# SAS® Analytics Accelerator 2.1 for Teradata

## Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS® Analytics Accelerator 2.1 for Teradata: Guide*. Cary, NC: SAS Institute Inc.

**SAS® Analytics Accelerator 2.1 for Teradata: Guide**

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

1st electronic book, September 2011

# Contents

# Chapter 1

# SAS Analytics Accelerator 2.1 for Teradata: Guide

## Contents

## Overview of In-Database Computing

SAS applications are often built to work with large volumes of data in environments that demand rigorous IT security and management. When the data are stored in an external database, such as a Teradata database, the transfer of large data sets to the computers that run the SAS System can cause a performance bottleneck and possible unwanted security and resource management consequences for local data storage. SAS Analytics Accelerator 2.1 for Teradata addresses these challenges by moving computational tasks closer to the data and by improving the integration between the SAS System and the database management system (DBMS).

At present, there are in-database versions of six SAS/STAT procedures, one SAS/ETS procedure, and three SAS Enterprise Miner procedures. The SAS/STAT in-database procedures include CANCORR, FACTOR, PRINCOMP, REG, SCORE, and VARCLUS; the SAS/ETS procedure is TIMESERIES; and the SAS Enterprise Miner procedures are DMDB, DMINE, and DMREG.

There are also five Base SAS in-database procedures (CORR, FREQ, MEANS, RANK, and SUMMARY), but Base SAS in-database procedures, with the exception of the CORR procedure, do not require the installation of SAS Analytics Accelerator 2.1 for Teradata. Table 1.1 lists the software licenses that are required to take advantage of in-database computing.

**Table 1.1** Software Licensing Requirements for In-Database Computing

| **Procedures** | **Licenses Required** |
|---|---|
| SAS/STAT procedures<br>(CANCORR, FACTOR, PRINCOMP,<br>REG, VARCLUS, and SCORE) | Base SAS 9.3<br>SAS/ACCESS 9.3 (SAS/ACCESS Interface to Teradata)<br>SAS/STAT 9.3<br>SAS Analytics Accelerator 2.1 for Teradata |
| SAS/ETS procedure<br>(TIMESERIES) | Base SAS 9.3<br>SAS/ACCESS 9.3 (SAS/ACCESS Interface to Teradata)<br>SAS/ETS 9.3<br>SAS Analytics Accelerator 2.1 for Teradata |
| SAS Enterprise Miner procedures<br>(DMDB, DMINE, and DMREG) | Base SAS 9.3<br>SAS/ACCESS 9.3 (SAS/ACCESS Interface to Teradata)<br>SAS Enterprise Miner 7.1<br>SAS Analytics Accelerator 2.1 for Teradata |
| Base SAS statistical procedures<br>(FREQ, MEANS, RANK, and SUMMARY) | Base SAS 9.3<br>SAS/ACCESS 9.3 (SAS/ACCESS Interface to Teradata) |
| BASE SAS statistical procedures<br>(CORR) | Base SAS 9.3<br>SAS/ACCESS 9.3 (SAS/ACCESS Interface to Teradata)<br>SAS Analytics Accelerator 2.1 for Teradata |

All SAS Analytics Accelerator 2.1 for Teradata in-database functions require Teradata 13.00.00.15 or higher.

Installation instructions for SAS Analytics Accelerator 2.1 for Teradata are published in the section “Configuring SAS Analytics Accelerator for Teradata” in the *Configuration Guide for SAS 9.3 Foundation for UNIX Environments* and the *Configuration Guide for SAS 9.3 Foundation for Microsoft Windows*.

In SAS 9.3, SAS/STAT 9.3, and SAS/ETS 9.3 in-database computing is available only for the Teradata DBMS. Future releases might provide in-database computing for other DBMSs.

