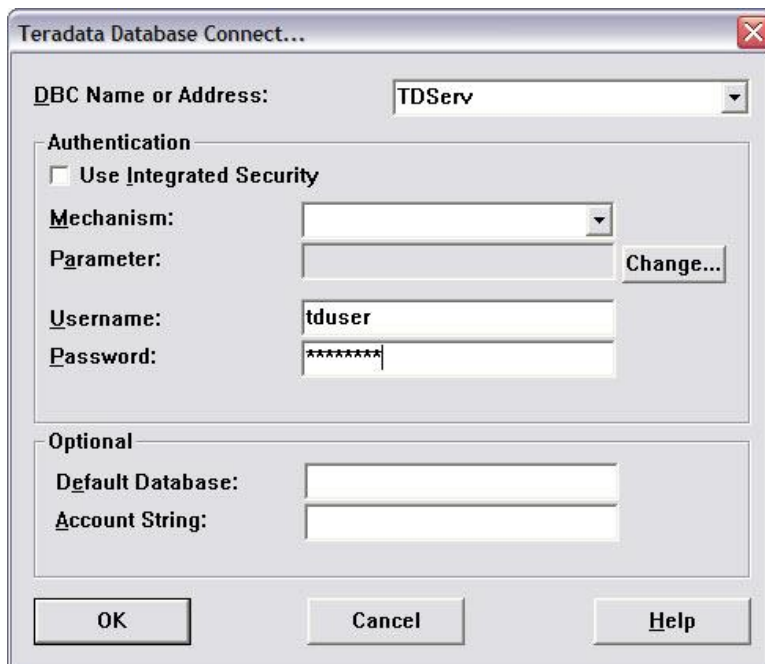


6. In this example we need to connect to TDServ. That is where our data lives. Click on **TDServ** to select the correct datasource. If you don't have a Teradata ODBC data source they you can create one using **New...**.

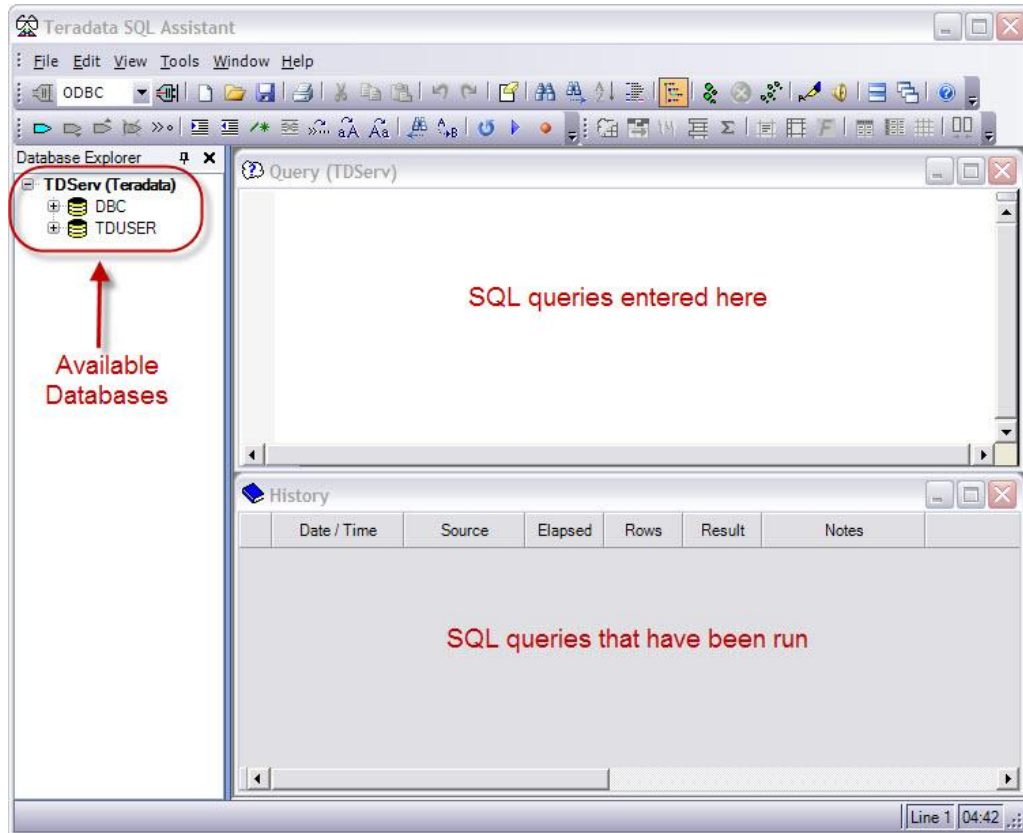
Once you have your data source highlighted, select **OK**.



7. You can store a user ID and password in the ODBC Data Source. From a security perspective, that is not usually a good idea. If you don't store the password you will be prompted for it. Enter the Password and then select **OK**.




