



---

Global Business Intelligence and Data Integration Practice  
**Effectively Moving SAS Data into Teradata**

Document Type: Best Practice

Date: February 1, 2010

---

**Contact Information**

Name: Jeffrey D. Bailey

Title: Database Technical Architect

Phone Number: (919) 531-6675

E-mail address: [Jeff.Bailey@sas.com](mailto:Jeff.Bailey@sas.com)

**THE  
POWER  
TO KNOW®**

# Revision History

Version	By	Date	Description
1.0	Jeffrey D. Bailey	July 11, 2008	Initial Version
2.0	Jeffrey D. Bailey	Feb 1, 2010	Removed Confidential Information Added a case study Added performance statistics

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Purpose of the Paper .....	1
1.2	Target Audience .....	1
1.3	Overview .....	1
1.4	Business Problem(s) Addressed .....	1
1.5	Technical Problem(s) Addressed .....	2
1.6	Environment Requirements .....	2
<b>2</b>	<b>Connecting to Teradata (Review) .....</b>	<b>3</b>
2.1	Verify that you can connect to Teradata Using SAS .....	3
2.2	Connecting to Teradata Using SAS® Foundation.....	7
<b>3</b>	<b>Creating Teradata Tables– A Word of Warning ....</b>	<b>8</b>
3.1	Teradata Parallelism – By Design .....	8
3.2	Primary Indexes .....	8
3.3	Why Is This a Problem with SAS .....	9
3.4	Specify the Primary Index Using SAS .....	12
<b>4</b>	<b>Teradata FastLoad and MultiLoad Utilities.....</b>	<b>13</b>
4.1	SQL Insert Statements.....	13
4.2	The Teradata FastLoad Utility .....	18
4.3	Loading Data Using the FastLoad Utility .....	20
4.4	The Teradata MultiLoad Utility .....	21
4.5	Loading Data Using the MultiLoad Utility .....	22
<b>5</b>	<b>Loading SAS Data Into Teradata (FastLoad) .....</b>	<b>26</b>
5.1	FastLoad Protocol Restrictions .....	26
5.2	SAS Bulkloading Using Teradata FastLoad.....	27

<b>6</b>	<b>Loading SAS Data into Teradata (MultiLoad) .....</b>	<b>37</b>
6.1	MultiLoad Protocol Restrictions .....	37
6.2	Configuring SAS for MultiLoad .....	38
<b>7</b>	<b>A Real Case Study .....</b>	<b>64</b>
<b>8</b>	<b>Bibliography .....</b>	<b>68</b>
<b>9</b>	<b>Credits and Acknowledgements .....</b>	<b>69</b>

# 1 Introduction

## 1.1 Purpose of the Paper

Teradata is one of the most robust and best performing relational database management systems (RDBMS) available. Its performance with huge volumes of data is amazing. Some of that data will definitely come from SAS. But, what is the best way to move that SAS data into the Teradata server? This paper will explore that question. Fortunately, SAS is able to use the Teradata utilities that make loading Teradata tables from SAS fast and easy.

## 1.2 Target Audience

The target audience for this paper is SAS Intelligence Platform administrators, SAS programmers, and consultants who need to move SAS data into Teradata.

## 1.3 Overview

Efficiently loading Teradata from SAS is not very difficult, it certainly isn't magic, but you need to be aware of some traps. We will cover the various loading methods available and help you decide which of these methods best meets your needs.

We will:

- Learn about the Teradata supplied load functionality – FastLoad and MultiLoad.
- Learn how to leverage the Teradata FastLoad protocol from within SAS.
- Learn how to leverage the Teradata MultiLoad protocol from within SAS.
- Learn to avoid adding entries to the Teradata Journal Table.
- Learn which Teradata objects will prohibit you from using Teradata load protocols.

## 1.4 Business Problem(s) Addressed

The business problem is straight forward. We have large amounts of data that we need to move into Teradata. We want to load this data as efficiently as possible. This paper will provide you with options on how to do this effectively.

## 1.5 Technical Problem(s) Addressed

The technical issues are going to fall into a couple of categories. These categories are:

- Which Teradata load protocol do we use to load our data?
- How do Teradata indexes affect our choice of load protocol?

## 1.6 Environment Requirements

In order to follow the troubleshooting guidelines we are recommending you 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 that will allow you to load data into the tables.
- SAS 9.2 (or 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.

If you do not have everything listed above, do not despair. Understanding the contents of this paper will allow you to request that the DBA do some of this for you. 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. Even if you do not have this knowledge, 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 or UNIX client machine, SAS Enterprise Guide (via a SAS Workspace Server or SAS Stored Process Server) or BASE SAS, the SAS/ACCESS Interface to Teradata, a Teradata server that you can access 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 Connecting to Teradata (Review)

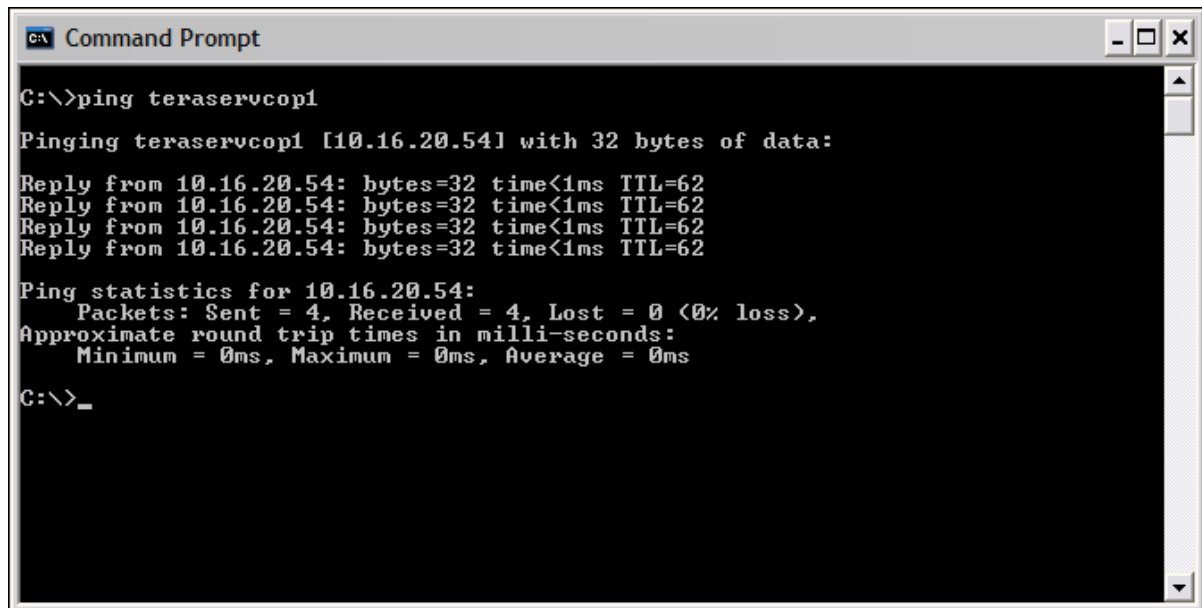
### 2.1 Verify that you can connect to Teradata Using SAS

There are many ways that you can connect to Teradata. You can use Teradata utilities, BASE SAS, SAS Enterprise Guide (via a SAS Workspace Server or SAS Stored Process Server) or SAS Management Console (via a SAS Workspace Server). We 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 Domain Name System (DNS) entries or in the client machine's hosts file. To see if the entry exists in your DNS server you can use the ping command. For example, if our Teradata server is on `teraservcop1` we can issue the ping command.



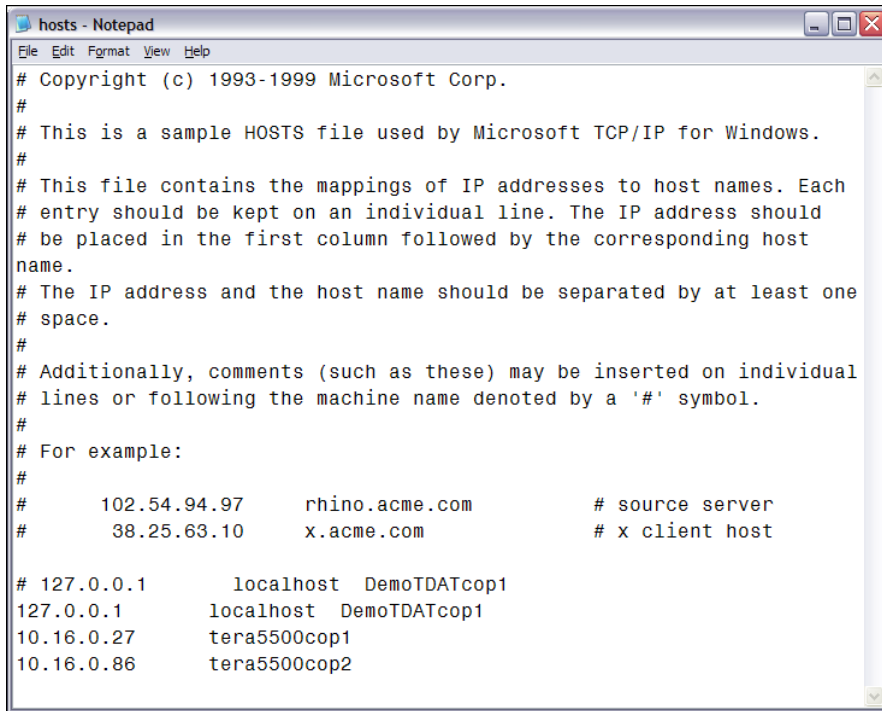
```
C:\>ping teraservcop1

Pinging teraservcop1 [10.16.20.54] with 32 bytes of data:
Reply from 10.16.20.54: bytes=32 time<1ms TTL=62
Reply from 10.16.20.54: bytes=32 time<1ms TTL=62
Reply from 10.16.20.54: bytes=32 time<1ms TTL=62
Reply from 10.16.20.54: bytes=32 time<1ms TTL=62

Ping statistics for 10.16.20.54:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>_
```

If you cannot ping the server you may want to have your DNS administrator add an entry for your server to the DNS server. In the meantime you can add the entry to the hosts file on your client machine. This will allow work to proceed. On UNIX the hosts file is typically located in the `/etc` directory. On Windows the hosts file is located in the `C:\WINNT\system32\drivers\etc` directory. Open the file in Notepad.



```

hosts - Notepad
File Edit Format View Help
# 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 DemoTDATcop1
127.0.0.1       localhost DemoTDATcop1
10.16.0.27      tera5500cop1
10.16.0.86      tera5500cop2

```

The Teradata server that we will be using for this discussion is tera5500. Notice that there are two references to this server. This is a multi-node (multi-machine) Teradata environment. We do not need to use the “cop1” or “cop2” extensions when addressing the Teradata Server in the Teradata utilities. Teradata will handle balancing the connections between the multiple servers. When we need to specify a Teradata server in a Teradata utility or SAS we will use tera5500.

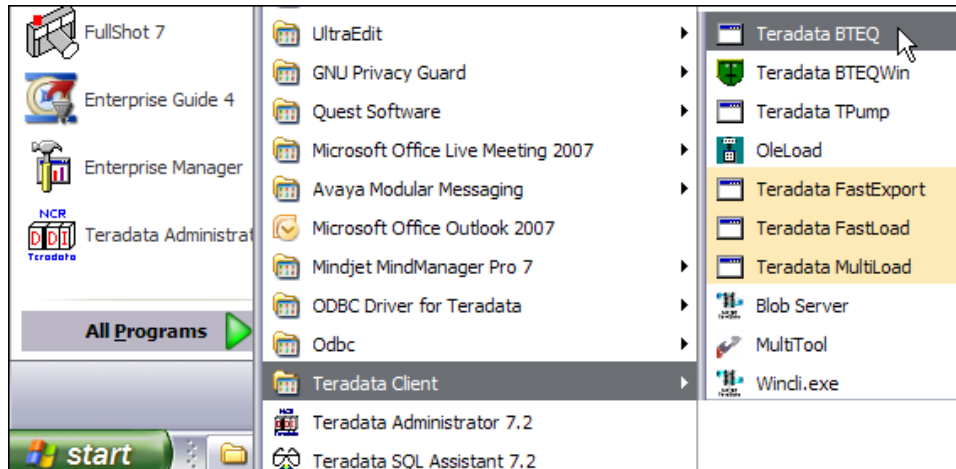
**Note: tera5500 is the Teradata server that we will use in this discussion.**

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? It is a good idea to test the Teradata Clients that have recently been installed or have been recently reconfigured. 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 a Teradata utility (Basic Teradata Query (BTEQ) or Teradata SQL Assistant) for this test. Teradata SQL Assistant uses ODBC for its connection; BTEQ uses the Call Level Interface (CLI).



The SAS/ACCESS Interface to Teradata is a CLI client application, so BTEQ is the best choice for the test. Here is an example.

1. Select **Teradata BTEQ** via the **Start Button**. Start → All Programs → Teradata Client → 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 tera5500 and the user ID is bailey. The password is a secret.

Here is an example of connecting. Enter the LOGON command, **.logon tdserv/tduser**. You will be prompted for your password. This screenshot shows a successful connection.

```

C:\>Command Prompt - bteq

C:\>bteq

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

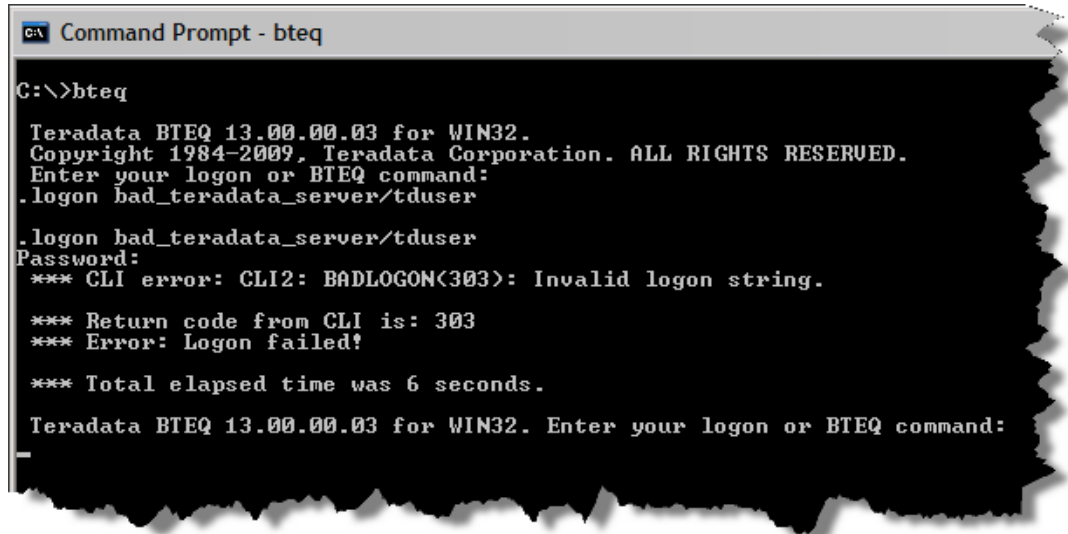
.logon tdserv/tduser
Password:

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

*** Total elapsed time was 2 seconds.

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

Here is an example of a failed connection. The Teradata server value, `bad_teradata_server`, is incorrect.



```

C:\>bteq

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
Password:
*** CLI error: CLI2: BADLOGON(303): Invalid logon string.

*** Return code from CLI is: 303
*** Error: Logon failed!

*** Total elapsed time was 6 seconds.

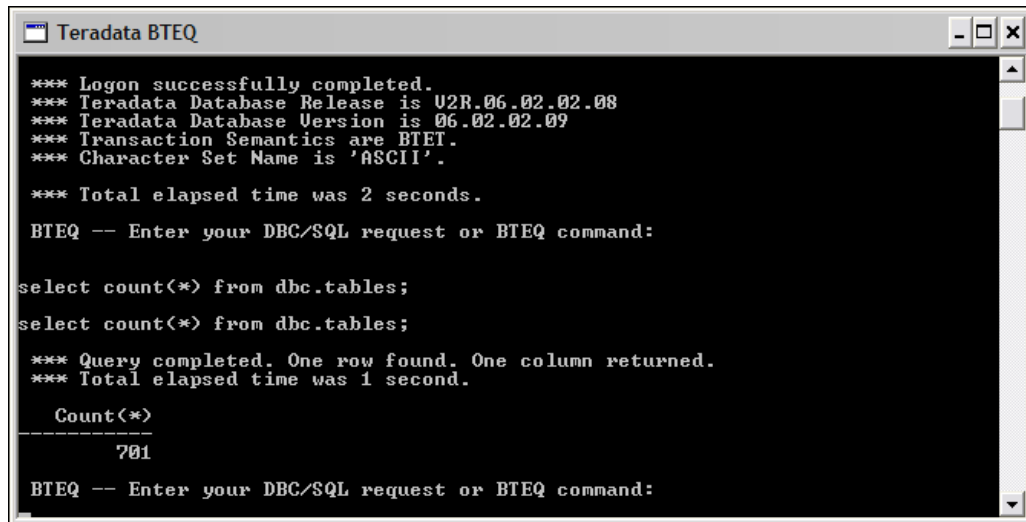
Teradata BTEQ 13.00.00.03 for WIN32. Enter your logon or BTEQ command:

```

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.

Enter the following query and press the Enter key:

```
select count(*) from DBC.Tables;
```



```

Teradata BTEQ

*** Logon successfully completed.
*** Teradata Database Release is U2R.06.02.02.08
*** Teradata Database Version is 06.02.02.09
*** Transaction Semantics are BTEI.
*** Character Set Name is 'ASCII'.

*** Total elapsed time was 2 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(*)
-----
701

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

```

If you don't have access to the `dbc.tables` table you may want to try querying the `DBC.TablesX` views. You can exit Teradata BTEQ by typing `exit` (notice the period in front of the command) and then pressing the Enter key.

## 2.2 Connecting to Teradata Using SAS® Foundation

We recommend testing the connection via BASE SAS for new SAS Foundation installs or when an upgrade has been performed. We are going to make sure that the SAS/ACCESS Interface to Teradata has been installed and configured properly. The SAS/ACCESS software needs to be installed on any machine that will be using BASE SAS, a SAS Workspace Server, or a SAS Stored Process Server to access data stored in a Teradata Server.

We can do that by submitting code similar to this (note: TDPID= is an alias for SERVER=. They can be used interchangeably).