## Teradata and the SAS System

In a conventional environment where in-database computing is not possible, a large amount of data is stored in the Teradata database. When a SAS procedure executes, it must read the data through the SAS/ACCESS engine. This movement of data from the database to the SAS workspace server causes a performance penalty. In an environment where in-database computing is possible, such as that provided by SAS Analytics Accelerator 2.1 for Teradata, when a comparable Base SAS, SAS/STAT, SAS/ETS, or SAS Enterprise Miner in-database procedure executes, some of the procedure's computations are performed within Teradata, and only the results of those computations, rather than the raw data, are passed to the SAS workspace server. The benefit is a reduction in data movement between the database and the SAS workspace server. The cost of this reduction in data movement is the difference in processing time due to using Teradata versus the SAS System to process the data. The trade-off in processing time versus data movement time depends on several dynamic factors, including client and server workloads, network speed and workload, and data size. The net benefit from using in-database procedures varies based on the particular environment. In general, the net benefit increases as the number of rows in a database table increases.

## SQL Generation

When performing in-database modeling, a SAS/STAT, SAS/ETS, or SAS Enterprise Miner procedure dynamically generates SQL code, which is based on the procedure options and statements. It then submits the SQL code directly to the database. The code can be standard SQL that can be interpreted by any database, or it can be tuned specifically for Teradata. The choice for the type of SQL code is determined by the complexity of the required analysis. The code can include SAS formats that are executed in Teradata. The query returns result sets that are used by the in-database procedure to complete the analysis before supplying the results to one of the following: the SAS output listing, the ODS listing, ODS Graphics, or output tables. The end result is usually the same as when you use Base SAS tables rather than Teradata tables. However, with in-database computing the relational database software is responsible for optimizing and executing the query. When the table to be analyzed is very large, the in-database computing approach can result in significantly lower total elapsed processing time and in reduced data movement between the SAS System and the Teradata database.

## SAS In-Database Functions

When a SAS/STAT, SAS/ETS, or SAS Enterprise Miner procedure dynamically creates and submits SQL code, this code includes references to new user-defined functions (UDFs), developed by SAS, which are installed on the Teradata system. These UDFs are the key to performing advanced statistical computations efficiently in-database. For example, SAS has developed a Teradata UDF called SAS_TACORR for computing uncorrected sum of squares and crossproducts (SSCP) matrices (also known as **X'X** matrices) and related statistics in-database. The SAS/STAT procedures CANCORR, FACTOR, PRICOMP, REG, and VARCLUS have been modified to take advantage of the SAS_TACORR UDF, which enables them to perform advanced statistical computations on a DBMS table without downloading the entire table with SAS/ACCESS software.

To better understand how the SAS_TACORR UDF enhances computing efficiency, consider the following. The SSCP matrix is a condensed representation of the relationships between variables. The size of the SSCP matrix is determined by the number of variables that are used by the procedure and does not depend on the number of rows. For example, a data set that contains 100 numeric columns and 5 million rows of data produces an SSCP matrix that contains, nominally, 100x100 cells. However, a SAS/STAT in-database procedure needs only the lower triangle of 100x50 cells. Thus, the data that is transferred from Teradata to SAS is reduced from 500,000,000 raw data cells to 5,000 SSCP data cells, which is a reduction of 99.999 percent. As the example illustrates, the benefit of computing the SSCP matrix in-database versus passing the raw data to the SAS System increases greatly as the number of rows increases. Large data applications benefit the most from in-database processing.

The SAS/STAT in-database procedure constructs SQL code that uses the SAS in-database function to create the SSCP matrix. This function is not intended for general-purpose SQL queries, but it is useful for SAS/STAT in-database procedures. The elements of the SSCP are created and transferred to the SAS System. After the crossproducts have been transferred to the SAS System, the database is available to continue processing queries while the SAS/STAT procedure continues the analysis.

# SAS In-Database Procedures

Several SAS/STAT, SAS/ETS, SAS Enterprise Miner, and Base SAS procedures have been modified to move critical data-intensive operations into the database. Such operations include basic summarization and exploratory data analysis. These SAS procedures are commonly found in many SAS programs and represent a large opportunity to improve efficiency when working with relational databases. Current changes are optimized for the Teradata relational database. The modifications to the SAS/STAT, SAS/ETS, and SAS Enterprise Miner in-database procedures, as well as the Base SAS CORR procedure, require the installation of the SAS Analytics Accelerator 2.1 for Teradata and are documented here.