8. At this point we are connected to the TDServ Teradata server. Here is an overview of the user interface.

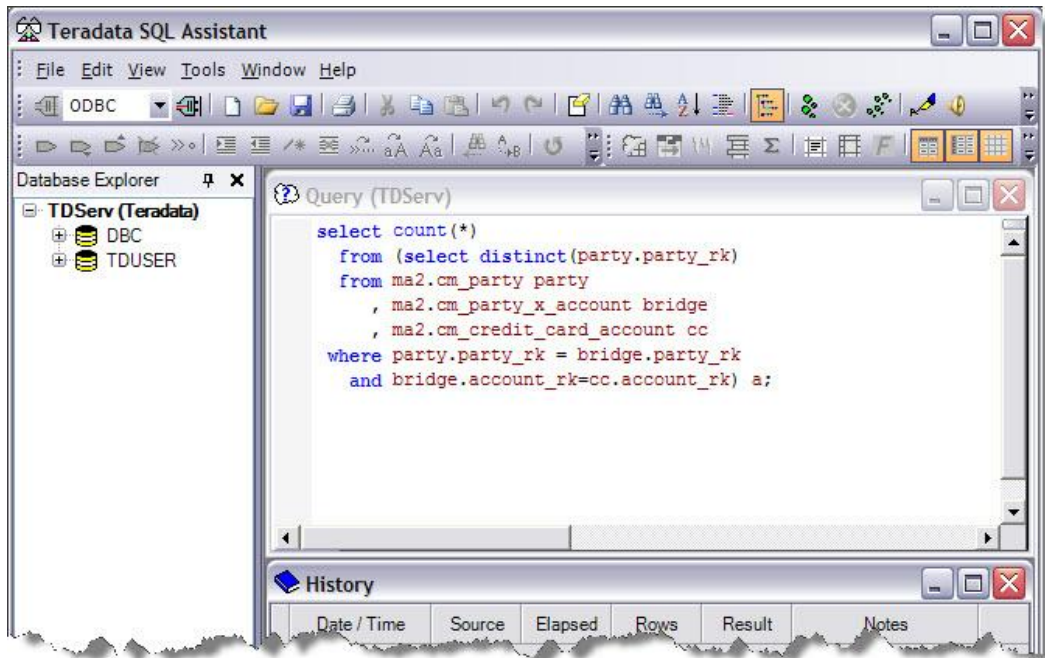


9. Let's execute a query using Teradata SQL Assistant. First, we enter the query into the Query pane. I had problems with cut and paste. Typing something into the Query pane and then deleting it seems to clear that up.

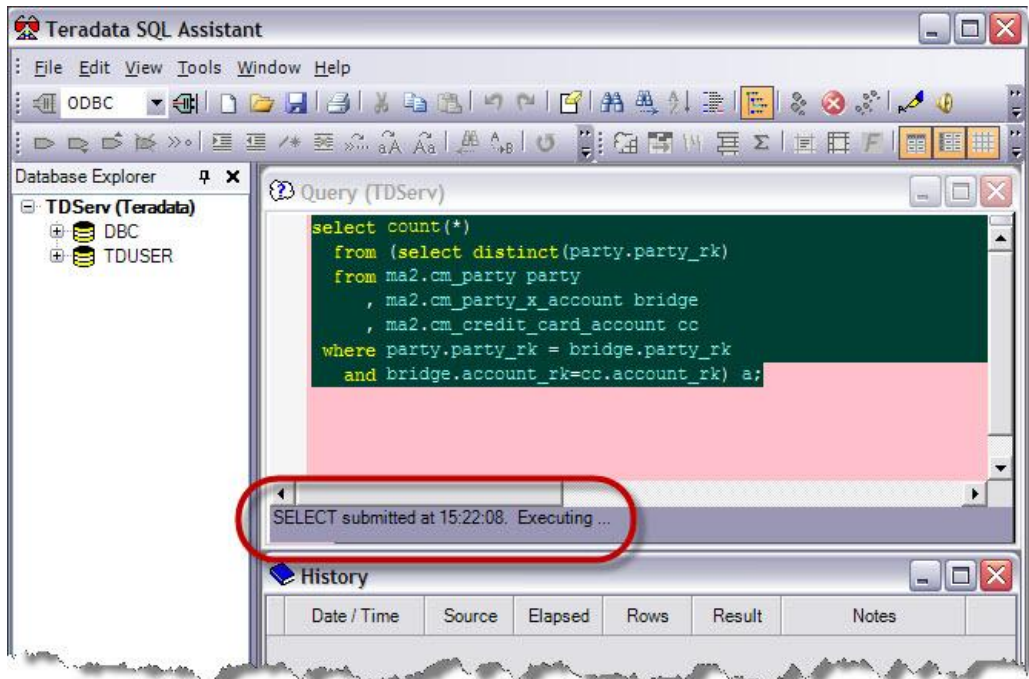
Here is the query we need to run.

```
select count(*)
  from (select distinct(party.party_rk)
        from ma2.cm_party party
            , ma2.cm_party_x_account bridge
            , ma2.cm_credit_card_account cc
 where party.party_rk = bridge.party_rk
        and bridge.account_rk=cc.account_rk) a;
```

Once you have entered the query, select . It is on the toolbar.




10. At this point we wait for few minutes. You can see the time that the query was run in the middle of the window. Do nothing until the query completes.