```
LIBNAME mytera TERADATA USER=bailey PASSWORD=mypassword TDPID=tera5500;
```

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

```
5 LIBNAME mytera TERADATA USER=bailey PASSWORD=XXXXXX TDPID=tera5500;  
NOTE: Libref MYTERA was successfully assigned as follows:  
      Engine:          TERADATA  
      Physical Name: tera5500
```

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 register a SAS Library for your Teradata Server in SAS Management Console. If you have a problem doing this, you will want to run through client and SAS LIBNAME statement tests again. It is much easier to determine the problem via Teradata BTEQ or SAS Foundation than it is using SAS Management Console.

## 3 Creating Teradata Tables– A Word of Warning

### 3.1 Teradata Parallelism – By Design

In Teradata the rows of every table are distributed to all the access module processes (AMP) in the Teradata environment. AMPs are responsible for a subset of the data. This leads to a highly parallelized architecture and is one of the reasons that it is so fast; when you run a query the AMPs are able to read data in parallel. When you insert or update data the AMPs write in parallel. Teradata is designed so that no one AMP is required to read a huge amount of data. It is a classic “divide and conquer” scenario.

One of the goals that Teradata users should have is to evenly distribute data amongst the AMPs in the Teradata Server. Evenly distributed data typically results in evenly distributed workloads. This is a big deal. If you have a poorly performing table, you will definitely want to look into this.

What can users do to ensure that the data is evenly distributed amongst the AMPs? They can make sure that they chose a primary index that distributes the rows of data evenly across the AMPs .

### 3.2 Primary Indexes

Choosing a good primary index is very important in Teradata. If you chose a well distributed primary index then your rows will be evenly distributed across the AMPs. If you choose a poorly distributed primary index then you will have some AMPs responsible for many rows and others that may not be responsible for any.

The primary index is defined when you create the table. You can have a single column or multiple column primary index. The maximum number of columns in your primary index depends upon the version of Teradata that you are using. Recent versions of Teradata allow up to 64 columns in a primary index. Typically, you will not put that many columns in your primary index.

A primary index is used for both accessing data and choosing where to store data. The primary index that you choose for the table should map to the typical way the data will be accessed. It defines the most efficient path used to access the rows in the table. The primary index values are hashed and the hash value is used to determine which AMP that has control of the row.

If you create a table and do not specify a primary index then Teradata will determine which column to use as the primary index. Teradata will choose a primary index whether you specify one or not.

Here is the process:

If there is no primary index defined for the table then the primary key defined for the table will be used to create a unique primary index. If there is no primary key defined on the table then Teradata

will look for a column defined as unique. It will use the first one that it finds. If there is no primary key or unique column defined on the table then Teradata will use the first column in the table.

Letting Teradata choose your primary index is usually not a great idea. Remember, data rows are distributed across the AMPs based on the hash value of the primary index.

If a primary index is created on a column that has poor distribution then two things may happen:

One, you may encounter space problems while loading your table. This occurs because a few AMPs may be overloaded with data. This will cause the allotted space for the AMP to fill up. If you have space issues while loading your tables this is one of the leading causes.

Second, the primary index should be the most effective method of accessing the data in your table. If a poorly distributed primary index is chosen then some AMPs will be working very hard and other may not be working at all. Query performance is typically dictated by the slowest AMP.

### 3.3 Why Is This a Problem with SAS

SAS makes it very easy to copy a table from one Teradata Server onto another. Let's assume that we have a table named CM\_FINANCIAL\_ACCOUNT on the Teradata Server MATERA and we need to recreate it on TDServ. We run our code to create the new table and load data into it but we run out of space. This happens even though there is more than enough free space allocated to us in the database.

```
LIBNAME MATERA TERADATA USER=myid PASSWORD=mypassword SERVER=MATERA;
LIBNAME TDServ TERADATA USER=myid PASSWORD=mypassword SERVER=TDServ;

proc append base=TDServ.cm_financial_account (bulkload=yes)
            data=MADATA.cm_financial_account;
run;
```

After encountering the space issue it is a good idea to check the data distribution on the source Teradata system. We want to check to see how the data is distributed across the AMPs. This query uses the DBC.TableSize table to report on space used by SASDB1.CM\_FINANCIAL\_ACCOUNT table.

```
SELECT DatabaseName
       , TableName
       , Vproc
       , CurrentPerm
       , PeakPerm
FROM DBC.TableSize
WHERE DatabaseName= 'SASDB1'
      AND TableName=' CM_FINANCIAL_ACCOUNT'
ORDER BY Vproc;
```

The resultant set from the query clearly shows that AMPs 2 and 5 have much more data than the others. The distribution of primary key values is definitely skewed. This skewing was causing the load to fail.

Space by AMP - MA1.CM_FINANCIAL_ACCOUNT				
	TableName	Vproc	CurrentPerm	PeakPerm
1	CM_FINANCIAL_ACCOUNT	0	3,072	3,072
2	CM_FINANCIAL_ACCOUNT	1	3,072	3,072
3	CM_FINANCIAL_ACCOUNT	2	3,115,713,024	3,115,713,024
4	CM_FINANCIAL_ACCOUNT	3	3,072	3,072
5	CM_FINANCIAL_ACCOUNT	4	3,072	3,072
6	CM_FINANCIAL_ACCOUNT	5	3,115,713,024	3,115,713,024

If you would like to see how many rows are assigned to each AMP you can issue the following query (change the value **Your\_Primary\_Index\_Columns** to the columns included in the primary index defined on your table).

```
SELECT hashamp(hashbucket(hashrow(Your_Primary_Index_Columns))) as "AMP"
, count(*) as "Row Count"
FROM Your_Table_Name
GROUP BY AMP
ORDER BY AMP;
```

Using the Teradata Administrator utility we can see the data definition language (DDL) statements that can be used to create the table. We have deleted some of this to make it easier to read.

```
CREATE MULTISSET TABLE MA1.CM_FINANCIAL_ACCOUNT ,FALLBACK ,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM = DEFAULT
(
COMPLAINTS_13_24_MTHS_CNT DECIMAL(16,0) ,
ACCOUNT_RK INTEGER,
PARTY_RK INTEGER,
ACCOUNT_ID CHAR(32) CHARACTER SET LATIN CASESPECIFIC,
ACCOUNT_REGISTRATION_NM CHAR(100) CHARACTER SET LATIN CASESPECIFIC,
.
.
.
INQUIRIES_OTH_13_24_MTHS_CNT DECIMAL(16,0) ,
COMPLAINTS_LAST_3_MTHS_CNT DECIMAL(16,0) ,
COMPLAINTS_LAST_12_MTHS_CNT DECIMAL(16,0))
PRIMARY INDEX ( COMPLAINTS_13_24_MTHS_CNT );
```

Notice that the primary index is COMPLAINTS\_13\_24\_MTHS\_CNT, the first column defined in the table. This does not mean that whoever created the table let Teradata choose the primary index, but it is suspicious. In this case it is causing a significant skewing problem. You may hear this referred to as “lumpy data.”

Changing the primary index to ACCOUNT\_RK allows the data to load and be distributed perfectly. It is important to know your data so that you can choose a good primary index. The primary index determines how the table will be physically arranged. If you do not have access to the Teradata Administrator utility you can use the SHOW TABLE *tablename* command to get the same information from BTEQ or Teradata SQL Assistant.

**SHOW TABLE *tablename*;**

Here is an example using BTEQ.

```

C:\>bteq

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 02R.06.02.02.44
*** Teradata Database Version is 06.02.02.44
*** Transaction Semantics are BTET.
*** Character Set Name is 'ASCII'.

*** Total elapsed time was 2 seconds.

BTEQ -- Enter your DBC/SQL request or BTEQ command:
show table employee;
show table employee;

*** Text of DDL statement returned.
*** Total elapsed time was 1 second.

-----
CREATE MULTISET TABLE TDUSER.employee ,NO FALLBACK ,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM = DEFAULT
(
  EMP_NO INTEGER NOT NULL,
  DEPT_NO INTEGER NOT NULL,
  EMP_NAME VARCHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
  EMP_NICKNAME VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC)
UNIQUE PRIMARY INDEX ( EMP_NO );

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

```

### 3.4 Specify the Primary Index Using SAS

When you are creating Teradata tables using SAS, it is important to choose the best primary index. You can specify the primary index using the DBCREATE\_TABLE\_OPTS= option.

#### **DBCREATE\_TABLE\_OPTS='DBMS-SQL-clauses'**

This option enables you to add DBMS-specific clauses to the end of SAS generated CREATE TABLE statements. Here is an example.

```
DBCREATE_TABLE_OPTS='PRIMARY INDEX (B)'
```

Here is a simple example which uses the DBCREATE\_TABLE\_OPTS= to explicitly set the primary index for a table. In this example a Teradata table will be created based on the SAS data set work.test. If we don't use the DBCREATE\_TABLE\_OPTS= option then "column a" will be chosen as the primary index. That will pose a problem because of the distribution.

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

```
data work.test;
  a='aaaaa'; b=1; output;
  a='aaaaa'; b=2; output;
  a='aaaaa'; b=3; output;
run;
```

```
data mytera.test (DBCREATE_TABLE_OPTS='PRIMARY INDEX (B)');
  set work.test;
run;
```

Here is the DDL from Teradata Administrator. Notice that "column b" is the primary index.

```
CREATE MULTiset TABLE tduser.test, NO FALLBACK,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL,
  CHECKSUM = DEFAULT
(
  a CHAR(5) CHARACTER SET LATIN CASESPECIFIC,
  b FLOAT)
PRIMARY INDEX ( b );
```



## 4 Teradata FastLoad and MultiLoad Utilities

### 4.1 SQL Insert Statements

The most basic way to get data into Teradata is the SQL INSERT statement. However, inserting a large amount of data into the database is inefficient. FastLoad and MultiLoad are much more efficient. Using the Teradata loading utilities enables block writes and fully parallelizes the load process.

You may be wondering, “When does SAS insert data into Teradata?” That is a good question. If you are using SAS to move data into Teradata and you are not using bulk loading, then the data is being inserted into the database.

Here is an example of inserting data into Teradata using SAS SQL (explicit Pass Through).

```
PROC SQL;
    CONNECT TO TERADATA(USER=myuserid PASSWORD=mypassword SERVER=terasrv);
    EXECUTE (INSERT INTO TESTTAB VALUES (1,'Christine','Lewis') BY
TERADATA;
    EXECUTE (INSERT INTO TESTTAB VALUES (2,'Jack','Johnson') BY TERADATA;
    EXECUTE (INSERT INTO TESTTAB VALUES (3,'Joe','Jackson') BY TERADATA;
    EXECUTE (INSERT INTO TESTTAB VALUES (6,'Larry','Nomar') BY TERADATA;
    EXECUTE (INSERT INTO TESTTAB VALUES (5,'Robert','Bob') BY TERADATA;
QUIT;
```

The data on each of the INSERT statements is inserted using one unit of parallelism, and inserted one at a time in a sequential process.

Let's assume we want to move the data from a very large SAS data set named `transaction_file` into a Teradata table of the same name. We can run code similar to this.

```
LIBNAME mytera TERADATA USER=myuserid PASSWORD=mypassword SERVER=terasrv;
LIBNAME sasdata 'C:\sasdata';

PROC SQL;
    CREATE TABLE mytera.transaction_file AS
        SELECT * FROM sasdata.transaction_file;
QUIT;
```

Notice that in the above code that we are taking SAS data and inserting it to Teradata (it will create the Teradata table as part of the process). The SAS data set `transaction_file` might contain tens-of-millions of observations. This could result in a less than stellar performance since the data is being moved over a network and is inserted using one unit of parallelism, and inserted one row at a time in a sequential process, which can result in the same physical block being rewritten multiple times. We will learn how to more efficiently handle this situation in Section 4.2.

### 4.1.1 Batch INSERT Statements to Increase Performance

If you need to insert data into Teradata using SAS there are SAS options that may increase the performance. The MULTISTMT= option was added to the SAS 9.1.3 SAS/ACCESS Interface to Teradata. This option determines whether the INSERT statements should be passed to Teradata one at a time or in batch (multiple INSERT statements at one time).

#### **MULTISTMT=NO|YES**

NO – sends inserts one row at a time. This is the default.

YES – sends as many inserts as will fit into a 64K buffer. If multi-statement inserts are not possible then single statement inserts will be used.

By default, this commits database changes after the last INSERT statement

The MULTISTMT= option is very useful if you are inserting data into Teradata Volatile Tables. Volatile tables are useful in situations where you need a table for very limited amount of time. The volatile table definitions are not stored in the Teradata Data Dictionary. This improves performance. The definition is stored in memory. This means that if you restart the system the definition of your volatile table is lost. MultiLoad or FastLoad cannot be used to load volatile tables. Plus, only the creator of the volatile table can access it.

### 4.1.2 Use Commit Points to Avoid Locking Problems

The DBCOMMIT= option lets you tell the database how often to commit changes to the database data. INSERT, UPDATE and DELETE statements are all subject to commit points. A commit makes changes to database data permanent.

#### **DBCOMMIT=*n***

*n* – specifies an integer greater than or equal to 0

If you set this to 0 it commits at the end of the job. This can cause problems with rowhash locking.

You will need to experiment with this value to determine the best value for your situation. More on this below.

### 4.1.3 INSERT Statement Performance Considerations

Inserting large amounts of data into Teradata is not a good idea. INSERT statements are slow and have other issues.

How slow is it? I ran some tests using my laptop and an R&D Teradata Server. My laptop is really nice, but it is still a laptop. The Teradata Server is a single node environment with 6 AMPs. In short, this environment is not going to set the world on fire. Here is my point, use these performance numbers with care. I am sharing them so that you get an idea of relative performance.

I was able to insert 100,000 rows into a heavily indexed table in 27 minutes 25 seconds. In order to get this to work on some environments you must specify the DBCOMMIT= option. In this case, I set it to 1 (commit each row as it is inserted). This is a bad idea because committing after each insert is very slow.

If you let DBCOMMIT= default then you have another problem. Approximately 75,000 rows of my test table were inserted and then I received an error.

**Transaction exceeded maximum number of rowhash locks allowed.**

Locking is required for each item inserted, updated, or deleted. In Teradata 12 the amount of space allocated to locking cannot be changed. In Teradata 13 the DBA can change parameters in the Teradata Server (DBSControl) and allow more locks. But, they probably won't do that. They will tell you that the problem is with the way the application behaving. You know what? They are right. Use DBCOMMIT= to minimize locking. Or better yet, use FastLoad or MultiLoad.

If you are using INSERT statements you will want to use DBCOMMIT= and MULTISTMT=YES in concert with one another. Dropping indexes and referential integrity (RI) can help out as well but if you can do that you can use FastLoad or MultiLoad.

**Performance Tests: INSERT 100,000 rows**

DBCOMMIT=	MULTISTMT=YES	MULTISTMT=NO
1	27 min 25 sec	29 min 23 sec
50	10 min 1 sec	14 min 12 sec
20000	8 min 45 sec	13 min 17 sec
50000	8 min 56 sec	13 min 36 sec

This simple performance test shows us two things:

- There is a point of diminishing returns in setting DBCOMMIT= (bigger is not necessarily better).
- MULTISTMT=YES increases performance.

#### 4.1.4 Moving Data Within a Teradata Server

The previous example details how you can move data from SAS into Teradata using INSERTs. What happens if we create a new Teradata table based on an existing Teradata table (assume that both tables live in the same Teradata database)? We are talking about copying a table. This can be done very efficiently or very inefficiently.

Here is the TERRIBLY INEFFICIENT way to do this using a SAS DATA step. Assume that the PRODDB.TRANSACTION\_FILE table is huge. Huge in Teradata means billions of rows.

```
LIBNAME mytera TERADATA USER=myuserid PASSWORD=mypassword
      SERVER=terasrv DATABASE=PRODDB;

DATA mytera.new_transaction_file;
  SET mytera.transaction_file;
RUN;
```

Carefully look at the code. Do you see a problem?

When this code runs it selects the entire contents of the PRODDB.transaction\_file table, and moves it into SAS. Once the data has been moved to SAS the new table is defined in Teradata. Finally, the data is inserted into the new table. Typically, the SAS process is running on a machine separate from the Teradata server. That means there will be two network hops and writing the table's contents into SAS: that is a lot of overhead.

This PROC SQL code will do the same processing. Note: this is implicit pass through.

```
LIBNAME mytera TERADATA USER=myuserid PASSWORD=mypassword
      SERVER=terasrv DATABASE=PRODDB;

PROC SQL;
  CREATE TABLE mytera.new_transaction_file AS
    SELECT * from mytera.transaction_file;
RUN;
```

Network traffic is not the only problem with this method of moving data within a Teradata server. The rows are inserted using one unit of parallelism, and inserted one row at a time in a sequential process, which can result in the same physical block being rewritten multiple times. It is best to avoid this situation when adding large amounts of data because it can contribute to very inefficient performance.

This is not a performance benchmarking paper but for example, we ran a small job that moved 50,000 rows using this technique and it took 5 minutes and 30 seconds.

Teradata has an optimized form of INSERT called Fast Path INSERT ... SELECT. This is very useful if you want to move data from one Teradata table to another Teradata table. Using this technique you can use a simple INSERT ... SELECT and have it perform as if it is using a load utility. This option is fully parallelized and uses block writes. In addition, INSERTs into an empty target table will bypass transient journaling overhead. The target table is the table into which the data will be inserted.

Load utilities have the ability to put very large amounts of data into a database very quickly. Typically, they do this using all available units of parallelism, by not logging (FastLoad only), and by writing data blocks directly (SQL processing is not used).

Teradata will use Fast Path in a CREATE TABLE ... SELECT statement. Let's revisit our PROC SQL example, but this time let's use the DBIDIRECTEXEC= system option. The DBIDIRECTEXEC= option tells SAS to pass a CREATE TABLE ... SELECT statement to Teradata.

This statement will allow Teradata to use Fast Path INSERT ... SELECT to move the data from the old table into the new table

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
LIBNAME mytera TERADATA USER=myuserid PASSWORD=mypassword
        SERVER=terasrv DATABASE=PRODDB;
OPTIONS DBIDIRECTEXEC;

PROC SQL;
    CREATE TABLE mytera.new_transaction_file AS
        SELECT * from mytera.transaction_file;
QUIT;
```

This time the CREATE TABLE ... SELECT was passed to Teradata. We know that because the SASTRACE= option shows us the SQL statement that created the table and inserted data into the new table was executed by Teradata. Teradata was able to use Fast Path INSERT ... SELECT when writing the data to the table.

```
TERADATA_4: Executed: on connection 1
CREATE MULTISET TABLE "new_transaction_file" as ( select
"transaction_file"."HOUSEHOLD_RK",
"transaction_file"."HHOLD_ID", "transaction_file"."HHOLD_TYPE_CD",
"transaction_file"."HHOLD_CLIENTS_CNT",
...
Some column definitions deleted
...
"transaction_file"."HHOLD_PROFITABILITY_TODATE_AMT",
"transaction_file"."HHOLD_PROFIT_PROJECTED_AMT",
"transaction_file"."HHOLD_PROFIT_POTENTIAL_AMT"
from "transaction_file" ) WITH DATA
```

This operation was fully parallelized and used block writes, bypassed writing to the Transient Log thus avoiding a lot of I/O due to the target table being empty, avoided two network traversals, and avoided writing the contents of the table to SAS temporary data sets. As a result: the data was moved in approximately 9.5 seconds as opposed to 5 minutes and 30 seconds in the previous example.

### 4.1.5 Inserting Rows into an Existing Teradata Table

One of the great things about Teradata is that it will use the loading technology when moving data within a Teradata Server. Previously, we saw that creating a table and loading it in one step is very efficient. That same performance is available to you if you want to insert data from a Teradata table into an existing Teradata table. This is true for both empty and populated target tables. To do this you will use SQL similar to this.

```
INSERT INTO TESTDB.NEW_TABLE SELECT * FROM TESTDB.OLD_TABLE;
```

In order for this to work, efficiently, this SQL statement must be executed by Teradata. Fortunately, that is easy to do. Here is an example.

```
OPTIONS SASTRACE=',,,d' SASTRACELOC=SASLOG NOSTSUFFIX;
PROC SQL;
  CONNECT TO TERADATA (USER=username PASSWORD=password SERVER=myserver);
  EXECUTE (INSERT INTO TESTDB.NEW_TABLE
          SELECT * FROM TESTDB.OLD_TABLE) BY TERADATA;
QUIT;
```

You should see a message in your SAS Log stating that the query was executed by Teradata. Here is an example.

```
TERADATA_28: Executed: on connection 2
INSERT INTO TESTDB.NEW_TABLE SELECT * FROM TESTDB.OLD_TABLE
```

It is a good idea to always write the database query information into the SAS log when you are developing your extraction, transformation, and load (ETL) jobs. As a developer your goal should be to have the Teradata Server do as much of the work as possible.