## Base SAS In-Database Procedures

Base SAS currently offers in-database implementations of the CORR, FREQ, MEANS, RANK, and SUMMARY procedures. The Base SAS in-database procedures work differently from the SAS/STAT, SAS/ETS, and SAS Enterprise Miner in-database procedures and do not require the installation of the SAS Analytics Accelerator for Teradata (with the exception of the CORR procedure). The Base SAS in-database procedures are included in the Base SAS software and are documented in *Base SAS Procedures Guide*. However, some notes about the options that affect the Base SAS in-database procedures are included in this document for convenience.

## SAS/STAT In-Database Procedures

In-database versions of the CANCORR, FACTOR, PRINCOMP, REG, SCORE, and VARCLUS procedures are available in SAS/STAT 9.3. Each procedure generates SQL code which, except for the SCORE procedure, calls the SAS in-database function for SSCP creation. These procedures are commonly used in exploratory data analysis and regression model building. The procedure syntax and output for in-database computing is identical to the syntax and output for conventional SAS processing in most cases. In those cases, existing procedure steps in SAS programs are able to run in-database without modification.

However, some issues with in-database processing can cause the output to differ because of numerical precision issues and BY processing issues. These issues are discussed in the section "Conditions That Prevent In-Database Processing." Together with the in-database Base SAS procedures, the SAS/STAT procedures can be used for basic tasks that are often executed at the beginning of larger analytical tasks. Use of these techniques can greatly reduce the total data transfer between relational databases and the SAS System.

### The CANCORR Procedure

The CANCORR procedure performs canonical correlation, partial canonical correlation, and canonical redundancy analysis.

Canonical correlation is a generalization of multiple correlation for analyzing the relationship between two sets of variables. In multiple correlation, you examine the relationship between a linear combination of a set of explanatory variables, **X**, and a single response variable, **Y**. In canonical correlation, you examine the relationship between linear combinations of the set of **X** variables and linear combinations of a set of **Y** variables. These linear combinations are called canonical variables or canonical variates. Either set of variables can be considered explanatory or response variables, since the statistical model is symmetric in the two sets of variables. Simple and multiple correlation are special cases of canonical correlation in which one or both sets contain a single variable.

The SSCP matrix is computed in-database and is transferred to the SAS System for further processing. The options in the CANCORR procedure that are affected by in-database processing are detailed in the section "CANCORR Procedure Options Affected by In-Database Computing."

### The FACTOR Procedure

The FACTOR procedure performs a variety of common factor and component analyses and rotations. The methods for factor extraction are principal component analysis, principal factor analysis, iterated principal factor analysis, unweighted least squares factor analysis, maximum likelihood (canonical) factor analysis, alpha factor analysis, image component analysis, and Harris component analysis. A variety of methods for prior communality estimation are also available.

The SSCP matrix is computed in-database and is transferred to the SAS System for further processing. The options in the FACTOR procedure that are affected by in-database processing are detailed in the section "FACTOR Procedure Options Affected by In-Database Computing."

## The PRINCOMP Procedure

The PRINCOMP procedure computes a principal component transformation of the SSCP matrix into orthogonal components. The first component accounts for the maximum amount of variation; the second accounts for the next largest amount of variation; and so on. This procedure is typically used in exploratory data analysis and visualization. Scatter plots that are generated from the principal component dimensions often reveal interesting relationships among the data points.

Principal component analysis (PCA) reduces the dimensionality of data for predictive modeling by replacing the variables in the original data with fewer principal component terms in the final model. The SSCP matrix is computed in-database and is transferred to the SAS System for further processing. The options in the PRINCOMP procedure that are affected by in-database processing are detailed in the section "PRINCOMP Procedure Options Affected by In-Database Computing."

## The REG Procedure

The REG procedure computes a model in which a dependent variable is modeled as a linear equation of multiple independent variables by a least squares function. PROC REG has numerous model-fitting options, which include many model selection variations and hypothesis tests. Regression analysis is often used to identify a subset of independent variables that have unique and significant relationships with the dependent variable. In this case, PROC REG is used for its exploratory analysis, in addition to its extensive model fitting and reporting functions. You might want to use the REG procedure to compute candidate models in the database. A full SSCP matrix is computed in Teradata and is transferred to the SAS System, in which the model fitting occurs.