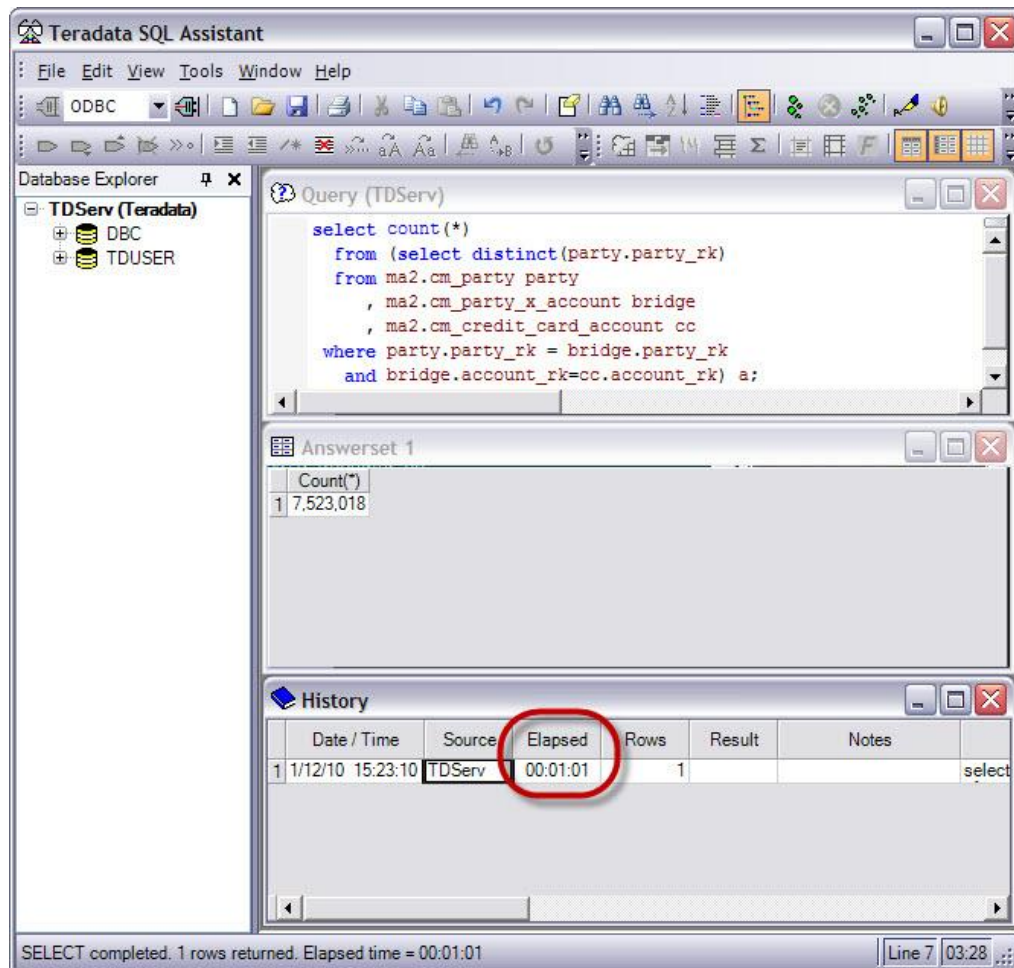


11. One of the nice things about the Teradata SQL Assistant is all the information that it shows you. You can see the query, the resultant set, and the history. The History pane displays the runtime. In this case 1 minute and 1 second. Teradata performs so well that it is hard to find long running queries in a test environment. Keep that in mind if you are creating production queries in a testing environment.

In addition to displaying out and the current SQL you are working with it saves all the queries you submit. This is quite useful because it allows you to place them in a script. It is a nice application. Remember, you can use it for any ODBC data source.

If you want to disconnect from the database, select .

If you want to close the application, select **File**→ **Exit** from the menus.



2.6 See how Teradata is executing the SQL query

So far we have seen how to determine the SQL that SAS is sending to Teradata. You cannot do a great deal of performance tuning without knowing the exact SQL statements that are being passed to the database.

In the current example we have an SQL query that is being generated by SAS and sent to Teradata. Usually the majority of your SAS programs execution time is being spent processing the data in Teradata; you can use BTEQ and Teradata SQL Assistant to prove that to the doubting DBA.

We know what SAS is sending to Teradata, but how do we know what Teradata is doing with the query? That is a good question. Fortunately, there is a simple answer.

2.6.1 The EXPLAIN SQL Statement

The EXPLAIN SQL Statement

The EXPLAIN statement displays a summary of the query plan that was generated by the query optimizer. One of the nice things about EXPLAIN is that it displays the query plan but does not execute the query. This is a great time saver. If your problem query is taking hours or days. Stick the EXPLAIN keyword in the statement, in goes right up front, and the statement returns in seconds.

You can only explain SQL statements that you have privileges to execute. For example, to execute the SQL statement below you must have SELECT privileges on the MA2.CM_PARTY, MA2.CM_PARTY_X_ACCOUNT and MA2.CM_CREDIT_CARD_ACCOUNT tables.

```
explain select count(*)
  from (select distinct(party.party_rk)
  from ma2.cm_party party
    , ma2.cm_party_x_account bridge
    , ma2.cm_credit_card_account cc
 where party.party_rk = bridge.party_rk
   and bridge.account_rk=cc.account_rk) a;
```

One of the nice things about the Teradata EXPLAIN utility is that it outputs a useful description of what the SQL optimizer is doing. It is quite easy to read. It can be hard to understand. Fortunately, you can show it to your DBA. Here is the Explanation from the above query.

Query Explanation Example:

- 1) First, we **lock** a distinct ma2."pseudo table" for **read** on a **RowHash** to prevent global deadlock for **ma2.bridge**.
- 2) Next, we **lock** a distinct ma2."pseudo table" for **read** on a **RowHash** to prevent global deadlock for **ma2.party**.
- 3) We **lock** a distinct ma2."pseudo table" for **read** on a **RowHash** to prevent global deadlock for **ma2.cc**.
- 4) We **lock ma2.bridge** for **read**, we **lock ma2.party** for **read**, and we **lock ma2.cc** for **read**.
- 5) We do an all-AMPs **RETRIEVE** step from **ma2.cc** by way of an all-rows scan with a condition of ("NOT (ma2.cc.ACCOUNT_RK IS NULL)") into **Spool 2** (all_amps) fanned out into 12 hash join partitions, which is **built locally** on the AMPs. The input table will not be cached in memory, but it is eligible for synchronized scanning. The size of **Spool 2** is estimated with high confidence to be **10,000,000 rows**. The estimated time for this step is **10.42 seconds**.
- 6) We do an all-AMPs **JOIN** step from **ma2.party** by way of a **RowHash match** scan with no residual conditions, which is joined to

ma2.bridge by way of a **RowHash match** scan with a condition of (**"NOT (ma2.bridge.ACCOUNT_RK IS NULL)"**). **ma2.party** and **ma2.bridge** are joined using a **merge join**, with a join condition of (**"ma2.party.PARTY_RK = ma2.bridge.PARTY_RK"**). The input tables **ma2.party** and **ma2.bridge** will not be cached in memory, but **ma2.party** is eligible for synchronized scanning. The result goes into **Spool 3** (**all_amps**), which is **redistributed** by hash code to all AMPs into 12 hash join partitions. The result spool file will not be cached in memory. The size of **Spool 3** is estimated with no confidence to be **31,050,000 rows**. The estimated time for this step is **1 minute and 21 seconds**.

- 7) We do an all-AMPs **JOIN** step from **Spool 2** (Last Use) by way of an **all-rows scan**, which is joined to **Spool 3** (Last Use) by way of an **all-rows scan**. **Spool 2** and **Spool 3** are joined using a **hash join** of 12 partitions, with a join condition of (**"ACCOUNT_RK = ACCOUNT_RK"**). The result goes into **Spool 1** (**all_amps**), which is **redistributed** by hash code to all AMPs. Then we do a **SORT** to order **Spool 1** by the sort key in spool field1 eliminating duplicate **rows**. The result spool file will not be cached in memory. The size of **Spool 1** is estimated with no confidence to be **31,050,000 rows**. The estimated time for this step is **30.04 seconds**.
- 8) We do an all-AMPs **SUM** step to aggregate from **Spool 1** (Last Use) by way of an **all-rows scan**. Aggregate Intermediate Results are computed globally, then placed in **Spool 6**. The size of **Spool 6** is estimated with high confidence to be **1 row**. The estimated time for this step is **1.21 seconds**.
- 9) We do an all-AMPs **RETRIEVE** step from **Spool 6** (Last Use) by way of an **all-rows scan** into **Spool 4** (**group_amps**), which is **built locally** on the AMPs. The size of **Spool 4** is estimated with high confidence to be **1 row**. The estimated time for this step is **0.00 seconds**.
- 10) Finally, we send out an **END TRANSACTION** step to all AMPs involved in processing the request.
 - > The contents of **Spool 4** are sent back to the user as the result of statement **1**. The total estimated time is **2 minutes and 3 seconds**.