This can be written so that the SQL is passed to the Teradata Server implicitly. It is always a good idea to check the SAS Log to ensure that the database is handling the query.

## 4.2 The Teradata FastLoad Utility

FastLoad lives to put large amounts of data into an **empty Teradata table** as quickly as possible. FastLoad is both a Teradata utility and a protocol. The SAS/ACCESS Interface to Teradata can use the FastLoad protocol to quickly load data into empty Teradata tables.

In order to load data using FastLoad:

- The target table must be empty.
- The target table must have no secondary indexes, join indexes, or hash indexes defined on it.
- The target table must have no triggers defined on it.
- The target table must have no standard referential integrity or batch referential integrity defined on it (Soft Referential Integrity is allowed).
- Duplicate rows cannot be loaded.

There are many restrictions on using FastLoad for loading tables. Many of these restrictions revolve around indexing. In order to use FastLoad you may need to drop indexes, triggers, and referential integrity (RI) prior to loading. The indexes can then be recreated after the load is complete.

RI is a method of enforcing parent – child relationships within the database. There are three types of referential integrity available in Teradata:

- Standard referential integrity
- Batch referential integrity
- Soft referential integrity.

<b>Standard Referential Integrity</b>	<p>Guarantees that the parent – child relationships are maintained as each row is altered (insert, update, or delete) in the tables which participate in the relationship. This form of RI check incurs a performance hit.</p> <p>You <b>CANNOT</b> use utilities (or protocols) like FastLoad, MultiLoad, Teradata Parallel Transporter LOAD, and UPDATE options on tables defined with this type of RI.</p>
<b>Batch Referential Integrity</b>	<p>This form of RI is less expensive performance-wise because it is enforced as an all-or-nothing operation. If RI is violated the entire transaction fails.</p> <p>You <b>CANNOT</b> use utilities (or protocols) like FastLoad, MultiLoad, Teradata Parallel Transporter LOAD, and UPDATE options on tables defined with this type of RI.</p>
<b>Soft Referential Integrity (Referential Constraints)</b>	<p>The Teradata optimizer is sometimes able to design more efficient execution plans if there is RI defined on tables. This is a one of the potential benefits of having RI defined. Not being able to use utilities is one of the great disadvantages of having RI defined.</p> <p>Soft RI is the best of both worlds. The optimizer can use the relationships when creating execution plans while the utilities can be used to load the tables. It is the responsibility of the developer to manually ensure that RI integrity is maintained as defined.</p> <p>You <b>CAN</b> use utilities (or protocols) like FastLoad, MultiLoad, and Teradata Parallel Transporter on tables defined with this type of RI.</p>

The restriction against loading duplicate rows into tables using FastLoad is interesting. Teradata was designed to adhere to strict relational rules, one of which is that you cannot have duplicate rows in a table. Over the course of time this restriction was loosened. Now you can create a table that will allow duplicate rows. This is called a multiset table. Unfortunately you cannot use FastLoad to load these tables if the load data contains and requires duplicate rows

### 4.3 Loading Data Using the FastLoad Utility

Although it is not the purpose of this paper to teach you how to use the Teradata FastLoad utility, it is a good idea to look at a simple script. This script does nothing fancy, it simply loads data from a text file into a Teradata table. The one interesting thing about the script is that it loads varying length records into the table.

The script:

1. Connects to the Teradata server.
2. Drops the target and error tables if they exist.
3. Creates the target table and specifies a primary index.
4. Defines the layout of the data file.
5. Inserts the data into the new table.

teraserver	The name of your Teradata server. This can be found in your hosts file. The name will end in copn (where n is a number). For example. If you find teraprodcop1 in your hosts file, you can use teraprod in your LOGON statement.
userid	This is your Teradata userid. You can get this from your database administrator.
password	This is your Teradata password. You can get this from your database administrator.

Sample Script:

```
SESSIONS 1 ;
ERRLIMIT 25 ;

LOGON teraserver/userid,password;

DROP TABLE load_test ;
DROP TABLE Error1 ;
DROP TABLE Error2 ;

CREATE MULTiset TABLE load_test ,
    FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
```



```

(
    ACCOUNT_ID          INTEGER,
    ,OPEN_DT            TIMESTAMP(0) FORMAT 'DDMMYY4:HH:MI:SS'
    ,FIRST_NM           CHAR(40) CHARACTER SET LATIN CASESPECIFIC
    ,LAST_NM            CHAR(40) CHARACTER SET LATIN CASESPECIFIC)
PRIMARY INDEX ( ACCOUNT_ID );

SET RECORD VARTEXT "|" DISPLAY_ERRORS;

BEGIN LOADING load_test
    ERRORFILES Error1, Error2
    CHECKPOINT 100000;

DEFINE
    in_ACCOUNT_ID      (VARCHAR(11))
    , in_OPEN_DT        (VARCHAR(19))
    , in_FIRST_NM       (VARCHAR(40))
    , in_LAST_NM        (VARCHAR(40))
    FILE=load_test.dat;

INSERT INTO load_test
( ACCOUNT_ID          = :in_ACCOUNT_ID
  , OPEN_DT           = :in_OPEN_DT   (FORMAT 'DDMMYY4:HH:MI:SS')
  , FIRST_NM          = :in_FIRST_NM
  , LAST_NM           = :in_LAST_NM
);
END LOADING;
QUIT;

```

The script loads this data into the load\_test table.

```

731594|21SEP1994:00:00:00|Stella|Wright
876328|15MAR1988:00:00:00|Tom|Olson
315280|24FEB1988:00:00:00|Bob|Door
315281|24FEB1988:00:00:00|Don|Barely
315282|24FEB1988:00:00:00|Christine|Nortiv
315283|24FEB1988:00:00:00|Joe|Nomar
315284|24FEB1988:00:00:00|Carter|Larry

```

The data file must end with a blank line. If you do not include a blank line you will get an error such as this:

```

**** 09:30:45 I/O Error on File Read: 35, Text: EOF encountered before
        end of record

```

## 4.4 The Teradata MultiLoad Utility

MultiLoad is designed to put large amounts of data into a **non-empty Teradata table** as quickly as possible. Like FastLoad, MultiLoad is both a Teradata batch utility and a protocol. The SAS/ACCESS Interface to Teradata can use the MultiLoad protocol to quickly load data into Teradata tables that already contain data. Unlike FastLoad, the MultiLoad protocol allows you to load

duplicate rows into multiset tables. We will refer to the table into which data is to be loaded as the target table.

In order to load data using MultiLoad:

- The target table must have no unique secondary, join, or hash indexes defined on it.
- The target table must have no triggers defined on it.
- The target table must have no standard referential integrity or batch referential integrity defined on it (Soft Referential Integrity is allowed).
- The MultiLoad input file must have data to qualify all columns defined in the Primary Index of the target table.

There are many restrictions on using MultiLoad for loading tables. Many of these restrictions revolve around indexing. In order to use MultiLoad you may need to drop unique secondary indexes, join indexes or hash indexes, triggers, and RI prior to loading. There are no hard and fast rules regarding the dropping of indexes and performance. For example, you may find that for low volumes of data that you load it is faster to use simple inserts. For large volumes of data it may be faster to FastLoad the load data into an empty staging table, and then use a Teradata INSERT ... SELECT process (see Section 3.3) to populate to target table. Keep in mind that dropping and recreating indexes can take lots of time if your tables are large. Choosing the appropriate method requires testing in your specific environment.

FastLoad will only load data into one table at a time. With MultiLoad you can place data in up to five tables at one time. MultiLoad can be used when you have an input file with more than one type of record in it. You can code logic in the MultiLoad script that will process each record type differently.

## 4.5 Loading Data Using the MultiLoad Utility

Although it is not the purpose of this paper to teach you how to use the Teradata MultiLoad utility, it is a good idea to look at a simple script. You can find more information on running MultiLoad scripts in the Teradata MultiLoad Reference.

MultiLoad will load data into existing tables, so we need an existing table. Here is the DDL that creates the table for this example. The script also places a few rows in it to show that it will load a table containing rows. You will need to make the following changes to the script in order to run it successfully.

teraserver	The name of your Teradata server. This can be found in your hosts file. The name will end in copn (where n is a number). For example. If you find teraprodcop1 in your hosts file, you can use teraprod in your LOGON statement.
userid	This is your Teradata userid. You can get this

	from your database administrator.
password	This is your Teradata userid. You can get this from your database administrator.
MYDB	This is the database in which you created your table. This will most likely be your userid but it doesn't have to be. If you are creating this table in your default database you do not need to specify this value.

```

LOGON teraserver/userid,password;
DROP TABLE MYDB.Scores;
DROP TABLE MYDB.UV_Scores;
CREATE MULTiset TABLE MYDB.scores ,NO FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT
    (
        IDNUM    INTEGER
        ,Name     CHAR(30)
        ,Points   INTEGER
    )
PRIMARY INDEX ( IDNUM );

INSERT INTO MYDB.Scores VALUES (1,'Christine',100);
INSERT INTO MYDB.Scores VALUES (2,'Jack',0);
INSERT INTO MYDB.Scores VALUES (3,'Bob',50);
LOGOFF;

```

The following script takes input data from an ASCII text file and for record type 'A' adds a value to the POINTS column; for record type 'S' it subtracts a value from the POINTS column; and for record type 'I' it inserts a new row.

The script:

1. Specifies a LOGTABLE for the MultiLoad job
2. Connects to the Teradata server
3. Begins the MLOAD process and specifies the output tables
4. Defines the layout of the data file
5. Defines the SQL to use for adding points (Add\_Points Label)
6. Defines the SQL to use for subtracting points (Sub\_Points Label)
7. Defines the SQL to use for inserting a row (Add\_Row Label)
8. Reads the input file (INFILE) and determines which action to take

9. Ends the MLOAD job

10. Logs off the database.

#### Sample Script:

```
.LOGTABLE scores_mload;
.LOGON teraserver/userid,password;
.BEGIN IMPORT MLOAD TABLES scores;
.Layout Record_Layout;
  .FIELD in_IDNUM      1   CHAR(11);
  .FIELD in_Rec_Type  13   CHAR(1);
  .FIELD in_Name       15   CHAR(30);
  .FIELD in_Points    46   CHAR(11);
.DML LABEL Add_Points;
  UPDATE scores set Points = (Points + :in_Points)
    WHERE IDNUM = :in_IDNUM;
.DML LABEL Sub_Points;
  UPDATE scores set Points = (Points - :in_Points)
    WHERE IDNUM = :in_IDNUM;
.DML LABEL Add_Row;
  INSERT INTO scores VALUES (:in_IDNUM, :in_Name, :in_Points);

.IMPORT INFILE MultiLoadsample.dat
  LAYOUT Record_Layout
  FORMAT TEXT
  APPLY Add_Points WHERE in_Rec_Type = 'A'
  APPLY Sub_Points WHERE in_Rec_Type = 'S'
  APPLY Add_Row    WHERE in_Rec_Type = 'I';
.END MLOAD;
.LOGOFF;
```

The script loads this data into the load\_test table. We are inserting a row for Jeff and subtracting 10 from Christine's points.

4	I	Jeff		100
1	S	Christine		10

Here are the contents of the SCORES table prior to running the MultiLoad script.

```

C:\> Command Prompt - bteq
BTEQ -- Enter your DBC/SQL request or BTEQ command:
select * from scores;

select * from scores;

*** Query completed. 3 rows found. 3 columns returned.
*** Total elapsed time was 1 second.

      IDNUM  Name                Points
-----
          2  Jack                  0
          3  Bob                   50
          1  Christine             100

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

```

Here are the contents of the table after running the MultiLoad script. Notice that the update and insert have been applied to our table.

```

C:\> Command Prompt - bteq
BTEQ -- Enter your DBC/SQL request or BTEQ command:
select * from scores;

select * from scores;

*** Query completed. 4 rows found. 3 columns returned.
*** Total elapsed time was 1 second.

      IDNUM  Name                Points
-----
          2  Jack                  0
          3  Bob                   50
          1  Christine             90
          4  Jeff                  100

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

```

The data file must end with a blank line. If you do not include a blank line you will get this error:

```

**** 11:00:41 UTY4015 Access module error '35' received during 'read'
operation on record number '1': 'EOF encountered before end of record'

```

## 5 Loading SAS Data Into Teradata (FastLoad)

### 5.1 FastLoad Protocol Restrictions

We previously saw how to invoke FastLoad to quickly load data into the Teradata database. Now let's take a look at how we can FastLoad from within SAS. Remember, this functionality is available using a utility program or an API. The SAS/ACCESS interface uses the FastLoad API. Let's review the restrictions when using FastLoad.

In order to load data using FastLoad:

- The target table must be empty.
- The target table must have no secondary indexes, join indexes, or hash indexes defined on it.
- The target table must have no triggers defined on it.
- The target table must have no standard referential integrity or batch referential integrity defined on it.. There is a detailed discussion of RI in Section 4.4.
- Duplicate rows cannot be loaded.

Restriction	How to verify
Target table must be empty	<p>You can issue a SELECT statement to determine if the table is empty. Using the LOAD_TEST table that we created and loaded above as an example we could use this query.</p> <pre>SELECT COUNT(*)   FROM TESTDB.LOAD_TEST;</pre>
Target table must have no secondary index, join index, or hash index defined on it.	<p>Queries similar to this can be used to find indexes defined on the tables you are planning to load.</p> <pre>SELECT *   FROM DBC.INDICES  WHERE TableName IN ('EMPLOYEE','DEPARTMENT')  ORDER BY DatabaseName, TableName, IndexName; IndexType  J = Join Index N = Hash Index S = Secondary Index V = Value Ordered Secondary Index</pre>
Target table must have no	<p>Queries similar to this can be used to find triggers defined on the tables you are planning to load.</p>

triggers defined on it.	<pre>SELECT *   FROM DBC.TriggersX  WHERE TableName='EMPLOYEE' ;</pre>
Target table must have no standard or batch RI defined on it.	<p>Queries similar to these can be used to find parent child relationships.</p> <pre>SELECT *   FROM DBC.ALL_RI_CHILDRENX  WHERE ParentDB='TESTDB' ;</pre> <p>Or</p> <pre>SELECT *   FROM DBC.ALL_RI_PARENTSX  WHERE ParentDB='TESTDB' ;</pre>
Duplicate rows cannot be loaded.	Check this using SAS (prior to loading the data).

## 5.2 SAS Bulkloading Using Teradata FastLoad

The SAS/ACCESS Interface to Teradata makes using the FastLoad protocol extremely easy. You need to be aware of four options:

### 5.2.1 BULKLOAD=

The BULKLOAD= data set and LIBNAME option tells SAS whether or not to use a DBMS facility to insert data into a database table. When using SAS/ACCESS Interface to Teradata the Teradata FastLoad protocol will be used to load data into Teradata tables.

#### **BULKLOAD=YES|NO**

**YES** – Teradata FastLoad protocol is used when appending or inserting data into the Teradata table.

**NO** – The FastLoad protocol is not used when appending or inserting data into the Teradata table.

**FASTLOAD=** is an alias for BULKLOAD=

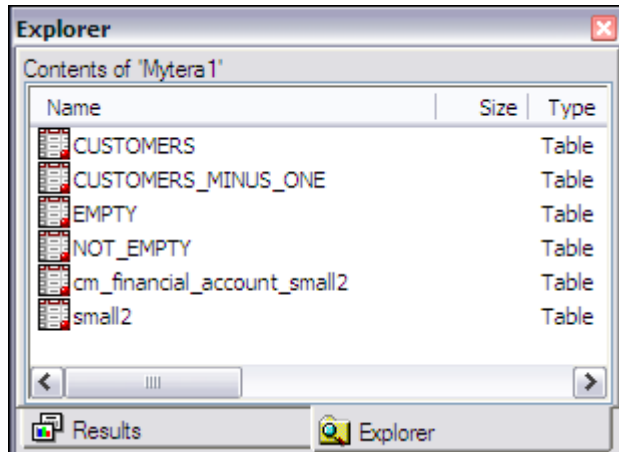
All of the FastLoad restrictions that were discussed earlier apply. In fact, later in this paper we will see how SAS reacts when we violate these rules.

Although the BULKLOAD= option can be used as both a data set and a LIBNAME option, it is more common to use it as a data set option. It makes sense to place the option where you intend to use it. Setting it at the library level tends to hide the fact that it is being used.

Here is an example of a SAS library that is defined with the BULKLOAD= option.

```
LIBNAME mytera1 TERADATA USER=myuserid PASSWORD=mypassword  
      SERVER=terasrv DATABASE=TESTDB BULKLOAD=YES;
```

When we issue the command on our system we see these tables.



Since we have defined that BULKLOAD=YES on the library any SAS code which adds rows to one of its tables will use the FastLoad protocol. That means that if the CUSTOMERS table has a join index defined on it the SAS code will fail. Since the BULKLOAD= option is “hidden” on the library, it is hard to see.

```
PROC APPEND BASE=MYTERA1.CUSTOMERS  
      DATA=WORK.NEW_CUSTOMERS;  
RUN;
```

That PROC Append code will use the FastLoad protocol. At least it will if we use the LIBNAME statement that was shown above.

What about this Data step code?

```
DATA TERAPROD.CUSTOMERS;  
      SET WORK.NEW_CUSTOMERS;  
RUN;
```

Without looking at the LIBNAME statement there is no way to know. This situation can be further complicated if you have your LIBNAME information defined in SAS metadata. There is no doubt that following SAS DATA step is using the BULKLOAD= option.



```
DATA TERAPROD.CUSTOMERS (BULKLOAD=YES) ;
  SET WORK.NEW_CUSTOMERS ;
RUN ;
```

You can use the FASTLOAD= alias for BULKLOAD=.

```
DATA TERAPROD.CUSTOMERS (FASTLOAD=YES) ;
  SET WORK.NEW_CUSTOMERS ;
RUN ;
```

### 5.2.2 BL\_LOG=

BL\_LOG= is a data set option that tells SAS where to store the output from the loading process. It is optional but highly recommended. When using SAS/ACCESS with other database management systems this option points SAS to a directory where the load logs can be written.

Teradata works differently; it writes this information to database tables. The BL\_LOG= option, when used with Teradata, allows you to choose a custom name for these tables. By default SAS will name them SAS\_FASTLOAD\_ERRS1\_randnum and SAS\_FASTLOAD\_ERRS2\_randomnum.

If you are loading multiple Teradata tables in a job then you may want to use BL\_LOG= to specify meaningful names. It will make debugging problems much easier.

There is another reason you may need to specify this option. Using the default value can be a problem in production Teradata environments. By default, the SAS/ACCESS Interface to Teradata will create these temporary tables in the database that the SAS Library points to. Electronic Data Processing (EDP) auditors are quick to point out that applications are not allowed, under any circumstances, to create tables in production databases. Pointing BL\_LOG= to another database will allow you to avoid this issue.

#### **BL\_LOG= a meaningful table name**

To specify this option you must specify BULKLOAD=YES.