PROC REG includes an option that uses a CORR or SSCP matrix as input and creates a CORR or SSCP matrix as output. For both in-database processing and matrix input, any option that requires access to individual rows of data is disabled. The disabled options include the general class of residual analysis, plot options, and confidence intervals. Complete details of which options and statements are affected by in-database processing are provided in the section "REG Procedure Options Affected by In-Database Computing."

## The SCORE Procedure

Many statistical procedures create output data sets (using the OUTEST= or OUTSTAT= options) that contain coefficients. The SCORE procedure can apply the coefficients in those data sets to a raw data set and compute new variables that are generically called scores. Each new score variable is formed as a linear combination of the raw data and the scoring coefficients. That is, for each observation in the raw data set, PROC SCORE multiplies the value of a variable in the raw data set by the matching scoring coefficient from the data set of scoring coefficients. This multiplication process is repeated for each variable in the VAR statement. The resulting products are then summed to produce the value of the new score variable. In other words, PROC SCORE performs a matrix multiplication on the two data sets.

For example, you can use the SCORE procedure to produce output in the form of a table in Teradata that contains predicted values or residuals from a model that is estimated using the REG procedure in-database. The SCORE procedure dynamically generates SQL code for the given model. PROC SCORE then submits the SQL code to the database, which produces a Teradata table without having to extract any rows of data into the SAS System. Details of which options in the SCORE procedure are affected by in-database processing are provided in the section "SCORE Procedure Options Affected by In-Database Computing."

PROC SCORE cannot be used for scoring nonlinear models.

The following SAS procedures produce output data sets that contain scoring coefficients that can be used by PROC SCORE:

- ACECLUS procedure
- CALIS procedure
- CANCORR procedure
- CANDISC procedure
- DISCRIM procedure
- FACTOR procedure
- PRINCOMP procedure
- TCALIS procedure
- VARCLUS procedure
- ORTHOREG procedure
- QUANTREG procedure
- REG procedure
- ROBUSTREG procedure

You can also use a DATA step or the IML procedure to output any coefficients you want in a data set to use with PROC SCORE.

## The VARCLUS Procedure

The VARCLUS procedure groups variables into clusters that are based on their correlations. The full SSCP matrix is created, and then clusters are chosen to maximize the variance that is associated with the first principal component within the cluster. An iterative process assigns variables to clusters to maximize the sum across clusters of the variance of the original variables, which is explained by the centroid cluster measure.

Similar to principle components, variable clustering is another technique that is used for exploratory data analysis and for variable reduction, which reduces the number of terms that are used in successive analyses. The SSCP matrix is computed in-database and is then transferred to the SAS System for further processing.

Details of which options in the VARCLUS procedure are affected by in-database processing are provided in the section "VARCLUS Procedure Options Affected by In-Database Computing."

## SAS/ETS In-Database Procedures

The TIMESERIES procedure analyzes time-stamped transactional data with respect to time and accumulates the data into a time series format. The procedure can perform trend and seasonal analysis on the transactions. After the transactional data are accumulated, time domain analysis and frequency domain analysis can be performed on the accumulated time series.

Details of which options in the TIMESERIES procedure are affected by in-database processing are provided in the section "TIMESERIES Procedure Options Affected by In-Database Computing."

## SAS Enterprise Miner In-Database Procedures

### The DMDB Procedure

SAS Enterprise Miner architecture is based on the creation of a data mining database (DMDB) that is a snapshot of the original data. PROC DMDB creates this DMDB from the input data source. It also compiles and computes metadata information about the input data based on variable roles and stores it in a metadata catalog. The DMDB and the associated metadata catalog facilitate subsequent data mining activities. They are both designed for processing and storage efficiencies.

### The DMINE Procedure

Many data mining databases have hundreds of potential model inputs. The DMINE procedure enables you to quickly identify the input variables that are useful for predicting the target variables based on a linear models framework. The procedure facilitates ordinary least squares or logistic regression methods.