2.6.2 Visual Explain

The **EXPLAIN** statement is not your only choice when trying to figure-out what the query optimizer is doing with your query. You can use Teradata Visual Explain. Teradata Visual Explain is a GUI application that draws pictures instead of writing English descriptions.

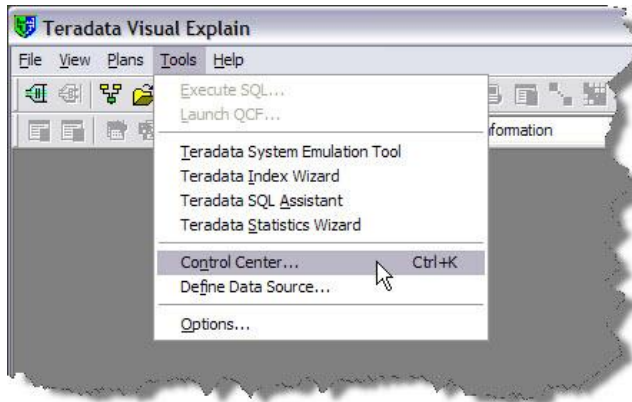
In order to use this Visual Explain you or your DBA will need to create a Query Capture Database. In order to create this database you must have sufficient privileges. You can have your DBA create the database for you. I will walk through the creation process for those of you who have the correct privileges.

Creating the Query Capture Database

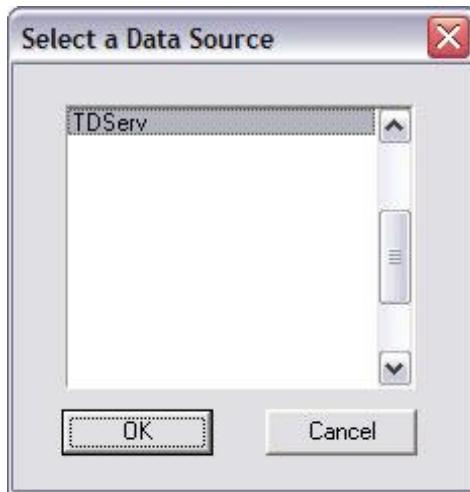
1. Invoke Teradata Visual Explain from the Start menu. Select **Start** → **All Programs** → **Teradata Visual Explain 13.0** (your version number might vary).



2. To create the QCD we need to bring-up the Control Center. Select **Tools** → **Control Center**.

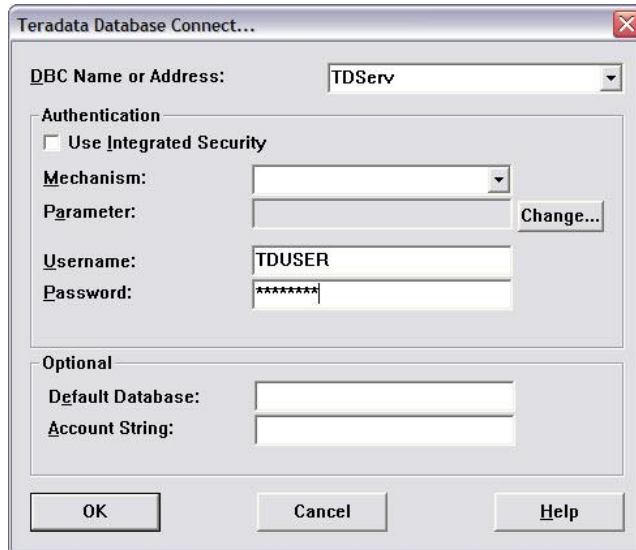


3. Select the Teradata server where the QCD needs to be created. In this example we will use TDServ. Select .

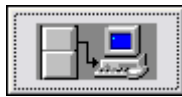


4. We must provide our password. Enter your password in the Password field. Select


OK



The 'Teradata Database Connect...' dialog box is shown. It has a title bar with a close button. The 'DBC Name or Address' is set to 'TDServ'. Under the 'Authentication' section, the 'Use Integrated Security' checkbox is unchecked. The 'Mechanism' is set to a dropdown menu. The 'Parameter' field is empty, with a 'Change...' button next to it. The 'Username' is 'TDUSER' and the 'Password' is masked with asterisks. There is an 'Optional' section with 'Default Database' and 'Account String' fields, both empty. At the bottom are 'OK', 'Cancel', and 'Help' buttons.



5. Setup QCD. Select



The 'Control Center' dialog box is shown. It has a title bar with a close button. The 'Manage QCD' tab is selected, showing 'Database Connectivity', 'Security', and 'Data Exchange' sub-tabs. The main area contains three icons: a server and computer icon (labeled 'Setup QCD'), a yellow cylinder icon (labeled 'Upgrade/Revert QCF Version'), and a yellow cylinder with a red arrow icon (labeled 'Cleanup QCD'). At the bottom are 'OK', 'Cancel', and 'Apply' buttons.

6. This panel can get a little complicated. If you need help contact your DBA.

First choose your QCD Name. I have chosen QCD + my user ID (**QCDDTDUSER**). The owner database should be named the same as your user ID. In this example I chose **tduser**. **Perm Space is set to 250 MB. Spool Space is set to 250 MB.** You might need to alter these values in your environment.

Make the changes and choose

Create

Setup QCD

Select

- ☒ Create all QCF Database objects
- ☐ Create QCF related views and macros

Teradata Database Information

QCF Version: QCF03.01.00 View Schema...

QCD Name: QCDDTDUSER

Owner: TDUSER

Perm Space: 250 KB MB GB

Spool Space: 250 KB MB GB

☒ Fallback

Create Modify Perm Clear Command... Close


7. At this point you will see the database objects which are being created and a status bar. Wait until the process completes. The window will go away on its own.