The Teradata bulk-load facility will create error tables based on this name. Use meaningful names.

For example, if you specify BL\_LOG=CUSTOMER\_ERR then errors are logged in CUSTOMER\_ERR1 and CUSTOMER\_ERR2.

You can place the utility tables in a separate database by prepending the database name and a period to the value specified in the BL\_LOG= option. Let's assume that we need to put the utility tables in the TEMPUTIL Teradata database and we want the table names to all start with INITIAL\_LD\_. Here is what our BL\_LOG= statement would look like.

```
BL_LOG=TEMPUTIL.INITIAL_LD
```

### 5.2.3 FBUFSIZE=

During the transition from SAS 8 to SAS 9 something happened to cause SAS/ACCESS Interface to Teradata's FASTLOAD performance to degrade. This is documented in [Problem Note 18170](#). There is an associated hot fix that can be installed to help address this issue. Currently, the hot fix is E9TE04. The hot fix will most certainly change. You can find a link to it in the Problem Note.

One of the more interesting tidbits of information in the note concerns the FBUFSIZE= data set option. This option is not easy to find in the product documentation. FBUFSIZE= allows you to change the amount of memory assigned to the buffers that are used to pass data to Teradata. During our testing we discovered that changing this value can affect performance. Setting FBUFSIZE= to larger values appears to be better. Keep in mind that you will need to experiment in your environment in order to determine the best value for FBUFSIZE=.

#### **FBUFSIZE= Integer from 1 to 66750**

The default value appears to be 64000. I discovered the maximum value through experimentation. We can find no documentation which clearly states the default value.

Setting FBUFSIZE= to larger values appears to be better. Finding an acceptable value for FBUFSIZE= may take experimentation. I was able to get the code to run when this option is set to 1, but it ran very slowly. The best results appear when you have this option set towards the higher end of the range.

### 5.2.4 DBCOMMIT=

The DBCOMMIT= option (CHECKPOINT= is an alias) causes a Teradata checkpoint after *n* rows are loaded or inserted.

#### **DBCOMMIT=*n***

DBCOMMIT= will cause a Teradata checkpoint every *n* rows. If you use BULKLOAD=YES but do not specify a value for DBCOMMIT= then no checkpoints are used. This is the same as setting DBCOMMIT=0. The commit occurs at the end of the job.

Using a value of 0 means COMMIT after the table is loaded. This works well on smaller loads.

If you are loading small tables there is not an issue with omitting the DBCOMMIT= option; you can simply delete the contents of the table and start over.

If the table that you are loading is large, you may want to issue a checkpoint. This will save you from having to restart the load. Internal training by Teradata suggests that you checkpoint every 15 minutes on large load jobs. You will need to experiment with DBCOMMIT= (CHECKPOINT=) in order to determine the proper value.

If the load job fails you will need to determine the most recent row added to the table and then code a SAS job to complete the load. Keep in mind, the FastLoad protocol cannot be used to load data into non-empty tables. You can use the MultiLoad protocol, which we will discuss later, to complete the load process. This may not be easy, but it may be quicker than starting over.

## 5.2.5 Loading Teradata Tables Using SAS

The mechanics of loading your empty Teradata tables using SAS is simple. Here is an example using PROC APPEND (notice that the FASTLOAD= alias is used instead of BULKLOAD=). Notice that we have not specified the DBCOMMIT= or CHECKPOINT= options. This means that Teradata will commit the changes after the entire table is loaded.

```
PROC APPEND
    BASE=mytera2.cm_financial_account_new
        (FASTLOAD=YES FBUFSIZE=66750 BL_LOG=TEST_LOAD)
    DATA=work.cm_financial_account;
RUN;
```

PROC APPEND is a great choice for loading data into existing tables. It is very common to load data into existing tables during the ETL process. As a general rule, you want to be careful when creating a Teradata table and loading it on the fly.

Here is an example of using PROC SQL to load SAS data into an existing, empty, Teradata table.

```
proc sql;
    insert into mytera2.cm_financial_account_new
        (FastLoad=YES FBUFSIZE=66570 BL_LOG=TEST_LOAD)
    select * from work.cm_financial_account;
quit;
```

Using PROC SQL in the above manner will work great. The FastLoad protocol enables the job to run more quickly.

Here is an example that you will want to think about before copying. Notice that there is no BULKLOAD= or FASTLOAD= options on the mytera2.cm\_financial\_account\_new dataset. This means that it will issue INSERT statements.

```
proc sql;
    insert into mytera2.cm_financial_account_new (CHECKPOINT=50000)
    select * from work.cm_financial_account;
quit;
```

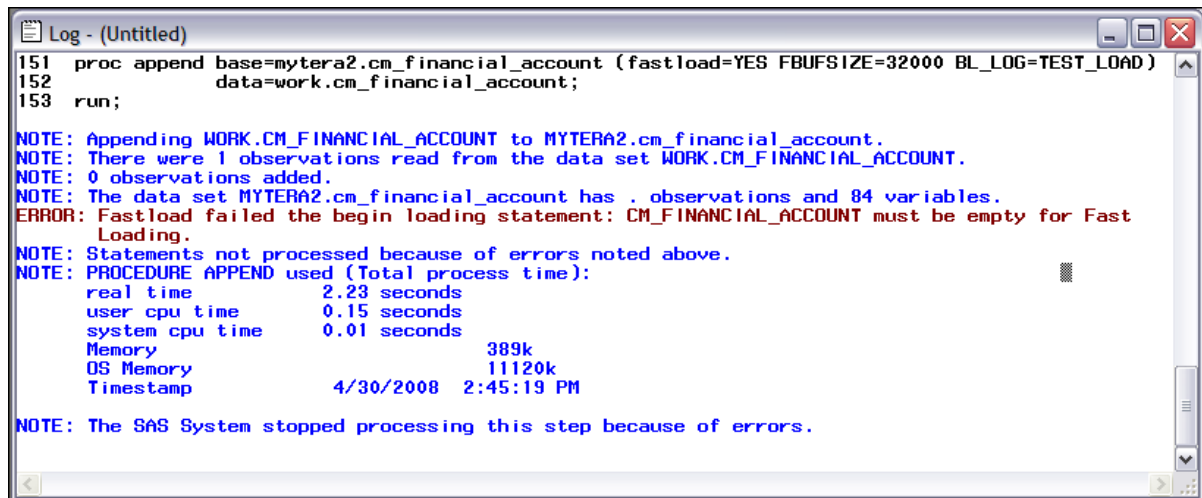
On our test system, this code will not run. It goes for a while and then fails. A rollback is issued and the added rows are removed from the table. This takes a lot of time (1 hour and 7 minutes). Here is the error that is encountered.

**ERROR: Teradata insert: The AMP Lock table has overflowed.**

The slowness is due to the fact that the rows are inserted using one unit of parallelism, and inserted one at a time in a sequential process. The error was caused by the requirement for each inserted row to have a separate lock in the AMP (unit of parallelism) lock table. If you encounter an error such as this, a different load approach will be required, such as FastLoad to load the data into an empty staging table and then use a Teradata INSERT/SELECT process (see Section 3.3) to populate to target table.

## Sample Bulk Loading Problems

### 5.2.5.1 FastLoad Into a Non-Empty Table



```

Log - (Untitled)
151 proc append base=mytera2.cm_financial_account (fastload=YES FBUFSIZE=32000 BL_LOG=TEST_LOAD)
152     data=work.cm_financial_account;
153 run;

NOTE: Appending WORK.CM_FINANCIAL_ACCOUNT to MYTERA2.cm_financial_account.
NOTE: There were 1 observations read from the data set WORK.CM_FINANCIAL_ACCOUNT.
NOTE: 0 observations added.
NOTE: The data set MYTERA2.cm_financial_account has . observations and 84 variables.
ERROR: Fastload failed the begin loading statement: CM_FINANCIAL_ACCOUNT must be empty for Fast
Loading.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           2.23 seconds
      user cpu time       0.15 seconds
      system cpu time     0.01 seconds
      Memory              389k
      OS Memory           11120k
      Timestamp           4/30/2008 2:45:19 PM

NOTE: The SAS System stopped processing this step because of errors.
  
```

If you encounter this error you must ask yourself, “Should this table be empty?” If the answer to that question is “Yes” then emptying the table may be the correct action to take. There could be many reasons for the table to contain rows when it should not. A failed load could leave extraneous rows in the table. If you do decide to remove the rows, use this command.

**DELETE FROM mydb.mytable ALL;**

This form of the command quickly removes the rows from the table without logging the changes. It behaves like Oracle’s TRUNCATE command.

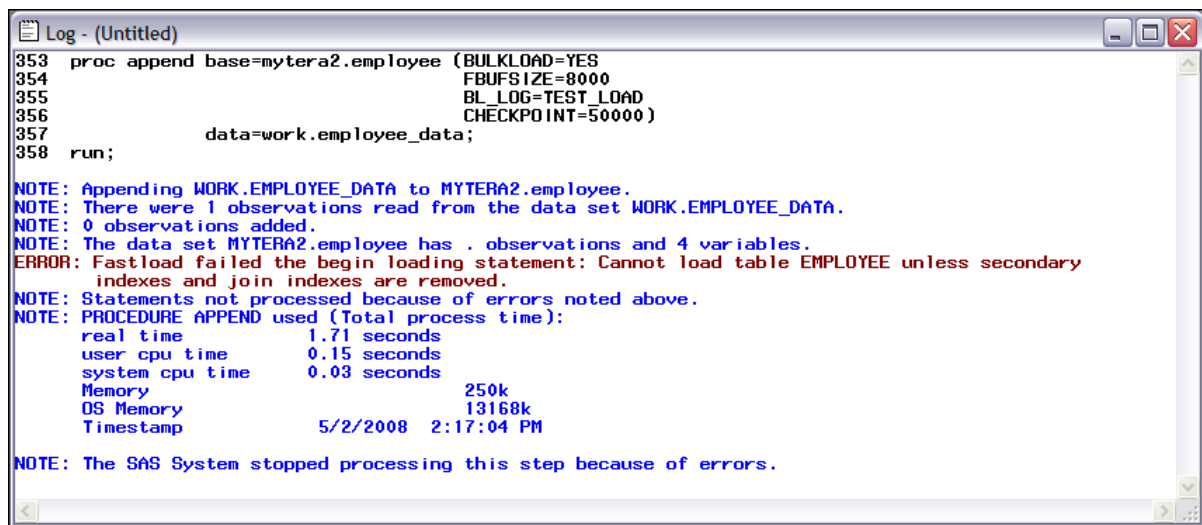
If the table is not supposed to be empty then you must use a different load technique, probably MultiLoad. We will discuss the alternative techniques in a subsequent section of this paper.

### 5.2.5.2 FastLoad a Table With a Join Index Defined on it

If we define any of the following indexes on our EMPLOYEE table then our load will fail and display the following error:

```
CREATE INDEX DEPT_NUSI_INDX (DEPT_NO) ON EMPLOYEE;

CREATE JOIN INDEX EMPDEPT_INDX,
  NO FALLBACK AS
  SELECT E.EMP_NO, E.DEPT_NO, E.EMP_NAME, E.EMP_NICKNAME, D.DEPT_NAME
  FROM EMPLOYEE E INNER JOIN DEPARTMENT D
    ON E.DEPT_NO = D.DEPT_NO
  PRIMARY INDEX (DEPT_NO);
```



```
Log - (Untitled)
353 proc append base=mytera2.employee (BULKLOAD=YES
354                                     FBUFFSIZE=8000
355                                     BL_LOG=TEST_LOAD
356                                     CHECKPOINT=50000)
357       data=work.employee_data;
358 run;

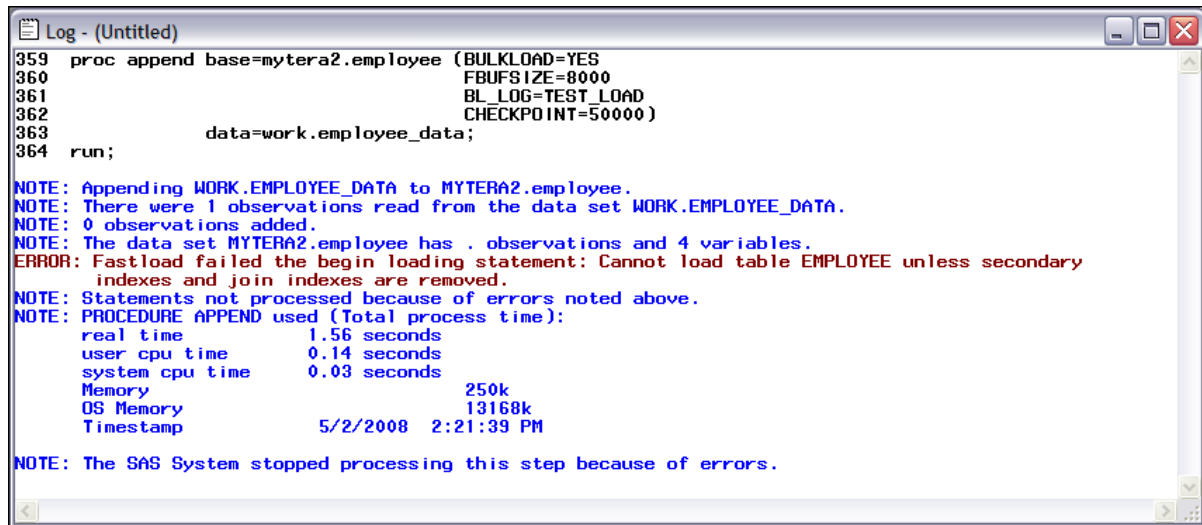
NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 1 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 0 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
ERROR: Fastload failed the begin loading statement: Cannot load table EMPLOYEE unless secondary
       indexes and join indexes are removed.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
       real time          1.71 seconds
       user cpu time      0.15 seconds
       system cpu time    0.03 seconds
       Memory              250k
       OS Memory          13168k
       Timestamp          5/2/2008  2:17:04 PM

NOTE: The SAS System stopped processing this step because of errors.
```

### 5.2.5.3 FastLoad a Table With a Hash Index Defined on it

Although the hash index is not specifically mentioned in the error message it is not allowed. Notice that the error messages are the same when you have a prohibited index defined on the table.

```
CREATE HASH INDEX EMP_NAME (EMP_NAME, EMP_NICKNAME)
  ON EMPLOYEE BY (EMP_NAME)
  ORDER BY HASH      (EMP_NAME);
```



```
Log - (Untitled)
359 proc append base=mytera2.employee (BULKLOAD=YES
360                                     FBUFFSIZE=8000
361                                     BL_LOG=TEST_LOAD
362                                     CHECKPOINT=50000)
363       data=work.employee_data;
364 run;

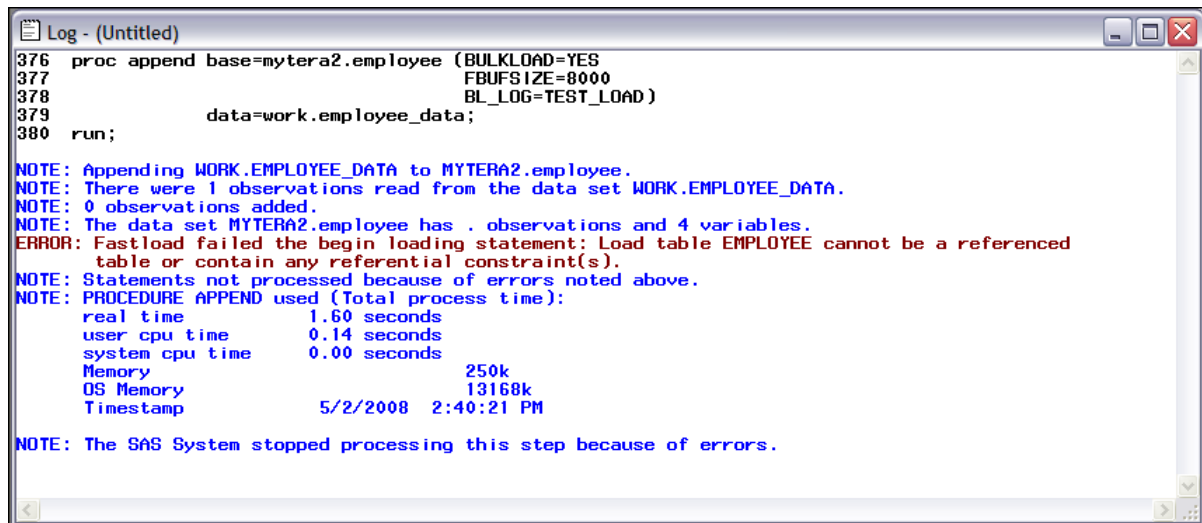
NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 1 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 0 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
ERROR: Fastload failed the begin loading statement: Cannot load table EMPLOYEE unless secondary
       indexes and join indexes are removed.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
       real time          1.56 seconds
       user cpu time      0.14 seconds
       system cpu time    0.03 seconds
       Memory              250k
       OS Memory          13168k
       Timestamp          5/2/2008  2:21:39 PM

NOTE: The SAS System stopped processing this step because of errors.
```

### 5.2.5.4 FastLoad a Table Having RI Defined on it

If you have RI or Batch RI defined on the table then you will see “referenced table” mentioned in the error message.

```
ALTER TABLE EMPLOYEE ADD CONSTRAINT DEPT_NO_FK
  FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEPT_NO);
```



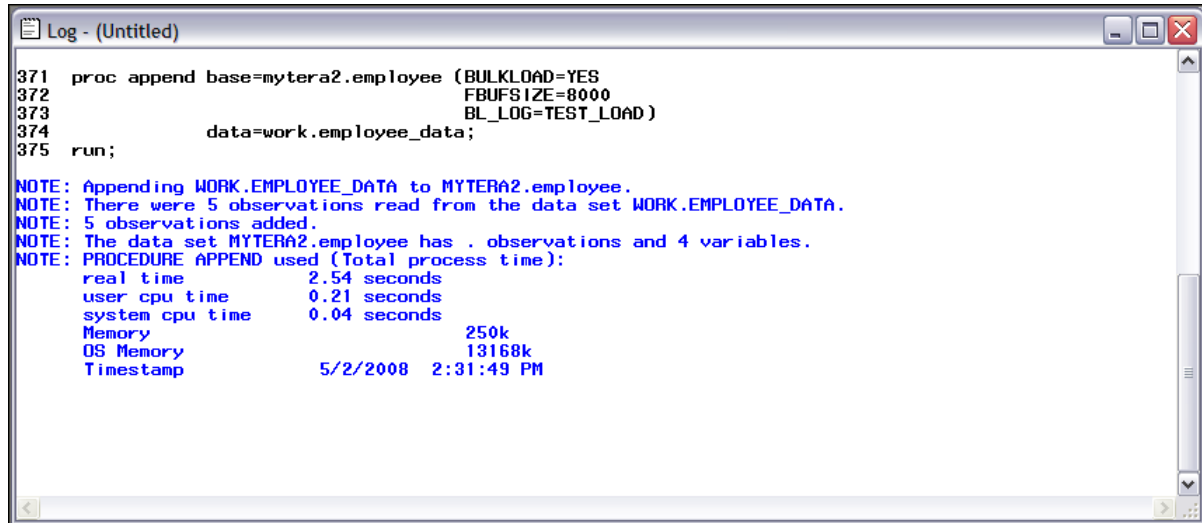
```
Log - (Untitled)
376 proc append base=mytera2.employee (BULKLOAD=YES
377                                     FBUFFSIZE=8000
378                                     BL_LOG=TEST_LOAD)
379       data=work.employee_data;
380 run;

NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 1 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 0 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
ERROR: Fastload failed the begin loading statement: Load table EMPLOYEE cannot be a referenced
       table or contain any referential constraint(s).
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
       real time          1.60 seconds
       user cpu time      0.14 seconds
       system cpu time    0.00 seconds
       Memory              250k
       OS Memory          13168k
       Timestamp          5/2/2008  2:40:21 PM

NOTE: The SAS System stopped processing this step because of errors.
```

If Soft RI is defined on a table then you can load it. Remember, Soft RI tells the optimizer about the relationship but the relationship is not enforced. The **WITH NO CHECK OPTION** tells Teradata that this is Soft RI.