### The DMREG Procedure

The DMREG procedure enables you to fit both linear and logistic regression models. Linear regression attempts to predict the value of a continuous target as a linear function of one or more independent inputs. Logistic regression attempts to predict the probability that a categorical (binary, ordinal, or nominal) target will acquire the event of interest as a function of one or more independent inputs. The procedure supports forward, backward, and stepwise selection methods. It also enables you to score data sets or generate SAS DATA step code to score a data set.

# SAS In-Database Macros

## The %EM_INDBIN Macro

The macro %EM_INDBIN performs quantile-based binning of all interval variables in an in-database table. The macro also generates a file that contains the SQL code that is used for binning, a SAS data set that contains mapping information for the variables that are binned, and the weight-of-evidence statistic for each bin. Installation of both SAS Analytics Accelerator 2.1 or later and SAS® Enterprise Miner 7.1 or later is required.

The %EM_INDBIN macro permits the following options:

**ADJUSTFACTOR=***constant*

specifies the *constant* that is used to adjust the weight of evidence if either the bin event counts or non-event counts are zeros. See the OPTIONS=WOEADJUST option for details. The default is 0.5.

**BINMAPPINGS=***mapping_table*

specifies a table that contains the mapping information for all binned variables. If the BINMAPPINGS= option is not specified, then the data set Work.b is created.

**BINNINGCODE=***fileref*

specifies a *fileref* location where binning code is generated.

**DATA=***libref.data_table*

specifies the in-database data table to be binned. You must associate a SAS libref directly with the table, using the SAS/ACCESS LIBNAME statement to assign a libref to DBMS data.

**EVENTVAL=***event_value*

defines the event value for the target variable if the OPTIONS=WOEADJUST option is also specified. If the target is a character variable, you must quote your string. By default (if you do not specify a value), the level is based on the variable ordering that is specified in the LEVELORDER= option.

**LEVELORDER=ASCENDING | ASC | DESCENDING | DES | RAREST**

specifies the level ordering when the EVENTVAL= option is not specified. The level of interest relates to the column (variable) with a ROLE=TARGET in the metadata table that is specified in the VARTABLE= option. If you specify LEVELORDER=ASCENDING or LEVELORDER=ASC, the levels of the target variable are sorted in ascending order and the level of interest is the first level. If you specify LEVELORDER=DESCENDING or LEVELORDER=DES, the levels of the target variable are sorted in descending order and the level of interest is the first level. If you specify LEVELORDER=RAREST, the levels of the target variable are sorted by their total counts and the level of interest is the one with the lowest total count. The default is DESCENDING.

**NUMBINS=***number*

specifies the maximum *number* of bins to create for each input variable. The *number* must be less than or equal to 500; the default is 5.

**OPTIONS=WOEADJUST**

creates the WTEV (weight of evidence), EVENTCNT (event counts), and BINCNT (total counts) in the table specified in the BINMAPPING= option. You must specify a table in the OUT= option when OPTIONS=WOEADJUST is specified.

**OUT=***libref.output_table*

specifies an output table that contains the original variables plus the binned variables. The libref must be the same as the libref that you specify in the DATA= option.

**PRECISION=***number*

specifies the precision (number of decimal places) for cutoffs. *number* must be an integer; the default value is 3.

**TARGETVAR=***target_variable*

specifies the target variable. If the table specified by the VARTABLE= option contains a record with ROLE=INPUT and LEVEL=TARGET, the TARGETVAR= option redefines the target variable. If *target_variable* contains characters that are not supported by SAS software by using VALIDVARNAME=ANY, use the %NRSTR() macro to quote your variable. For example, use TARGETVAR= %NRSTR(Purchase Yes/No) for a column name that contains embedded blanks and a slash.

**VARTABLE=***metadata_table*

specifies a SAS table that contains metadata for the columns. This table must contain the following columns: NAME, ROLE, and LEVEL. If this option is not specified, a column metadata table is created using the %EMDADVISECOLUMNS macro, which is stored in the SAS catalog *sashelp.emmeta.emadvisecolumns.source*.

See the section "Example: Using the %EM_INDBIN Macro" on page 74 for a demonstration of how to invoke the %EM_INDBIN macro.