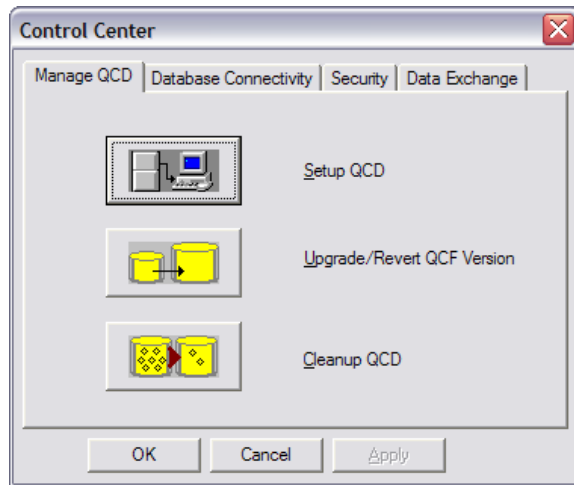
Setup in progress...

Creating view 'FieldViewX' of 'QCF03.01.00' version...

Time Elapsed: 00:00:05

Abort

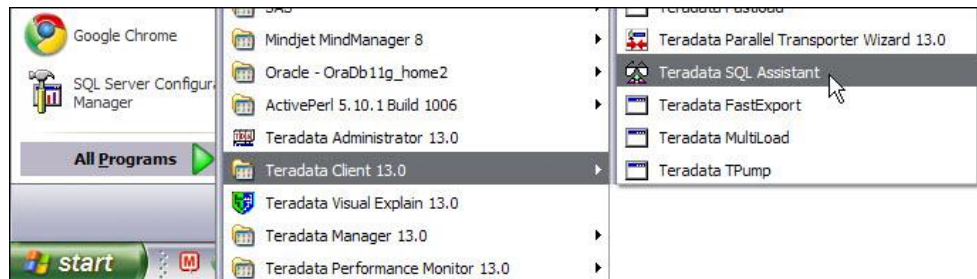
8. You will see the Control Center dialog box. Select .



9. At this point you can use Teradata Visual Explain to look at your query's access path. Let's do that now. It is easier to invoke this tool from Teradata SQL Assistant. Shutdown Teradata Visual Explain. Select **File** → **Exit**.

When asked if you really want to quit, select **Yes**.

12. Invoke Teradata SQL Assistant via the Start button. **Start** → **All Programs** → **Teradata Client 13.0** → **Teradata SQL Assistant** (for a reminder of how to connect to Teradata refer back to 2.5.2 Executing a Query with Teradata SQL Assistant on page 17).

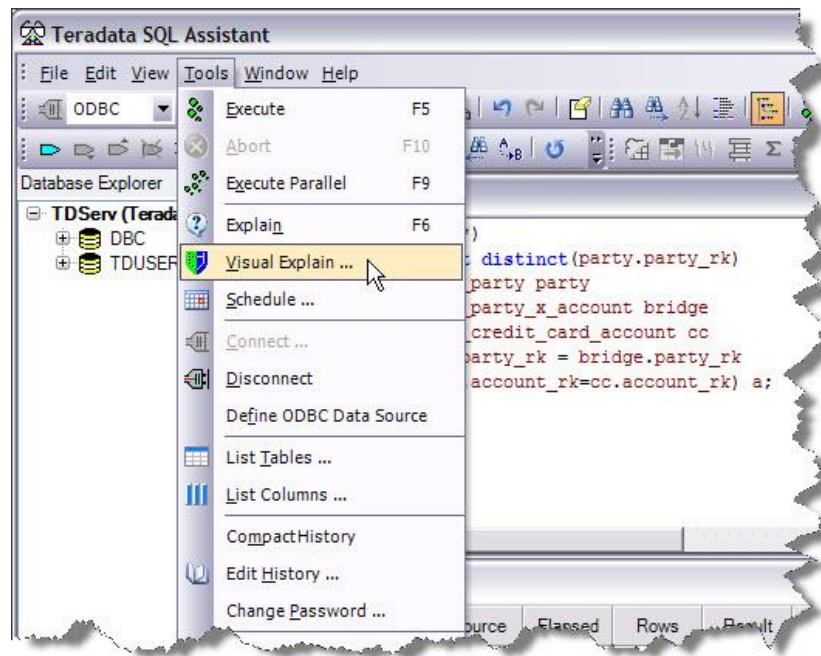


13. Let's take the query we ran earlier (in Teradata SQL Assistant) and enter it into the Query pane.

Here is our example query.

```
select count(*)
  from (select distinct(party.party_rk)
        from ma2.cm_party party
             , ma2.cm_party_x_account bridge
             , ma2.cm_credit_card_account cc
       where party.party_rk = bridge.party_rk
             and bridge.account_rk=cc.account_rk) a;
```

14. Once you have entered the query, select **Tools** → **Visual Explain** from the Menu bar.



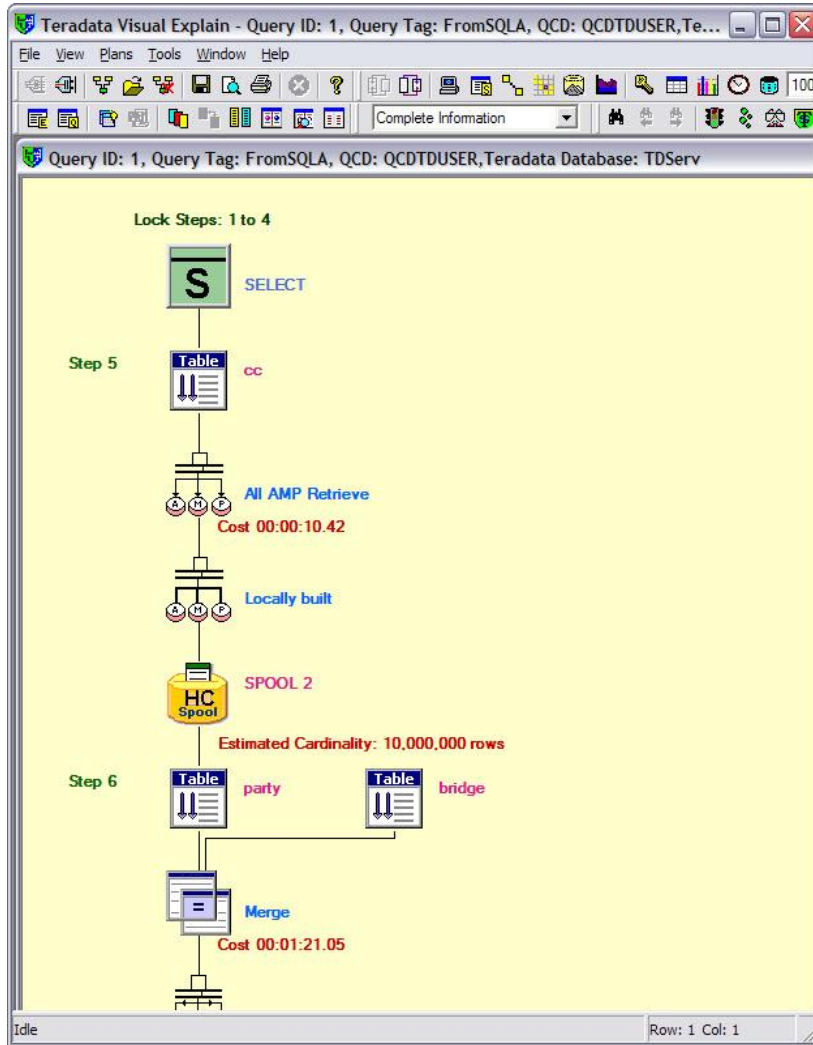
10. Enter the name of your QCD, in this example it is QCDDUSER. Select .