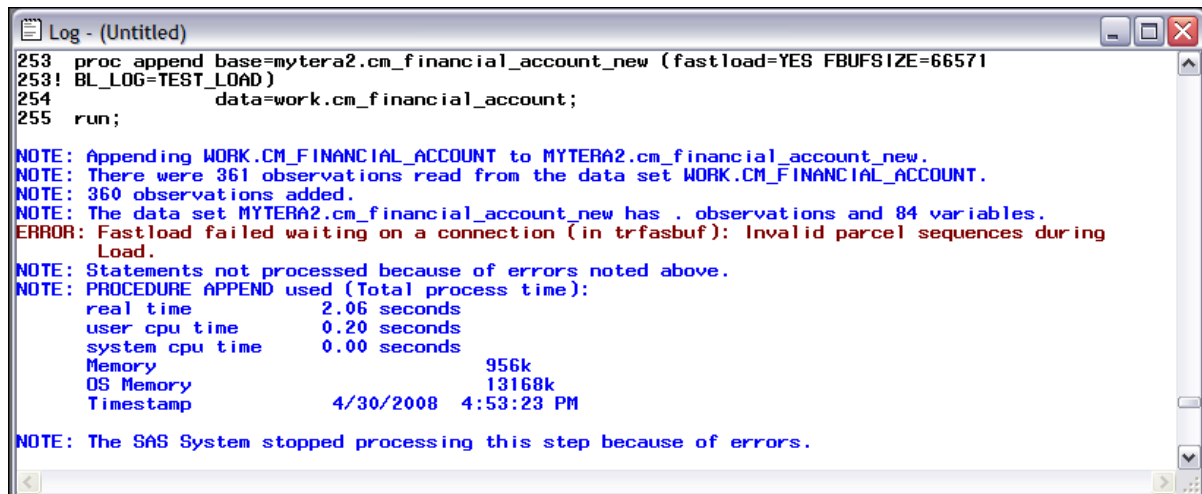
```
ALTER TABLE EMPLOYEE ADD CONSTRAINT DEPT_NO_SOFT_RI_FK
FOREIGN KEY (DEPT_NO) REFERENCES WITH NO CHECK OPTION DEPARTMENT
(DEPT_NO) ;
```



```
Log - (Untitled)
371 proc append base=mytera2.employee (BULKLOAD=YES
372                                     FBUSIZE=8000
373                                     BL_LOG=TEST_LOAD)
374       data=work.employee_data;
375 run;

NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 5 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 5 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           2.54 seconds
      user cpu time       0.21 seconds
      system cpu time     0.04 seconds
      Memory              250k
      OS Memory           13168k
      Timestamp           5/2/2008  2:31:49 PM
```

### 5.2.5.5 FastLoad a Table With FBUSIZE= Set Too High



```
Log - (Untitled)
253 proc append base=mytera2.cm_financial_account_new (fastload=YES FBUSIZE=66571
253! BL_LOG=TEST_LOAD)
254       data=work.cm_financial_account;
255 run;

NOTE: Appending WORK.CM_FINANCIAL_ACCOUNT to MYTERA2.cm_financial_account_new.
NOTE: There were 361 observations read from the data set WORK.CM_FINANCIAL_ACCOUNT.
NOTE: 360 observations added.
NOTE: The data set MYTERA2.cm_financial_account_new has . observations and 84 variables.
ERROR: Fastload failed waiting on a connection (in trfasbuf): Invalid parcel sequences during
Load.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           2.06 seconds
      user cpu time       0.20 seconds
      system cpu time     0.00 seconds
      Memory              956k
      OS Memory           13168k
      Timestamp           4/30/2008  4:53:23 PM

NOTE: The SAS System stopped processing this step because of errors.
```

If you encounter this problem, you can delete all the rows from the table and resubmit the job. Dropping and recreating the target table will also work. If the table has been defined properly, this is unnecessary.

### 5.2.6 Resolving Common Bulk Loading Problems

There are going to be times when your load jobs will fail. There are two types of failure.

One type of failure is easy. There is a syntax problem or perhaps there is an index that is preventing FastPath loading. In this situation no data is placed into the table. You can correct the problem and rerun. You may need to remove RI constraints or drop a hash index. By now you know what to look for.

The other type of failure is a little more difficult. Let's say that you have run into a space problem in your database and the job fails. In this situation there may be data in the table. That means that you cannot simply restart the job.

If you were using FastLoad, then the target table was initially empty. In this case, it is easiest to rerun the FastLoad job from the start. You will need to have the Teradata DBA address the space issue in your database before you try again.

If you were using a load method other than FastLoad or MultiLoad, there may be a lot of data in the table. You may decide to write a program to add the remaining data to the table. This may be complicated. You will need to know the last observation added to the table. Your program should add the remaining data to the table. If there is not a lot of data to add you can use a simple INSERT statement. If there is a lot of data to add you want to use something that performs better. You want to use the MultiLoad protocol – our next topic.



## 6 Loading SAS Data into Teradata (MultiLoad)

### 6.1 MultiLoad Protocol Restrictions

Let's look at the list of MultiLoad restrictions, again.

In order to load data using MultiLoad:

- The target table must have no unique secondary, join, or hash indexes defined on it.
- The target table must have no triggers defined on it.
- The target table must have no standard referential integrity or batch referential integrity defined on it (Soft Referential Integrity is allowed).
- The MultiLoad input file must have data to qualify all columns defined in the Primary Index of the target table.

Here are the MultiLoad protocol restrictions and how to verify that your table meets them.

Restriction	How to verify
Target table must have no unique secondary index, join index, or hash index defined on it.	<p>Queries similar to this can be used to find indexes defined on the tables you are planning to load.</p> <pre>SELECT *   FROM DBC.INDICES  WHERE TableName IN ('EMPLOYEE','DEPARTMENT')  ORDER BY DatabaseName, TableName, IndexName; IndexType  J = Join Index N = Hash Index S = Secondary Index (and UniqueFlag = T) V = Value Ordered Secondary Index</pre>
Target table must have no triggers defined on it.	<p>Queries similar to this can be used to find triggers defined on the tables you are planning to load.</p> <pre>SELECT *   FROM DBC.TriggersX  WHERE TableName='EMPLOYEE' ;</pre>
Target table must have no standard or batch RI defined on it.	<p>Queries similar to these can be used to find parent child relationships.</p> <pre>SELECT *   FROM DBC.ALL_RI_CHILDRENX  WHERE ParentDB='TESTDB' ;</pre>

	<p>Or</p> <pre>SELECT *   FROM DBC.ALL_RI_PARENTSX  WHERE ParentDB='TESTDB' ;</pre>
--	---

There are differences between FastLoad and MultiLoad.

MultiLoad can be used to load tables which contain data.	FastLoad can only load tables which are empty.
MultiLoad can be used to load duplicate rows into multiset tables.	FastLoad cannot load duplicate rows into any tables.

You can use MultiLoad to load data into a view but there is a problem. SAS 9.1.3 had an issue with MultiLoading into a view – it would not work. The problem was only with MULTILOAD=; FastLoading (BULKLOAD= and FASTLOAD=) works with no problem.

The MultiLoad into a view problem has been fixed by [Hot Fix E9TE10](#). You may have to search for the most current Hot Fix.

The SAS/ACCESS Interface to Teradata makes using the MultiLoad utility extremely easy, but there are some tricks that will help you out a great deal.

## 6.2 Configuring SAS for MultiLoad

SAS uses an API to access the capabilities of Teradata's Fastload protocol. Multiload is different: SAS uses the actual Multiload utility from Teradata. This means that there is configuration required. These steps are covered in the various configuration guides for the various operating systems (OS). Let's discuss Windows since you will probably do this first on your laptops.

The Teradata MultiLoad Utility (mload.exe) must be installed on the system running SAS. I have the TTU 13 installed and mload.exe lives in the C:\Program Files\Teradata\Client\13.0\bin directory. You may need to search for the file on your computer.

You will also need to know where the sasmlam.dll and sasmlne.dll files live. The location specified in the configuration guide does not match my system. I had to do a search to find these files.

The documentation states they live in: !sasroot\core\sasext.

On my system I found them in: C:\Program Files\SAS\SASFoundation\9.2\access\sasexe

Once you find it you will need to update the %PATH% environment variable for your system. In Windows XP you select Start→Control Panel. Double-click System. Select the Advanced tab. Click the Environment Variables button. Change the Path in the System Variables Add the two paths to the Path variable. Make certain that you separate the paths with semicolons (;).

The settings will be correct after a system reboot. In the meantime you can open a command prompt and enter a command similar to this:

```
Set %PATH%=%PATH%; C:\Program Files\Teradata\Client\13.0\bin;C:\Program
Files\SAS\SASFoundation\9.2\access\sasexe
```

Make sure you test to ensure that the system is properly configured.

### 6.2.1 MULTILOAD=

The MULTILOAD= data set option tells SAS whether or not to use the Teradata MultiLoad facility to insert data into a database table or view. It is important that you realize that there is no MULTILOAD= LIBNAME statement option.

If you submit a LIBNAME statement containing a MULTILOAD= option an error stating “Invalid option name MULTILOAD” will appear in the SAS log.

#### **MULTILOAD=YES|NO (Data Set option only)**

**YES** – Teradata MultiLoad protocol is used when appending or inserting data into Teradata tables using SAS/ACCESS software.

**NO** – The MultiLoad protocol is not used when appending or inserting data into the Teradata table. Data is inserted one row at a time.

All of the MultiLoad restrictions that were discussed earlier apply. In fact, later in this paper we will see how SAS reacts when we violate those rules.

### 6.2.2 ML\_LOG=

ML\_LOG= specifies a prefix that will be used when creating the temporary tables that will be used by the Teradata MultiLoad process. It is optional but highly recommended since it makes the temporary tables easier to identify in the database

Teradata MultiLoad uses four temporary tables when it performs its load operation. If you do not specify a value for ML\_LOG= then “SAS\_ML\_” will be used. Here is what the default tables will look like:

Restart table	SAS_ML_RS_randomnumber
Acquisition error table (ERROR TABLE 1)	SAS_ML_ET_randomnumber
Application error table (ERROR TABLE 2)	SAS_ML_UT_randomnumber
Work table	SAS_ML_WT_randomnumber

If you are loading lots of Teradata tables then you will want to use ML\_LOG= to specify a meaningful name. It will make debugging problems much easier.

There is another reason you may need to specify this option. Using the default value can be a problem in production Teradata environments. By default, the SAS/ACCESS interface to Teradata will create these temporary tables in the database that the SAS Library points to. EDP auditors are quick to point out that applications are not allowed, under any circumstances, to create tables in production databases.

#### **ML\_LOG= a meaningful table name**

To specify this option you must specify MULTILOAD=YES.

The Teradata bulk-load facility will create error tables based on this name. Use meaningful names.

You can place the utility tables in a separate database by prepending the database name and a period to the value specified in the BL\_LOG= option. Let's assume that we need to put the utility tables in the TEMPUTIL Teradata database and we want the table names to all start with INITIAL\_LD. Here is what our BL\_LOG= statement would look like.

BL\_LOG=TEMPUTIL.INITIAL\_LD

If you happen to leave the ML\_LOG= option on a dataset being loaded using MULTILOAD=YES it will not be flagged as an error. The error, work, and restart tables will not be created using the value specified. In other words, the tables will be created if needed but they will follow the default naming convention.

```
DATA TERAPROD.CUSTOMERS (MULTILOAD=YES ML_LOG=CUSTOMERS) ;
  SET WORK.NEW_CUSTOMERS ;
RUN ;
```

### 6.2.3 ML\_RESTART=

The Teradata MultiLoad utility and protocol allows users to restart jobs. The restart table keeps track of the checkpoint information required to restart the load job. Checkpoint entries are notes that are added to the restart table that detail what has been processed by the MultiLoad job. Think of it as a list of completed tasks.

If a MultiLoad job fails, the restart table will tell a subsequent job the data to start processing.

You can use the ML\_RESTART= option to specify a name for this table. If you use this option you should not include the ML\_LOG= option.

#### **ML\_RESTART= a meaningful table name**

To specify this option you must also include the MULTILOAD=YES.

The table will be created in the same database, or userid, specified in the LIBNAME statement. If you specify an invalid table name then the default SAS form of the table name will be used:

- SAS\_ML\_WT\_randomnumber.
- SAS\_ML\_RS\_randomnumber

Here is sample DDL from a restart table. This table was created due to a SAS/ACCESS MultiLoad job. If you do not specify ML\_ERROR1= or ML\_LOG= SAS will generate the name for you.

```
CREATE SET TABLE TESTDB.SAS_ML_RS_1527691963,FALLBACK,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM = DEFAULT
(
  LogType INTEGER,
  Seq INTEGER,
  ReqRC INTEGER,
  ReqType INTEGER,
  ReqLen INTEGER,
  ReqMsg VARCHAR(255) CHARACTER SET UNICODE NOT CASESPECIFIC,
  SysInfo VARBYTE(255) ,
  MiscInt1 INTEGER,
  MiscInt2 INTEGER,
  MiscInt3 INTEGER,
  MiscInt4 INTEGER,
  MiscInt5 INTEGER,
  MiscInt6 INTEGER,
  MiscInt7 INTEGER,
  MiscInt8 INTEGER,
  MLoadSeq INTEGER DEFAULT 0 ,
  MLoadImpSeq INTEGER,
  MLoadSrcSeq INTEGER,
  CkptInterval INTEGER,
  MLoadCkpt VARBYTE(1024) ,
```

```
RunDate DATE FORMAT 'YY/MM/DD' DEFAULT DATE ,
RunTime FLOAT DEFAULT TIME )
UNIQUE PRIMARY INDEX ( LogType ,Seq ,MLoadSeq );
```

## 6.2.4 ML\_ERROR1=

The ML\_ERROR1= specifies the name of the table that is used to store data that caused a problem during the acquisition phase of the MultiLoad process.

The MultiLoad Acquisition Phase:

1. Imports data from the specified input data source
2. Evaluates each record according to specified application conditions
3. Load the select records into the worktables in the Teradata Database.

### **ML\_ERROR1=a meaningful table name**

To specify this option you must also include the MULTILoad=YES option.

The table will be created in the same database, or userid, specified in the LIBNAME statement. If you specify an invalid table name then the default SAS form of the table name will be used:

SAS\_ML\_ET\_randomnumber.

Please note that MultiLoad can support functionality that is not available via SAS/ACCESS. We are not going to cover the non-supported functionality of MultiLoad.

Here is sample DDL from the acquisition phase error table. This table was created due to a SAS/ACCESS MultiLoad job. If you do not specify ML\_ERROR1= or ML\_LOG= SAS will generate the name for you.

```
CREATE MULTISet TABLE testdb.SAS_ML_ET_1527691963 ,FALLBACK ,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM = DEFAULT
(
ImportSeq BYTEINT FORMAT 'ZZ9' NOT NULL,
DMLSeq BYTEINT FORMAT 'ZZ9' NOT NULL,
SMTSeq BYTEINT FORMAT 'ZZ9' NOT NULL,
ApplySeq BYTEINT FORMAT 'ZZ9' NOT NULL,
SourceSeq INTEGER FORMAT '-----9' NOT NULL,
ErrorCode INTEGER FORMAT '-----9' NOT NULL,
ErrorField CHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL,
HostData VARBYTE(64000) NOT NULL)
UNIQUE PRIMARY INDEX ( ImportSeq ,DMLSeq ,SMTSeq ,ApplySeq ,SourceSeq );
```

### 6.2.5 ML\_ERROR2=

The ML\_ERROR2= specifies the name of the table that is used to store data that caused a problem during the application phase of the MultiLoad process.

The MultiLoad Application Phase:

1. Acquires locks on the specified target tables and views in the Teradata Database
2. Inserts the data from the temporary work tables into the target tables or views in the Teradata Database
3. Updates the error tables associated with each MultiLoad task

#### **ML\_ERROR2=a meaningful table name**

To specify this option you must also include the MULTILOAD=YES option.

The table will be created in the same database, or userid, specified in the libname statement. If you specify an invalid tablename then the default SAS form of the table name will be used.

SAS\_ML\_UT\_randomnumber.

Please note that MultiLoad can support functionality that is not accessible via SAS/ACCESS. Those steps have been omitted from the description.

Here is sample DDL from application phase error table. This table was created using a SAS/ACCESS MultiLoad job. If you do not specify ML\_ERROR2= or ML\_LOG= SAS will generate the name for you.

```
CREATE MULTISSET TABLE testdb.SAS_ML_UT_1527691963 ,FALLBACK ,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL,
  CHECKSUM = DEFAULT
(
  x FLOAT,
  ImportSeq BYTEINT FORMAT 'ZZ9' NOT NULL,
  DMLSeq BYTEINT FORMAT 'ZZ9' NOT NULL,
  SMTSeq BYTEINT FORMAT 'ZZ9' NOT NULL,
  ApplySeq BYTEINT FORMAT 'ZZ9' NOT NULL,
  SourceSeq INTEGER FORMAT '-----9' NOT NULL,
  Uniqueness INTEGER FORMAT '-----9' NOT NULL,
  DBCErrCode INTEGER FORMAT '-----9' NOT NULL,
  DBCErrField CHAR(30) CHARACTER SET LATIN NOT CASESPECIFIC NOT
  NULL)
UNIQUE PRIMARY INDEX ( ImportSeq ,DMLSeq ,SMTSeq ,ApplySeq ,SourceSeq ,
  Uniqueness ,DBCErrCode );
```

## 6.2.6 ML\_WORK=

The ML\_WORK= option specifies the name of a temporary table. This table is used by Teradata to store intermediate information during the acquisition phase. This data is then read and placed into the target tables or views during the application phase.

### **ML\_WORK= temporary-table-name**

To specify this option you must also include the MULTILOAD=YES option.

The table will be created in the same database, or userid, specified in the LIBNAME statement. If you specify an invalid table name then the default SAS form of the table name will be used:

SAS\_ML\_WT\_randomnumber.

The table `testdb.SAS_ML_WT_1527691963` was created using a SAS/ACCESS MultiLoad job. If you do not specify ML\_WORK= or ML\_LOG= SAS will generate the name for you. Here is sample DDL from a work table.

```
CREATE MULTISSET TABLE testdb.SAS_ML_WT_1527691963 ,NO FALLBACK ,
      NO BEFORE JOURNAL,
      NO AFTER JOURNAL,
      CHECKSUM = DEFAULT
      (
        Field1 VARBYTE(64) FORMAT 'X(64)' NOT NULL)
PRIMARY INDEX ( Field1 );
```

It is important to note that the columns defined in the SAS data set or Teradata tables are not defined in this work table. In fact, this is a special type of table and you cannot read it with normal SQL. This table is used by Teradata and is not mean to be read by a user. Here is an example:

```
SELECT Field1
FROM TEST.SAS_ML_WT_1527691963
```

If you issue the SQL statement you will get this error:

```
2570:  Operation not allowed: testdb.SAS_ML_WT_1527691963 is a MLoad
worktable.
```



### 6.2.7 ML\_CHECKPOINT=

The ML\_CHECKPOINT= invokes a Teradata checkpoint every *n* minutes.