## The %EMIDBSAM Macro

The macro %EMIDBSAM performs in-database sampling for a data table that resides in a relational database. The sample rows are output to a SAS data set. The sampling methods include simple random sampling, top-n sampling, and stratified sampling. You can specify either a number of rows or a percentage of the population to select for the sample. If you are working with rare events, the macro can also perform oversampling. When you adjust the frequency for oversampling, the macro creates (or adjusts) a frequency variable that contains the sampling weights. The macro is available in SAS Enterprise Miner 7.1 and later.

The %EMIDBSAM macro permits the following options:

**CRITERION=EQUAL | LEVELBASED | PROPORTIONAL**

specifies the sampling criterion during stratified sampling. The values of the CRITERION= option are interpreted as follows:

**EQUAL** specifies that the number of rows in each stratum be the same. The total sample size is divided by the number of strata to determine the number of rows to sample from each stratum.

**LEVELBASED** specifies that the sample be based on the proportion captured and sample proportion of a specific level. Use the LEVELPROP= option to specify the parameter for the proportion of the selected level to be captured, the SAMPPROP= option to specify the parameter for the sample proportion, and the LEVEL= option to select the level of interest.

**NOTE:** If more than one stratification variable is used, CRITERION=LEVELBASED is replaced by CRITERION=PROPORTIONAL.

**PROPORTIONAL** specifies that the proportion of rows in each stratum be the same in the sample as it is in the population. This is the default.

**DATA=***libref.data_table*

specifies the in-database data table to be sampled. You must associate a SAS libref directly with the table, using the SAS/ACCESS LIBNAME statement to assign a libref to DBMS data.

**FREQVAR=***column*

specifies a *column* that contains a frequency variable with sampling weights. The sampling weights are used to adjust the frequency for oversampling. If this option is not specified but the ADJFREQ option is specified, the variable _FREQ_ is created. When the VARSTABLE= option is specified, this option is ignored.

**LEVEL=EVENT | RAREST | RARESTLEVEL**

specifies the level of interest. The level of interest relates to the column (variable) with a ROLE=TARGET in the metadata table that is specified in the VARSTABLE= option. If you specify LEVEL=EVENT, then the level is based on the variable ordering that is specified in the LEVELORDER= option. If you specify LEVEL=RAREST or LEVEL=RARESTLEVEL, then the level of interest is the level of the target variable that has the lowest frequency count.

**LEVELORDER=ASCENDING | DESCENDING**

specifies the level ordering of the target variable when LEVEL=EVENT. If you specify LEVELORDER=ASCENDING, the levels of the target variable are sorted in ascending order and the level of interest is the first level. If you specify LEVELORDER=DESCENDING, the levels of the target variable are sorted in descending order and the level of interest is the first level.

**LEVELPROP=***proportion*

specifies the proportion of the selected level of interest that is to be included in the sample.

**METHOD=DEFAULT | RANDOM | STRATIFY | TOP**

specifies the sampling method. The values of the METHOD= option are interpreted as follows:

**DEFAULT** specifies that if the STRATVARS= option is null, the default sampling method is RANDOM; otherwise, the default sampling method is STRATIFY.

**RANDOM** specifies that each row in the data table (population) have the same probability of being selected for the sample, independently of the other rows that happen to fall into the sample. For example, if row 145 is randomly drawn as the first member of the sample, you know that each member of the data table still has an equal chance of being the second member of the sample.

**STRATIFY** specifies stratified sampling; nominal, binary, or ordinal variables are chosen from the input data table to form strata (or subsets) of the total population. Within each stratum, all rows have an equal probability of being selected for the sample.

You perform stratified sampling to preserve the strata proportions of the population within the sample. This can preserve the classification precision of the fitted models.

**TOP** specifies that the top *n* (computed from either the NROWS= option or the PERCENTAGE= option) rows be selected from the input data table for the sample.

**MINSTRATASZE=***n*

specifies the minimum number of rows that are required to construct a valid stratum. *n* must be an integer that is greater than or equal to 1; the default is 5.

**NROWS=***number*

specifies the *number* of rows to be included in the sample. *number* must be a nonnegative integer. Do not specify both the NROWS= option and the PERCENTAGE= option.