11. Teradata Visual Explain is submitting the query to Teradata so that the plan can be generated.



12. You should see a nice picture of the steps that Teradata will go through while executing your query.



13. Feel free to close Teradata Visual Explain.

2.6.3 Explain Plan Recommendations

You should use the EXPLAIN facility often. You can learn a lot from it. Typically, you are looking for table scans and all-AMP processing when dealing with performance problems. It is quite useful to show the output from the EXPLAIN of problem queries to your DBA. Your DBA will know what to do.

Interpreting the output from Teradata Visual Explain is beyond the scope of this paper. Please look at the Teradata documentation for more information.

2.7 Collect statistics

The execution plan shows how the query optimizer will execute your SQL queries. The real question is, “How does the query optimizer determine which access path to use?” The answer is “statistics.”

Statistics about the tables, columns, and indexes in your databases are critical in obtaining great query plans. There are at least three approaches to collecting statistics. Please include collecting statistics in your implementation plans.

Three approaches to collecting statistics on your Teradata objects:

- Random AMP Sampling
- Full Statistics collection
- Random sampling with the USING SAMPLE option.

These techniques are covered in the Teradata documentation. We are not going to reproduce that documentation here; we are going to cover the highlights. In order to provide maximum performance you will likely need to use a mix of these techniques. You should expect to experiment with these various techniques – your DBA can help you make the final decision.

2.7.1 Random AMP Sampling

An AMP is an Access Module Process. It is an executing process that is tasked with accessing your Teradata data. Your tables will be split among multiple AMPs inside the Teradata database server. This is what makes Teradata so massively parallel – and fast. A Teradata node is a computer on which your Teradata system runs. You will have many AMPs running on each Teradata node.

Random AMP sampling helps you in those cases where you do nothing. If you don’t explicitly collect statistics on your database objects, random AMP sampling will be used.

When you submit your SQL for execution the optimizer will look for statistics. If it doesn’t find any, or if they are too old, then random AMP sampling will be used. Random AMP sampling is a high-level estimate made by randomly sampling the contents controlled by one AMP. Remember, each table is managed by many AMPs. You can control how many AMPs are sampled by this process. It is possible to use all the AMPs for this estimate. Your data will determine how effective the estimates are and if adding AMPs to the sample will help or hurt. Contact your DBA for more information.

The key take away here is that if you do nothing Teradata will try to help you. In many cases, this might be enough. But, why take a chance. If random AMP sampling suffices for your system, please state so explicitly in your project documentation.

2.7.2 Full Statistics Collection

Random AMP sampling leaves the collection of statistics up to Teradata. Believe me when I say, “Teradata is going to try its best to make things run super-fast.” But things are not perfect; you are likely to find that this method of collecting statistics does more harm than good. Fortunately, you and your DBA can take control of the situation and collect the specific statistics that help you get those great query plans that perform very fast.

The most complete method of collecting statistics is using the `COLLECT STATISTICS` statement. Using this statement you can collect demographic information about a specific column or index. There is a huge difference between using `COLLECT STATISTICS` and random AMP sampling. You must explicitly run the `COLLECT STATISTICS` statement, it will not happen automatically.

Collecting full statistics involves scanning tables and performing sorts. This makes sense since one of the most important statistics is cardinality (number of distinct values found in a column). This is the most precise way of gathering statistics, but it comes at with a rather large performance price tag.

One of the most common problems found in DBMSs is stale statistics. A stale statistic is one where the data no longer resembles what is actually stored in the table. Here is an example. You have a table with no rows of data. You collect statistics on that table using the `COLLECT STATISTICS` statement. Now you insert 100 Million rows of data into the table. The statistics will still indicate that the table is empty. You can expect poor query performance.

The statements that we are going to look at can be issued in Teradata BTEQ or Teradata SQL Assistant.

Here is an example `COLLECT STATISTICS` statement:

```
COLLECT STATISTICS ON myTable COLUMN (myColumn) ;
```

You can even run an `EXPLAIN` on the `COLLECT STATISTICS` statement:

```
EXPLAIN COLLECT STATISTICS ON myTable COLUMN (myColumn) ;
```

You can collect statistics on indexes:

```
COLLECT STATISTICS ON myTable INDEX (myKey) ;
```

Teradata knows when a column is unique. The optimizer will assume an equality predicate (`myUniqueCol = 'A Unique Value'`) only returns one row. This means that statistics collected on these columns have little impact on the optimizer. On the other hand, it is very important to collect statistics on non-unique indexes and columns commonly included in `WHERE` clauses.

It is sometimes useful to collect statistics on multiple columns and treat them as a group. This can help make queries with multiple predicates run much faster. For example:

```
select count(*)
  from myTable
 where color = 'Blue'
    and material = 'Burlap' ;
```

This query may benefit from this COLLECT STATISTICS statement.

```
COLLECT STATISTICS ON myTable COLUMN (color, material) ;
```

If you have statistics that no longer reflect the contents of your table, you can drop them. Here is an example of dropping the statistics collected with the previous COLLECT STATISTICS statement.

```
DROP STATISTICS ON myTable COLUMN (color, material) ;
```

If you would like to review which statistics have been collected for a table, you can use this command.

```
HELP STATS myTable ;
```

The HELP STATS statement will display the column names; the date and time the statistics were collected; and the number of unique values contained in the columns.