#### **ML\_CHECKPOINT=*n***

ML\_CHECKPOINT= will cause a Teradata checkpoint every *n* minutes. If you use MULTILoad=YES but do not specify a value for ML\_CHECKPOINT= then no checkpoints are used. This is the same as setting ML\_CHECKPOINT=0.

Using a value of 0 means commit after the table is loaded. This works well on smaller loads.

### 6.2.8 SLEEP=

SLEEP= and TENACITY= are related and often used together. There is a limit to the number of load jobs that can run on a Teradata system. If this number is maxed-out then your MultiLoad job will not run. The SLEEP= option allows you to keep trying. The SLEEP= option tells the job to try to LOGON every *n* minutes.

You can use this query to determine how many utilities are currently running on the system:

```
LOCK DBC.SessionInfo FOR ACCESS
SELECT PARTITION
      , COUNT(DISTINCT(LogonSequenceNo))
FROM DBC.SessionsInfo
WHERE Partition in ('MLOAD' , 'FASTLOAD' , 'EXPORT')
GROUP BY Partition;
```

#### **SLEEP=*number of minutes***

The default is 6 minutes. This is a Teradata default. If you specify this option you must set it to a value greater than 0. If you set SLEEP= 0 then your SAS code will fail and an error will be displayed in the SAS Log.

### 6.2.9 TENACITY=

TENACITY= and SLEEP= are related and often used together. There is a limit to the number of load jobs that can run on a Teradata system. If this number is maxed-out then your MultiLoad job will not run. The TENACITY= option tells the job to try to run for *n* hours. If the load on the system prevents the job from running in the number of hours specified by TENACITY= then the job will fail.

**TENACITY=number of hours**

The default is 4 hours. This is the Teradata default. If you specify this option you must set it to a value greater than 0. If you set TENACITY=0 then your SAS code will fail and an error will be displayed in the SAS Log.

**6.2.10 MBUFFSIZE=**

SAS uses memory buffers to transfer data from SAS to Teradata. There are two data set options that you can use to tune this process. MBUFFSIZE is one of these.

**MBUFFSIZE=size-of-shared-memory-buffers**

MBUFFSIZE specifies the size of the memory buffers that will be used. This is a numeric value between the size of the row being loaded and 1 MB. The default value is 64k. By default, 2 shared memory buffers are used for the transfer.

You can use the BUFFERS= option to change the number of buffers. You can specify 1 to 8 buffers.

Determining the most efficient values of MBUFFSIZE= and BUFFERS= will require experimentation on your specific system with your data.

**6.2.11 MultiLoading Teradata Tables Using SAS**

The mechanics of loading your Teradata tables using SAS/ACCESS MultiLoad is simple. Here is an example using PROC APPEND (notice that the MULTILoad= alias is used instead of BULKLOAD=). Keep in mind, we have not specified the ML\_CHECKPOINT= option. That means that Teradata will commit the changes after the entire table is loaded.

```
PROC APPEND
    BASE=mytera2.cm_financial_account_new
        (MULTILOAD=YES)
    DATA=work.cm_financial_account;
RUN;
```

PROC APPEND is a great choice for loading data into existing tables but it isn't the only way to do it. Loading data into existing tables is very common during ETL processing. As a general rule, you want to be careful when creating a Teradata table and loading it on the fly. The reason: First and foremost, your DBA probably will not allow you do create a table and second, you will want more control over the data types used in Teradata.

Here is an example of using PROC SQL to load SAS data into an existing Teradata table. Unlike the FASTLOAD= example, we can use this technique on tables which contain data. Let's specify table names for our restart, work, and error tables.

```
proc sql;
  insert into mytera2.cm_financial_account_new
    (MULTILOAD=YES
     ML_RESTART = FINANCIAL_RESTART
     ML_ERROR1  = FINANCIAL_ERR1
     ML_ERROR2  = FINANCIAL_ERR2
     ML_WORK    = FINANCIAL_WORK)
    select * from work.cm_financial_account;
quit;
```

The MultiLoad protocol, MULTILOAD=YES, will make the job run quickly.

Here is an example that you will want to think about before copying. Notice that there is no BULKLOAD=, FASTLOAD=, or MULTILOAD= options on the mytera2.cm\_financial\_account\_new dataset. This means that it will issue INSERT statements.

```
proc sql;
  insert into mytera2.cm_financial_account_new (CHECKPOINT=50000)
    select * from work.cm_financial_account;
quit;
```

On our test system, this code will not run. It executes for a while and then fails. A rollback is issued and the added rows are removed from the table. This takes a lot of time (1 hour and 7 minutes). Here is the error that is encountered:

**ERROR: Teradata insert: The AMP Lock table has overflowed.**

The slowness is due to the fact that the rows are inserted using one unit of parallelism and inserted one at a time in a sequential process. The error was caused by the requirement for each inserted row to have a separate lock in the AMP (unit of parallelism) lock table.

The MultiLoad protocol is interesting. In creating this document, we encountered an odd error. We created a table using SQL code similar to this:

```
CREATE MULTISSET TABLE TESTDB.CM_FINANCIAL_ACCOUNT ,FALLBACK ,
  NO BEFORE JOURNAL,
  NO AFTER JOURNAL,
  CHECKSUM = DEFAULT
  (
    ACCOUNT_RK INTEGER,
    PARTY_RK INTEGER,
    INQUIRIES_ACCT_LAST_3_MTHS_CNT DECIMAL(16,0))
PRIMARY INDEX ( ACCOUNT_RK );
```

We then tried to use the MultiLoad protocol to put data into the table. It didn't work.

```
LIBNAME mytera2 TERADATA
  USER=userid PASSWORD=password SERVER=terasrv;
```

```
proc append base=mytera2.cm_financial_account (MULTILOAD=YES)
            data=work.cm_financial_account;
run;
```

We checked the error message:

```
UTY0005 Bad data in the FIELD command at position 7, the name beginning
with ""INQUIRIES_PROD_LAST_3_MTHS_CN"
```

The column name INQUIRIES\_ACCT\_LAST\_3\_MTHS\_CNT is the maximum length for a Teradata column. By default SAS/ACCESS places quotes around the name. This made the name too long and causes the error.

The solution to the problem is the PRESERVE\_COL\_NAMES= option. If you set this to NO the PROC APPEND code works fine.

```
LIBNAME mytera2 TERADATA
        USER=userid PASSWORD=password SERVER=terasrv
        PRESERVE_COL_NAMES=NO;

proc append base=mytera2.cm_financial_account (MULTILOAD=YES)
            data=work.cm_financial_account;
run;
```

## 6.2.12 Upserting With MultiLoad Using SAS

Beginning with the SAS 9.2 SAS/ACCESS Interface to Teradata you can use the MultiLoad Protocol to perform UPSERT operations. UPSERT allows you combine an UPDATE and INSERT into one step. An UPDATE is performed when a target row exists in the table and an INSERT is performed when the target row does not exist. Currently, PROC APPEND is the only way that you can take advantage of MultiLoad upserts using SAS.

To use the UPSERT functionality you will need to use the UPSERT= data set option.

### **UPSERT=YES|NO**

YES – perform upsert using the MultiLoad protocol to load data into Teradata.

NO – Only inserts are performed. Existing rows are ignored.

Here is an example.

We want to create a Teradata table based on this SAS data set.

```
data work.test;
  length a 8. b $10.;
  a=1; b='Hi'; output;
  a=2; b='There'; output;
  a=3; b='World'; output;
run;
```

We will take this SAS data set and load it into Teradata using this code. Since we are creating this table it will be empty; we can use the FastLoad protocol to load it.

```
proc append base=mytera.greeting (FASTLOAD=YES) data=test;
run;
```

We need to take a look at the data just to make certain that it is correct. We can use this PROC SQL code, from within SAS, to do that:

```
proc sql;
  select a, b
    from mytera.greeting
   order by a;
run;
```

Here is what our data looks like:

a	b
1	Hi
2	There
3	World

Next, we need a transaction file. This data set will serve that purpose, nicely.

```
data work.upsert;
  length a 8. b $10.;
  a=3; b='Cruel'; output;
  a=4; b='World'; output;
run;
```

Column 'a' is our primary index. When we run PROC APPEND using UPSERT=YES the row with 'a' equal to 3 will be updated. Its value will become 'Cruel.' There is currently no row having 'a' equal 4, so one will be added.

This code will perform the MultiLoad UPSERT operation.

```
proc append base=mytera.greeting (MULTILOAD=YES UPSERT=YES)
          data=work.upsert;
run;
```

Here are the results. Notice that our expected changes have been applied.

a	b
1	Hi
2	There
3	Cruel
4	World

By default, upsert will use the primary index columns, in our example that is “column a” of the target table to generate the WHERE clause that will drive upsert processing. If you would like to use additional columns in the upsert where clause you can specify them using the UPSERT\_WHERE= option.

#### **UPSERT\_WHERE= list of columns**

List of columns – the columns to use when generating the where condition for upsert processing. You must include the primary index columns in this list.

Let’s assume that our example table has a column ‘c’ – not part of the primary index -- and that column should help determine when an update should occur. We will load this data into a Teradata table. Column ‘a’ will be the primary index. This is a MultiSet table so duplicate primary indexes are allowed.

```
data work.test;
  length a 8. c 8. b $10.;
  a=1; c=1; b='Hi'; output;
  a=2; c=1; b='There'; output;
  a=3; c=1; b='World'; output;
  a=3; c=2; b='World!'; output;
run;
```

We will take this SAS data set and load it into Teradata using this code. Since we are creating this table it will be empty and we can use the FastLoad protocol to load it.

```
proc append base=mytera.greeting (FASTLOAD=YES) data=test;
run;
```

We need to take a look at the data just to make certain that it is correct. We can use this PROC SQL code from within SAS to do that.

```
proc sql;
  select a, c, b
    from mytera.greeting
   order by a;
run;
```

Here is the result of the UPSERT operation:

a	c	b
1	1	Hi
2	1	There
3	2	World!
3	1	World

Column 'a' is our primary index. In this case we have duplicate primary indexes; notice that the digit 3 is listed twice. Let's assume that we want to change only one of the rows which have 3 as the primary index value. We can do that using the UPSERT\_WHERE= data set option.

When we run PROC APPEND using UPSERT=YES and UPSERT\_WHERE=(a c). The rows with 'a' equal to 3 will be updated with values based upon the value of the column 'c'.

Here is the transaction data that will be used in the UPSERT operation:

```
data work.upsert;
  length a 8. c 8. b $10.;
  a=3; c=1; b='311111'; output;
  a=3; c=2; b='322222'; output;
  a=4; c=1; b='World'; output;
run;
```

The row with 'a' = 3 and 'c' = 1 will have 'b' updated to '311111.' The row with 'a'=3 and 'c'=2 will have 'b' updated to '322222.' There is no row having 'a'=4, so it will be inserted into the table.

This code will perform the MultiLoad UPSERT operation:

```
proc append
  base=mytera.greeting (MULTILOAD=YES UPSERT=YES UPSERT_WHERE=(a c))
  data=work.upsert;
run;
```

Here is the result of the UPSERT operation:

a	c	b
1	1	Hi
2	1	There
3	2	322222
3	1	311111
4	1	World

### 6.2.13 Configuring the MultiLoad Utility

There are two problems you may face when dealing with MultiLoad. The first: How do you diagnose a problem. The second: How do I set MultiLoad options that are surfaced through the SAS/ACCESS interface.

Take a look at this SAS Log. On no! We have a problem.

```

113 proc append base=mytera2.employee (MULTILOAD=YES
114                               BL_LOG=TEST_LOAD)
115       data=work.employee_data;
116 run;

NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 5 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 5 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
ERROR: MultiLoad failed with DBS error 3 without taking any checkpoints.
Correct error and restart as an append process with dataset options ML_RESTART=C:\Documents
       Settings\sasxjb\BL_employee_12.dat, ML_ERROR1=, ML_ERROR2=, and ML_WORK=
If the first run used FIRSTOBS=n, use the same value for FIRSTOBS in the restart.
Otherwise use FIRSTOBS=1.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           46.51 seconds
      cpu time             0.03 seconds

NOTE: The SAS System stopped processing this step because of errors.

```

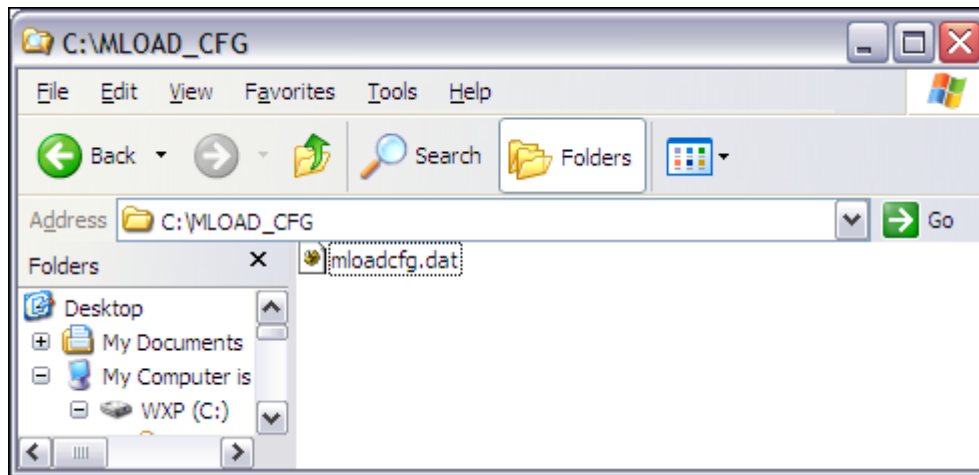
The specific cause of the problem is not mentioned in the SAS Log. We must set up our environment so that we can see the actual error message being returned by Teradata. The solution presented here is specific to the Windows environment. The setup for UNIX is very similar.

The first thing that we need to do is create a directory for MultiLoad configuration file. This file must be named **mloadcfg.dat**. At a minimum, this file should contain a line similar to this:

```
ERRLOG=c:\MLOAD_CFG\mload.txt
```

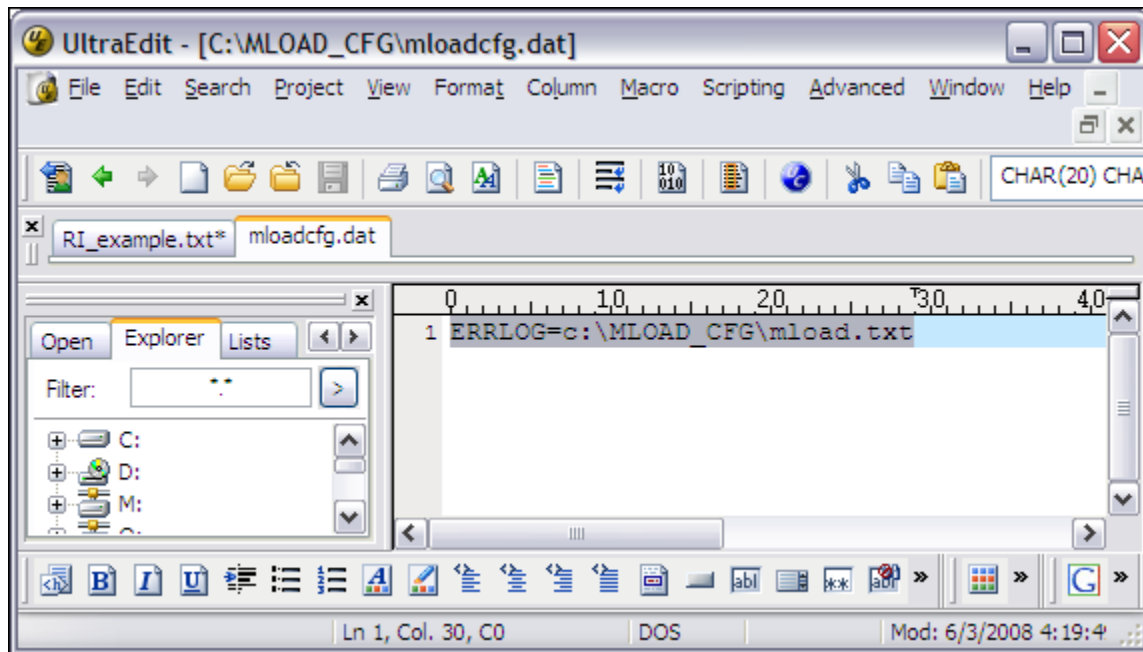


You can choose a different location or file name. Here is our setup:



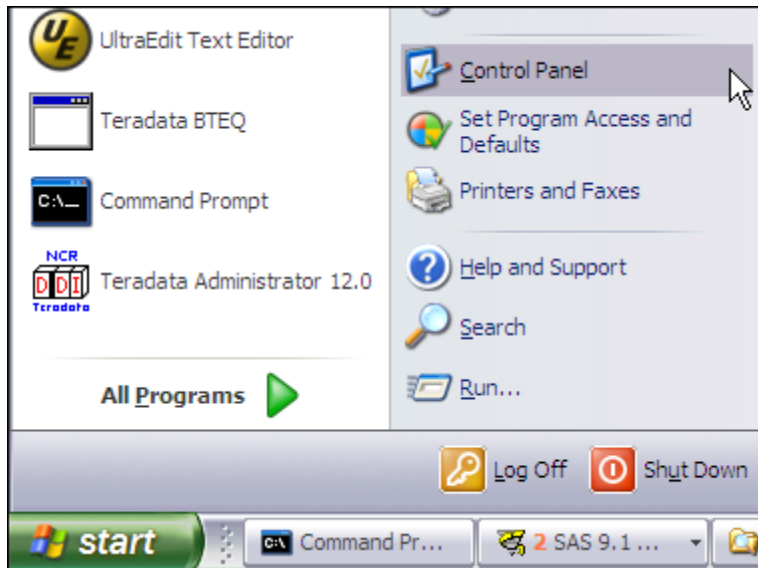
In this example we have created the file in the C:\MLOAD\_CFG directory. The directory you choose to house this file will need to be available to the machine running SAS. In this example we are creating a file named mload.txt. This file is where the MultiLoad utility error messages will be written.