Once you have collected the statistics for a table you can easily refresh them. I suggest that you issue the HELP STATS statement to see which columns have statistics before relying on this. Here is the command that will recollect statistics on the myTable table.

```
COLLECT STATISTICS myTable ;
```

It doesn't get any easier than this. But do not let "easy" get you into trouble. It would be "easy" to think that you are collecting the proper statistics using this simple command. It is also "easy" to be wrong. Poor statistics yield poor performance.

Using Teradata tools is fine, but SAS folks like to use SAS. Plus, you might want to use code like this in your SAS ETL job streams. Don't worry; it is easy to do this in SAS. There is a trick that you need to know. COLLECT STATISTICS and DROP STATISTICS are data manipulation language statements. Since we are using ANSI mode, we will need to issue COMMIT statements after running these commands. Examine the SAS/ACCESS Interface to Teradata documentation for a description of the differences between ANSI and Teradata modes.

Here is an example. Let's collect statistics, display the statistics, drop statistics and finally display the stats again to verify that the statistics for the `tduser.cm_financial_account` table were dropped. Pay special attention to the COMMIT statements. Here is the SAS explicit Pass-Through code:

```
PROC SQL;
  connect to teradata (user=tduser pw=tdpasswd server=TDserv);
  execute (COLLECT STATISTICS ON ma2.cm_party
          COLUMN(PARTY_RK)) by teradata ;
  execute (COMMIT) by teradata ;
  select * from connection to teradata
    (help stats ma2.cm_party) ;
  execute (COMMIT) by teradata ;
  execute (DROP STATISTICS ON ma2.cm_party
          COLUMN (PARTY_RK)) by teradata ;
  execute (COMMIT) by teradata ;
  select * from connection to teradata
    (help stats ma2.cm_party) ;
QUIT;
```

Here is the output produced by this code.

The SAS System 13:35 Wednesday, January 13, 2010			
Date	Time	Unique Values	Column Names
10/01/13	13:46:08	20,000,000	PARTY_RK

The second SELECT statement doesn't produce output because there are no stats; we dropped them in a previous EXECUTE step. Look in the SAS log for the message regarding the lack of statistics.

```
29 PROC SQL;
30   connect to teradata (user=tduser pw=XXXXXXXXX server=TDserv);
31   execute (COLLECT STATISTICS ON ma2.cm_party
32           COLUMN(PARTY_RK)) by teradata ;
33   execute (COMMIT) by teradata ;
34   select * from connection to teradata
35     (help stats ma2.cm_party) ;
36   execute (COMMIT) by teradata ;
37   execute (DROP STATISTICS ON ma2.cm_party
38           COLUMN (PARTY_RK)) by teradata ;
39   execute (COMMIT) by teradata ;
40   select * from connection to teradata
41     (help stats ma2.cm_party) ;
ERROR: Teradata prepare: There are no statistics defined for the table. SQL statement was: help
stats ma2.cm_party.
42 QUIT;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE SQL used (Total process time):
      real time          43.39 seconds
      cpu time           0.07 seconds
```

Be sure to discuss collecting statistics with your Teradata DBA. There is a great Orange Book, written by Carrie Ballenger of Teradata, on the Teradata Web site. The name of the book is: Teradata Orange Book: Collecting Statistics.

2.7.3 Random sampling with the USING SAMPLE option

Full statistics require scanning and sorting the contents of entire tables. It could take a while to collect statistics using this method. There might be times that we don't need to know exactly what is in the table, an educated guess would suffice. For those times we have the USING SAMPLE option.

Teradata will determine how to sample the data. It uses different techniques depending on if the column is not indexed, primary index or secondary index.

2.7.3.1 USING SAMPLE option – Column Not Indexed

A simple scan of the first 2% of the data blocks is used.

2.7.3.2 USING SAMPLE option – Primary Index

USING SAMPLE samples the actual rows in the table for a primary index. It intends to sample 2% of the blocks. It will skip blocks and then sample more rows. There is a formula that is used to determine how many blocks to skip. The percentage of blocks sampled will be increased if skew is detected. Skew is a disproportionate clustering of values. For example, having February represented 1,000,000 times while the other 11 months only occur once.

2.7.3.3 USING SAMPLE option – Secondary Index

USING SAMPLE samples the index subtable (index itself) for a secondary index. Once again 2% of the Non Unique Secondary Index (NUSI) subtable is checked. This represents 2% of the distinct values within the index are checked. The same approach is used for Unique Secondary Indexes (USI).

2.7.4 Statistics Collection Recommendations

To get great performance from Teradata, or any other DBMS for that matter, you must provide the query optimizer with the information it needs to create a great execution plan. It is highly likely that you will use a combination of the three approaches. You should discuss this with your DBA.

Keep in mind, the strategy that provides the best result can change as your project progresses. Don't assume that your initial design will work for the duration of the project.

Your statistics collection strategy should be noted in your project documentation. It doesn't matter if you choose to implement your strategy using SAS or the Teradata tools. What is important is that you design a working strategy, document the strategy and implement the strategy.

2.8 Hot AMPing

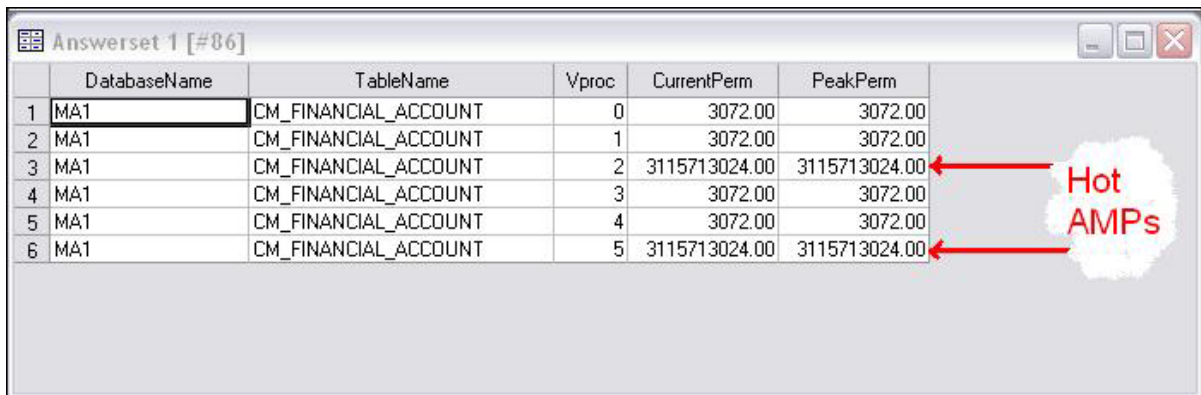
Teradata distributes data rows across multiple AMPs using hash buckets. This divides the data so that it can be acted upon in parallel. If you choose a poor index you may get terrible performance.