Here are the contents of the mloadcfg.dat file:

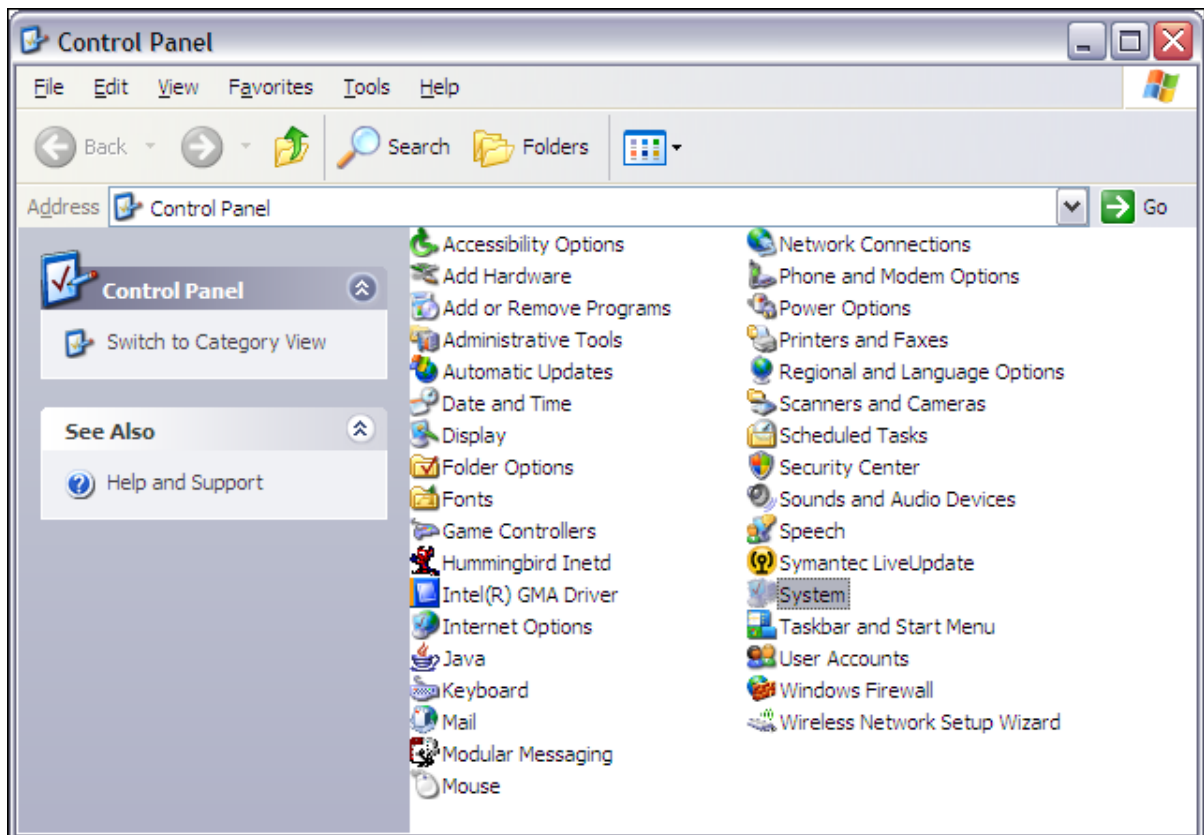


Next you will need to create an environment variable and point it to the directory you created for your configuration file.

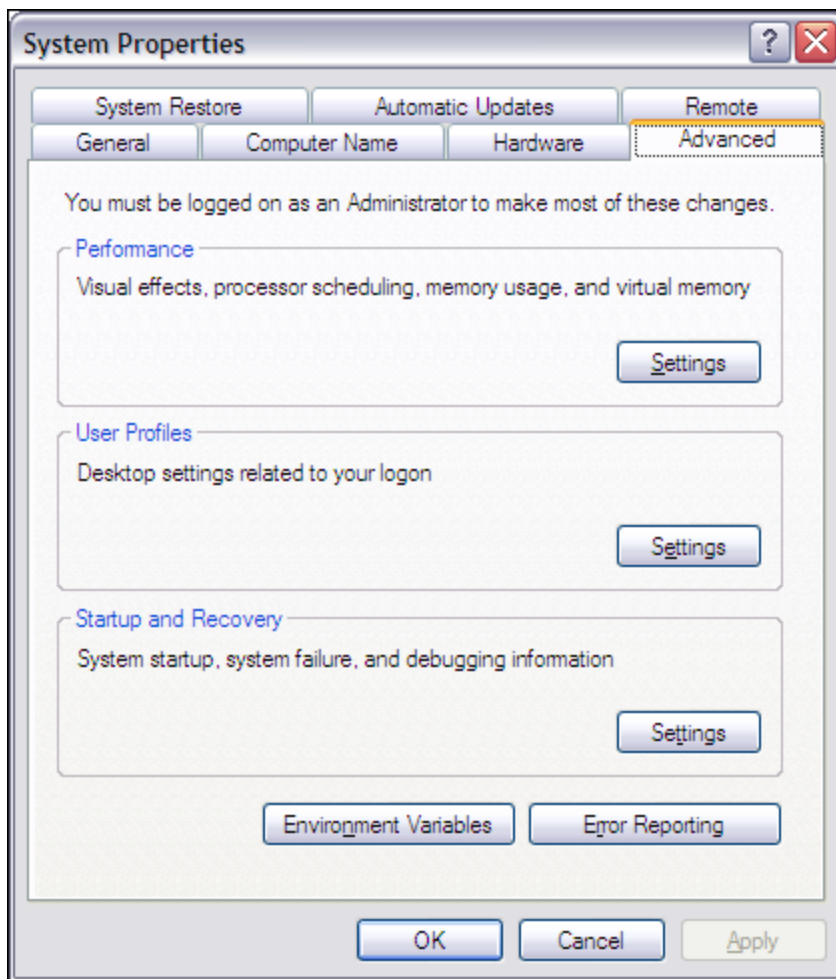
1. Select **Start → Control Panel**.



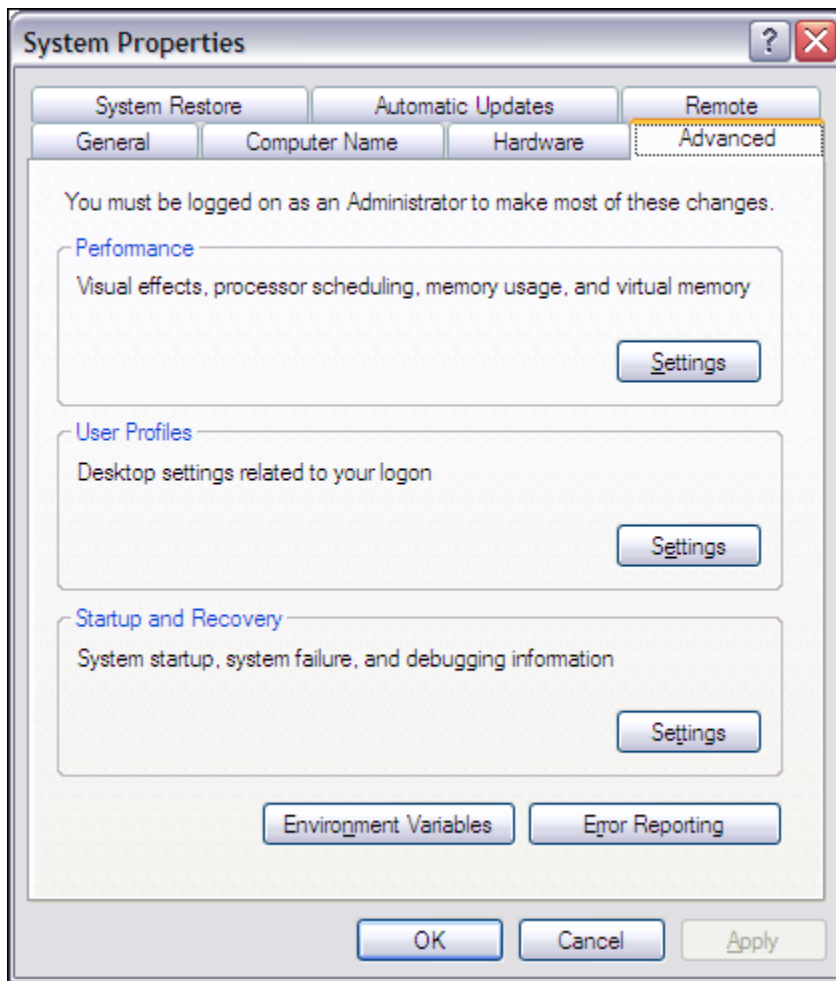
2. Select **System** to open the System dialog.




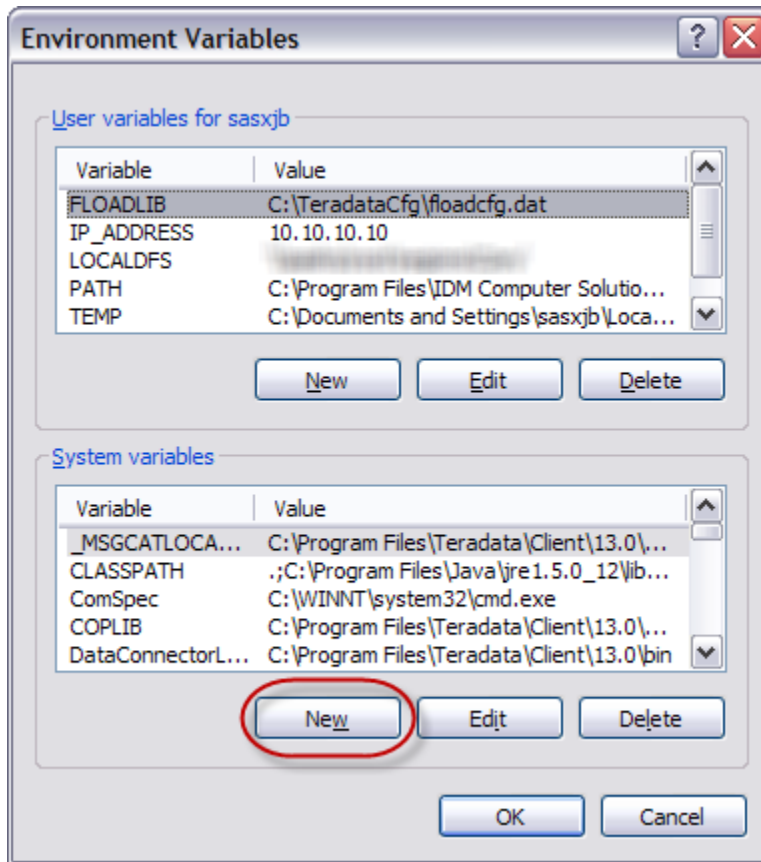
3. Select the **Advanced** tab.



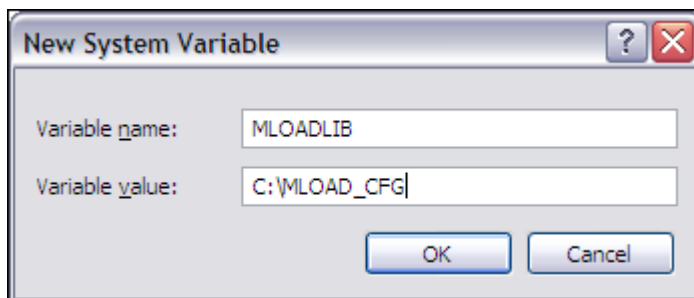
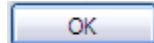
4. Select **Environment Variables** on the Advanced tab.



5. Select  under the System variables section.



6. Enter MLOADLIB for the Variable name and your directory for the Variable value. Select



7. Close the Environment Variables, System Properties and Control Panel dialogs.
8. In order for the MLOADLIB environment variable to take effect you must reboot your system. If you want to test this without rebooting your system you can manually create the environment variable by issuing the following command in a Command Prompt.

**set MLOAD=C:\MLOAD\_CFG**

In our testing we have found that the file containing the errors is overwritten whenever MultiLoad is used. You may need to programmatically handle this situation.

### 6.2.14 Specifying MINSESS and MAXSESS in MULTILOAD

Teradata DBAs are concerned about limiting the number of utility sessions being run in the Teradata Server. Using FASTLOAD= there is, unfortunately, no way to set this. That is because SAS option FASTLOAD= is implemented via an API. If this is a deal-breaker then you may want to look at the Teradata Parallel Transport options.

Since the MULTILOAD= option invokes the MultiLoad utility we can set these in the configuration mloadcfg.dat file that we discussed in the previous section.

#### **MINSESS=min-sessions**

This option lets you specify the minimum number of MultiLoad sessions required to run the job. This option must be set to a value greater than zero and less than the value specified by MAXSESS.

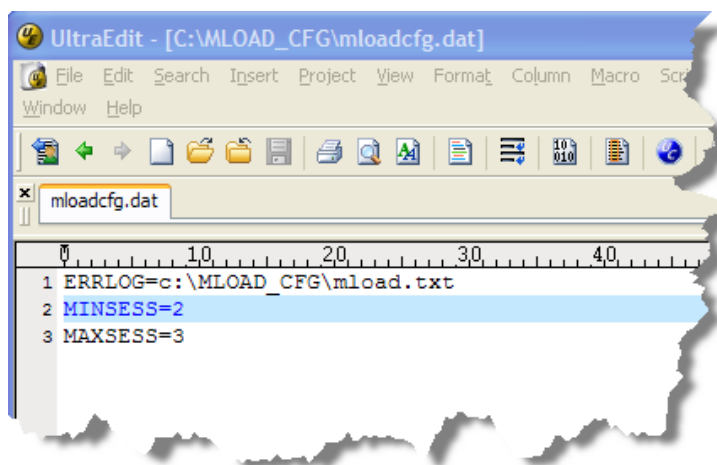
The default value is one.

#### **MAXSESS=max-sessions**

This option lets you set the maximum number of MultiLoad sessions logged into the Teradata Server. The value must be greater than one but less than the number of AMPs on the system. DBAs are very concerned about this. They do not like to see this value set too high and they consider the default too high.

The default value is one session for each AMP.

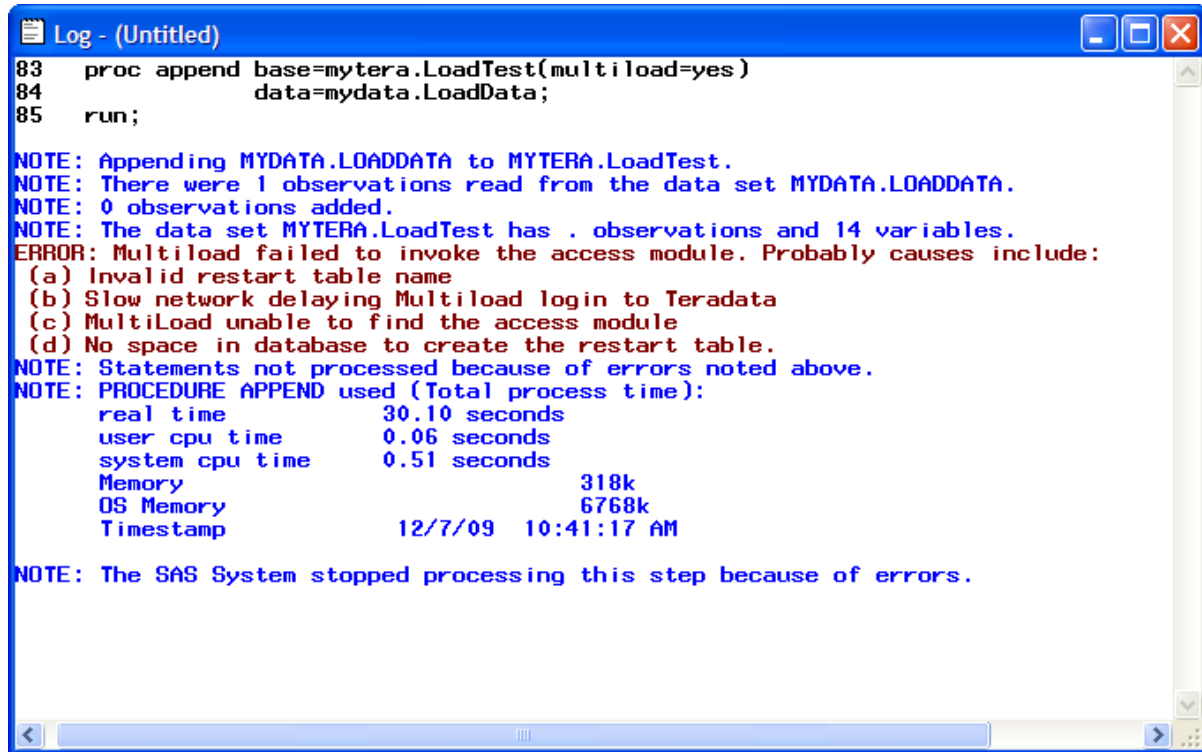
Here is our mloadcfg.dat file from a previous example. Notice that the MINSESS= and MAXSESS= options have been added to the file. The ERRLOG= is tagging along from a previous example.



For more information regarding the MultiLoad utility and the available options take a look at the Teradata MultiLoad Reference.

## 6.2.15 MULTILOAD Issues You May Encounter

### 6.2.15.1 MultiLoad Catchall Error



```

Log - (Untitled)
83 proc append base=mytera.LoadTest(multiload=yes)
84             data=mydata.LoadData;
85 run;

NOTE: Appending MYDATA.LOADDATA to MYTERA.LoadTest.
NOTE: There were 1 observations read from the data set MYDATA.LOADDATA.
NOTE: 0 observations added.
NOTE: The data set MYTERA.LoadTest has . observations and 14 variables.
ERROR: Multiload failed to invoke the access module. Probably causes include:
(a) Invalid restart table name
(b) Slow network delaying Multiload login to Teradata
(c) MultiLoad unable to find the access module
(d) No space in database to create the restart table.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           30.10 seconds
      user cpu time       0.06 seconds
      system cpu time     0.51 seconds
      Memory              318k
      OS Memory           6768k
      Timestamp           12/7/09 10:41:17 AM

NOTE: The SAS System stopped processing this step because of errors.

```

This error message is really a catchall. Possible causers are:

- There is a problem with the way that the MultiLoad utility is being called.
- There could be an improperly specified option in the mloadcfg.dat file.
- The user running the MultiLoad job does not have proper permissions on the databases.

Check to make sure that the paths were added properly to the %PATH% variable. Ensure that the options specified in the MultiLoad configuration file are correct. Verify that the user running the job has CREATE TABLE, DROP TABLE privileges on the database where the tables will be created.

If you encounter this error, review Sections 6.3.13 and 6.3.14 in this document.

### 6.2.15.2 MultiLoad a Table Having a Join Index Defined on It

We discussed this earlier but it is always a good idea to see exactly what happens when you “try it.”