It falls apart because of “hot AMP’ing.” Gee, that sounds cool! Hot AMP’ing is when you have one AMP working a lot harder than it should. When those rows were placed into the hash buckets one of those buckets got a lot more rows than the others. This is caused by an index that isn’t evenly distributed. In other words, you have skewed data. The Teradata Performance Management manual calls it “lumpy data.” I like that.

To determine if you have lumpy data you must look into the Teradata dictionary tables. Let’s say that you have a table named CM_FINANCIAL_ACCOUNT in the MA1 database and you want to see how the space is allocated. Here is an SQL query that will show you how the space is allocated. This query was run using Teradata SQL Assistant.

```
SELECT DatabaseName
      , TableName
      , Vproc
      , CurrentPerm
      , PeakPerm
FROM DBC.TableSize
WHERE DatabaseName= 'MA1'
      AND TableName=' CM_FINANCIAL_ACCOUNT'
ORDER BY Vproc;
```

This is the resultant set from that query.



	DatabaseName	TableName	Vproc	CurrentPerm	PeakPerm
1	MA1	CM_FINANCIAL_ACCOUNT	0	3072.00	3072.00
2	MA1	CM_FINANCIAL_ACCOUNT	1	3072.00	3072.00
3	MA1	CM_FINANCIAL_ACCOUNT	2	3115713024.00	3115713024.00
4	MA1	CM_FINANCIAL_ACCOUNT	3	3072.00	3072.00
5	MA1	CM_FINANCIAL_ACCOUNT	4	3072.00	3072.00
6	MA1	CM_FINANCIAL_ACCOUNT	5	3115713024.00	3115713024.00

Here is an example SAS program which does the same thing.

```
LIBNAME DBC teradata user=tduser password=tdpasswd
          schema='DBC' server=TDServ;

PROC SQL;

    SELECT DatabaseName, TableName, Vproc, CurrentPerm, PeakPerm
    FROM DBC.TableSize
    WHERE DatabaseName = 'MA1'
    AND TableName='CM_FINANCIAL_ACCOUNT'
    ORDER BY Vproc;

QUIT;
```

Here is the output from the SAS program.

The SAS System 09:50 Friday, February 22, 2008 2				
DatabaseName	TableName	Vproc	CurrentPerm	PeakPerm
MA1	CM_FINANCIAL_ACCOUNT	0	3072	3072
MA1	CM_FINANCIAL_ACCOUNT	1	3072	3072
MA1	CM_FINANCIAL_ACCOUNT	2	3.1157E9	3.1157E9
MA1	CM_FINANCIAL_ACCOUNT	3	3072	3072
MA1	CM_FINANCIAL_ACCOUNT	4	3072	3072
MA1	CM_FINANCIAL_ACCOUNT	5	3.1157E9	3.1157E9

Notice the same hot AMPs.

If you want to report on all the tables in a database then you can use this SQL query.

```
SELECT DatabaseName
       , TableName
       , Vproc
       , CurrentPerm
       , PeakPerm
FROM DBC.TableSize
WHERE DatabaseName= 'MA1'
ORDER BY TableName, Vproc;
```

Here is the SAS code that will do the same thing.

```
LIBNAME DBC teradata user=tduser password=tdpasswd
          schema='DBC' server=TDServ;

PROC SQL;

    SELECT DatabaseName, TableName, Vproc, CurrentPerm, PeakPerm
    FROM DBC.TableSize
    WHERE DatabaseName = 'MA1'
    ORDER BY TableName, Vproc;

QUIT;
```

3 Summary

3.1 Summary of Lessons Learned

Great SAS and Teradata performance is within your grasp. By using database tools and SAS options you can learn how your SAS code is performing; this will help enable you to get great performance.

You can use a mix of SAS and Teradata tools and facilities in your quest for great query performance. Here are some of the things we discussed:

- SASTRACE=, SASTRACELOC= and NOSTSUFFIX options and how you can use them do see the SQL being sent from SAS to Teradata
- using Teradata BTEQ and Teradata SQL Assistant to prove, to a DBA, that Teradata is having issues with query performance
- using Teradata EXPLAIN PLAN to verify that our queries are being executed efficiently by the database
- the importance of collecting statistics on the database objects and how they can impact performance
- the importance of having data evenly spread over the database AMPs – choosing the right primary index

The biggest challenge that you will face in using these ideas come from the DBAs at your customer site. If they won't allow you to use the tools it will limit your ability to verify and confirm problems. On the other hand, simply knowing about these things will help you on your path to outstanding SAS and Teradata performance. I can't stress that enough. What you learn from reading and studying these ideas will make you more effective in your dealings with the customer's DBA.

Plus, great SAS and Teradata performance leads to well delivered projects and happy customers.

4 Bibliography

Teradata Orange Book: Collecting Statistics – Carrie Ballinger

Teradata Database Performance Management, Release 12.0, B035-1097-067A, October 2007

5 Credits and Acknowledgements

Pravin Boniface, SAS

Allen Cunningham, SAS

Teri Huels, SAS

Brian Mitchell, Teradata

Greg Otto, Teradata

Austin Swift, SAS

Christine Vitron, SAS

SAS INSTITUTE INC. WORLD HEADQUARTERS SAS CAMPUS DRIVE CARY, NC 27513
TEL: 919 677 8000 FAX: 919 677 4444 U.S. SALES: 800 727 0025 **WWW.SAS.COM**



**THE
POWER
TO KNOW®**

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © 2006, SAS Institute Inc.

All rights reserved. 410703.0906