If we define any of the following indexes on our EMPLOYEE table then our load will fail and display this error:

```
CREATE INDEX DEPT_NUSI_INDIX (DEPT_NO) ON EMPLOYEE;

CREATE JOIN INDEX EMPDEPT_INDIX,
  NO FALLBACK AS
  SELECT E.EMP_NO, E.DEPT_NO, E.EMP_NAME, E.EMP_NICKNAME, D.DEPT_NAME
  FROM EMPLOYEE E INNER JOIN DEPARTMENT D
    ON E.DEPT_NO = D.DEPT_NO
  PRIMARY INDEX (DEPT_NO);
```

```
Log - (Untitled)
113 proc append base=mytera2.employee (MULTILOAD=YES
114                                     BL_LOG=TEST_LOAD)
115     data=work.employee_data;
116 run;

NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 5 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 5 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
ERROR: MultiLoad failed with DBS error 3 without taking any checkpoints.
      Correct error and restart as an append process with dataset options ML_RESTART=C:\Documents
        Settings\sasxjb\BL_employee_12.dat, ML_ERROR1=, ML_ERROR2=, and ML_WORK=
      If the first run used FIRSTOBS=n, use the same value for FIRSTOBS in the restart.
      Otherwise use FIRSTOBS=1.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time      46.51 seconds
      cpu time       0.03 seconds

NOTE: The SAS System stopped processing this step because of errors.
```

Here is the error message that was written to the error log file:

```
UTY0805 RDBMS failure, 5467: Cannot drop, MLOAD, or RESTORE a table with
join or hash indexes.
```

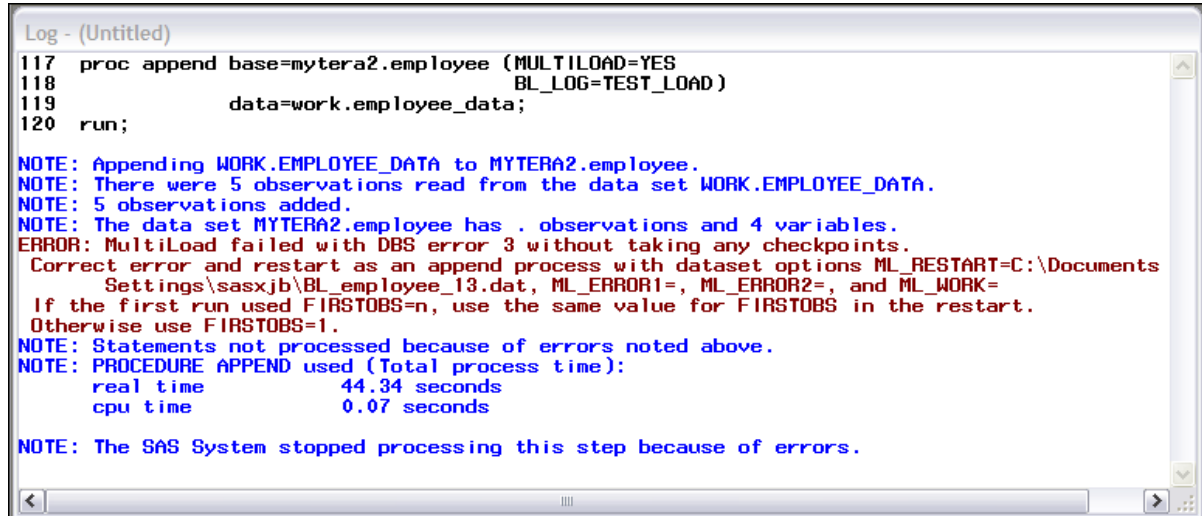
Remember, we created this file, C:\MLOAD\_CFG\mload.txt in the previous section.



### 6.2.15.3 MultiLoad a Table With a Join Index Defined on It

Although the hash index is not specifically mentioned in the error message it is not allowed. Notice that the error messages are the same when you have a prohibited index defined on the table.

```
CREATE HASH INDEX EMP_NAME (EMP_NAME, EMP_NICKNAME)
      ON EMPLOYEE BY (EMP_NAME)
      ORDER BY HASH      (EMP_NAME);
```



```
Log - (Untitled)
117 proc append base=mytera2.employee (MULTILOAD=YES
118                                     BL_LOG=TEST_LOAD)
119     data=work.employee_data;
120 run;

NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 5 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 5 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
ERROR: MultiLoad failed with DBS error 3 without taking any checkpoints.
      Correct error and restart as an append process with dataset options ML_RESTART=C:\Documents
      Settings\sasxjb\BL_employee_13.dat, ML_ERROR1=, ML_ERROR2=, and ML_WORK=
      If the first run used FIRSTOBS=n, use the same value for FIRSTOBS in the restart.
      Otherwise use FIRSTOBS=1.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time          44.34 seconds
      cpu time           0.07 seconds

NOTE: The SAS System stopped processing this step because of errors.
```

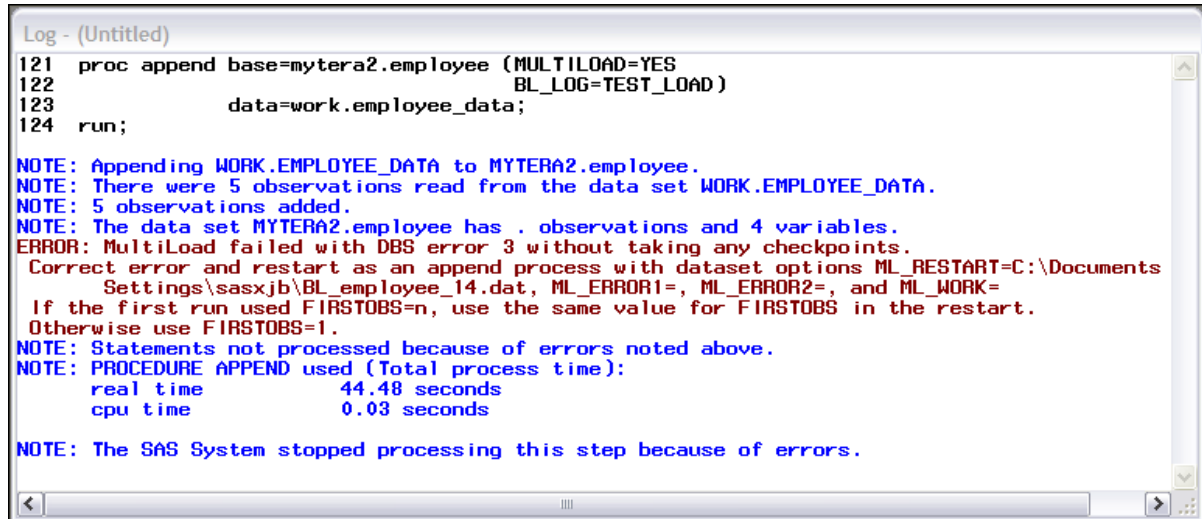
Here is the error message that was written to our error log file (mload.txt):

```
UTY0805 RDBMS failure, 5467: Cannot drop, MLOAD, or RESTORE a table with
join or hash indexes.
```

### 6.2.15.4 MultiLoad a Table With RI Defined

If you have RI or Batch RI defined on the table then you will see “referenced table” mentioned in the error message.

```
ALTER TABLE EMPLOYEE ADD CONSTRAINT DEPT_NO_FK
  FOREIGN KEY (DEPT_NO) REFERENCES DEPARTMENT (DEPT_NO);
```



```
Log - (Untitled)
121 proc append base=mytera2.employee (MULTILOAD=YES
122         BL_LOG=TEST_LOAD)
123         data=work.employee_data;
124 run;

NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 5 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 5 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
ERROR: MultiLoad failed with DBS error 3 without taking any checkpoints.
       Correct error and restart as an append process with dataset options ML_RESTART=C:\Documents
       Settings\sasxjb\BL_employee_14.dat, ML_ERROR1=, ML_ERROR2=, and ML_WORK=
       If the first run used FIRSTOBS=n, use the same value for FIRSTOBS in the restart.
       Otherwise use FIRSTOBS=1.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used (Total process time):
      real time          44.48 seconds
      cpu time           0.03 seconds

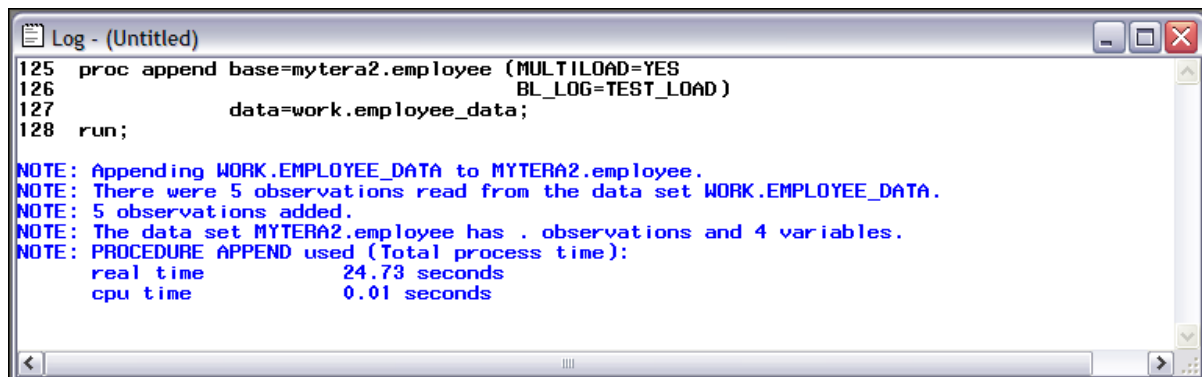
NOTE: The SAS System stopped processing this step because of errors.
```

Here is the error message that was written to our error log file (mload.txt):

```
UTY0805 RDBMS failure, 3971: MLoad target table cannot be a referenced
table or contain any referential constraint(s).
```

If Soft RI is defined on a table then you can load it. Remember, Soft RI tells the optimizer about the relationship but the relationship is not enforced. The **WITH NO CHECK OPTION** tells Teradata that this is Soft RI.

```
ALTER TABLE EMPLOYEE ADD CONSTRAINT DEPT_NO_SOFT_RI_FK
  FOREIGN KEY (DEPT_NO) REFERENCES WITH NO CHECK OPTION DEPARTMENT
  (DEPT_NO);
```



```
Log - (Untitled)
125 proc append base=mytera2.employee (MULTILOAD=YES
126         BL_LOG=TEST_LOAD)
127         data=work.employee_data;
128 run;

NOTE: Appending WORK.EMPLOYEE_DATA to MYTERA2.employee.
NOTE: There were 5 observations read from the data set WORK.EMPLOYEE_DATA.
NOTE: 5 observations added.
NOTE: The data set MYTERA2.employee has . observations and 4 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time          24.73 seconds
      cpu time           0.01 seconds
```

## 6.2.16 Resolving Common Bulk Loading Problems

There are going to be times when your load jobs will fail. There are two types of failure.

One type of failure is easy. There is a syntax problem or perhaps there is an index that is preventing FastPath loading. In this situation no data is placed into the table. You can correct the problem and rerun the job. You may need to remove RI constraints or drop a hash index. By now you know what to look for.

The other type of failure is a little more difficult. Let's say that you have run into a space problem in your database and the job fails. In this situation there may be data in the table. That means that you cannot simply restart the job. Once MultiLoad begins to populate the target table with data it must run to completion. If the MultiLoad job is not run to completion, the target table may not be recoverable.

You may be able to fix the problem by dropping and recreating the table, and restoring the initial data if the initial data is available from an archive source. If the initial data is not archived, you must restart and run the MultiLoad job to completion.

## 7 A Real Case Study

During any SAS and Teradata project you will be faced with decisions. One of the most important decisions is going to be, “What’s the best way to get my SAS data into Teradata?” Hey, it’s the subject of this paper. Unfortunately, this question is typically hidden in a question like, “Can you help me make my Teradata loads run faster?” That is the question that prompted this paper.

The first problem in this scenario is the consultant who asked the question is not “loading” data into Teradata. The consultant is inserting data into Teradata.

### 7.1.1 Let’s Think, Really Think, About the Problem

The problem of getting data into a database appears to be simple, but it is not. When you insert data, the database has to keep track of it and make certain that it actually ends-up in the table, correctly. This means that there must be logging going on. Logging requires writing to disk and that is one of the most expensive operations that a computer can perform.

Let’s make it worse. Assume that you are writing to a table used for business intelligence. Since ad hoc queries are common the table will likely have indexes defined on many columns. Indexes must be maintained by the system - more writing to disk.

This sounds really bad and you may be thinking, “I should never insert data into my Teradata tables.” Inserting data into a table is fine and sometimes the preferable way to get data into your database. The deciding factor is how long it takes to insert the data into the table. If you need to add 500 rows to a table every night, chances are that inserting will work just fine.

What if you need to add 100,000 rows to a table? Will inserting work in that situation? The answer to that depends upon the Teradata and SAS environments and time window. How much time can you afford? That is for you and the customer to decide. Chances are you will be looking for better performing alternatives – the ideas in this paper are a good place to start.

### 7.1.2 Speaking with the Teradata DBA

We SAS folk tend to use imprecise language when we discuss databases. This can cause huge misunderstandings when dealing with DBAs. Recently, I was on a customer call that included SAS consultants and Teradata DBAs where vague, imprecise, language caused many issues.

The DBAs speak a different language than most SAS consultants. They tend to be very precise in the words that they use. Let's review some vocabulary.

INSERT	A SQL statement used to add data to a database table one row at a time.
LOAD	Using the Teradata FASTLOAD API or Utility to quickly move data into the Teradata Server.
FASTLOAD	An API that is used by SAS/ACCESS software to quickly move data into the Teradata Server.  When a Teradata DBA hears the word "load" this is what they think you mean.
MULTILOAD	A utility that is used by SAS/ACCESS software to quickly move data into the Teradata Server.

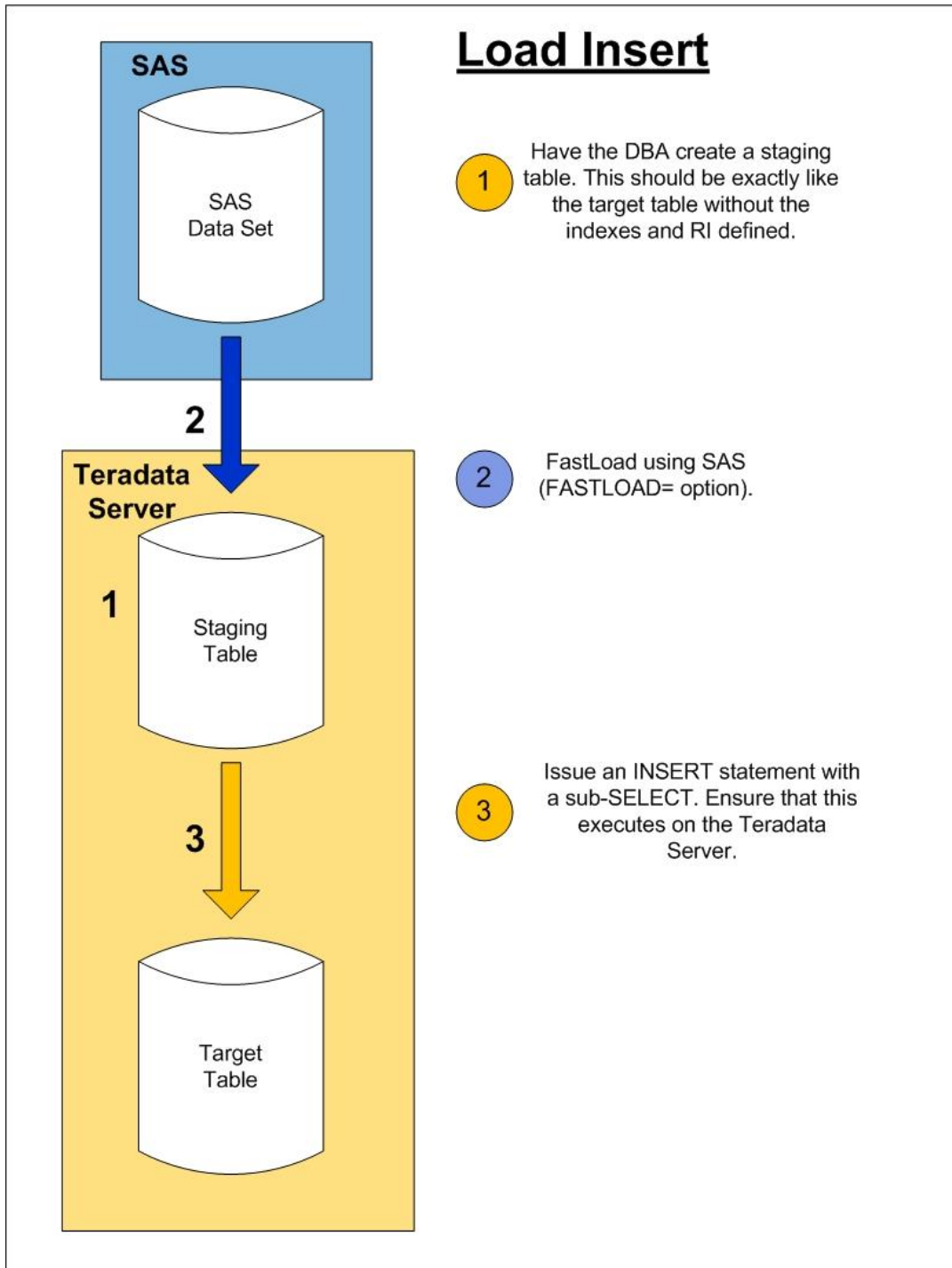
### 7.1.3 SAS AML and Teradata

The SAS Anti-Money Laundering solution requires a data mart. It was this data mart that our SAS consultant was "loading" – really inserting – data into. She was inserting data because the table had data in it (can't use FASTLOAD=) and there was referential integrity (RI) defined on it (can't use MULTILOAD=). MULTILOAD= could have been used if indexes and RI were dropped, but that is a hassle. Plus, it would have changed the way the system was designed to work. As far as she was concerned, she had no choice – inserting was the only option.

After experimenting with the SAS MULTISTMT= and DBCOMMIT= options they were able to get the 100,000 row insert job to run in approximately 45 minutes. Certainly an improvement, but they needed a faster solution.

### 7.1.4 This Doesn't Seem Like it Would be Faster – But it is

The solution to the problem is very interesting. It is commonly used in Teradata environments. Plus, it works great and it is very fast.



Here is sample SAS code that will implement this process.

```
libname mytera teradata server=TDserv user=myuser password=mypasswd;
libname mydata "C:\SASData";

options fullstimer;

/* Verify SQL sent to Terdata */
options sastrace=',,,d' sastraceloc=saslog;

/* FASTLOAD= into the staging table */
proc sql;
    insert into mytera.LoadTest_STAGE (FASTLOAD=YES)
        select * from mydata.AddData;
quit;

/* insert into the target table */
/* MODE=TERADATA is used because it auto commits */
proc sql;
    connect to teradata (server=TDserv
                        user=myuser password=mypasswd
                        MODE=TERADATA);
    execute
        (insert into LoadTest select * from LoadTest_STAGE) by teradata;
quit;
```

Recall the performance statistics from the INSERT statement testing. The fastest time when inserting 100,000 rows was 8 minutes and 57 seconds. This Load Insert technique will move the data into the target table in approximately 22 seconds. It's amazingly fast.

## 8 Bibliography

“SAS/ACCESS 9.2 for Relational Databases: Reference.” PDF. SAS, Cary, NC.

“SAS/ACCESS 9.1.3 for Relational Databases: Reference, Fifth Edition.” PDF. SAS, Cary, NC.

“SAS/ACCESS 9.1.3 Supplement for Teradata: SAS/ACCESS for Relational Databases, Second Edition.” PDF. SAS, Cary, NC.

“Teradata FastLoad Reference.” B035-2411-067A. Teradata Tools and Utilities 12.0. PDF. Teradata Corporation, Miamisburg, OH.

“Teradata MultLoad Reference.” B035-2409-067A. Teradata Tools and Utilities 12.0. PDF. Teradata Corporation, Miamisburg, OH.

“SQL Reference: Data Definition Statements.” B035-1144-067A. Teradata Tools and Utilities 12.0. PDF. Teradata Corporation, Miamisburg, OH.

“SQL Reference: Data Manipulation Statements.” B035-1146-067A. Teradata Tools and Utilities 12.0. PDF. Teradata Corporation, Miamisburg, OH.



## 9 Credits and Acknowledgements

Pravin Boniface, SAS.

Allen Cunningham, SAS.

Teri Huels, SAS.

John Graas, Teradata.

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**

---

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