**OPTIONS=***option <option option . . . >*

specifies a list of additional *options* separated by blanks. The available *options* and their interpretations are as follows:

**ADJFREQ** specifies that the frequency for oversampling be adjusted and that a biased stratified sample be created. The biased stratified sample uses a frequency variable (specified by the FREQVAR= option) that contains the sampling weights to adjust the target level proportions. If the FREQVAR= option is not specified, the variable _FREQ_ is created. Oversampling is performed only when this option is specified. The extent of the adjustment depends on how biased your sample is with respect to the input data source.

If the BASEDONCOUNT option is also specified, then the adjusted frequency of stratum *i* is computed as

$$AdjFreq_i = freq * \left(\frac{N_i}{n_i}\right)$$

where $N_i$ is the population size of stratum $i$, $n_i$ is the sample size of stratum $i$, *freq* is the value of the frequency variable, *freq*=1 if the frequency variable is not specified, and *freq*=0 if the EXCLMISSLVLS option is also specified and the stratification variables for stratum $i$ contain a missing value. Missing values in a database table are defined as either a blank or a NULL value for character variables, and are defined as a NULL value for numeric variables.

If the BASEDONCOUNT option is not specified, then the adjusted frequency of stratum *i* is computed as

$$AdjFreq_i = freq * \left(\frac{N_i}{n_i}\right) * \left(\frac{n}{N}\right)$$

where $N$ is the total population size and $n$ is the total sample size.

**BASEDONCOUNT** specifies that the frequency variable adjustment be based on counts.

**EXCLMISSLVLS** specifies that strata where the stratification variables contain missing values be excluded. The frequency variable for excluded strata is set to 0.

**IGNORESMALL** specifies that when stratified sampling is applied, any stratum that has a population less than the value specified in the MINSTRATASZE= option be excluded from the sample.

**PROFILE** specifies that the input data table be profiled (that is, that the frequency count of each stratum, the total frequency count, and the frequency ratio be reported in the input table).

**RANDOMIZED** specifies that a true random sampling of your table be taken. If this option is not specified, the default sampling method produces a sample with sufficient randomness instead of a true simple random sample. Specifying this option causes the macro to take more time compared to the default method, especially for very large tables. This option is applicable to Teradata only.

**REPORT** specifies that the frequency count of each stratum, the total frequency count, and the frequency ratio be reported in the sample table.

**OUT=**< *libref.* >*output_table*

specifies the sampling output table. If the libref is not specified, the default is Work. If the *output_table* is not specified, the default is Work.a.

**PERCENTAGE=***percentage*

specifies the proportion of the input table rows to be included in the sample. *percentage* must be a real number greater than 0; the default is 10.0. Do not specify both the NROWS= option and the PERCENTAGE= option.

**SAMPPROP=***proportion*

specifies the proportion of the sample that is to contain the selected level of interest.

**STRATVARS=***column <column column . . . >*

specifies a list of columns (variables) separated by blanks to be used for stratification. The combinations of levels of *variables* define the strata in the sample. If this option is not specified, no stratification occurs. When the VARSTABLE= option is specified, this option is ignored.

**VARS=***column <column column . . . >*

specifies a list of columns (variables) separated by blanks to be included in the sample. If this option is not specified, then all of the columns are selected. The column specified by the FREQVAR= option is automatically included in the sample table, even if it is not specified in the VARS= option. When the VARSTABLE= option is specified, this option is ignored.

**VARSTABLE=***metadata_table*

specifies a SAS table that contains metadata for the columns. This table must contain the following columns: NAME, ROLE, and LEVEL. When this option is specified, the VARS= ,
STRATVARS=, and FREQVAR= options are ignored. When this option is not specified, a column metadata table is created using the %EMDADVISECOLUMNS macro which is stored in the SAS catalog *sashelp.emmeta.emadvisecolumns.source.*

## Limitations of In-Database Procedures

Some features of SAS procedures do not execute in-database. The ability to perform in-database processing depends on the specific type of analysis that is provided by the statements and options of a procedure. These limitations are considered reasonable for accessing and exploring data. After a subset of data has been selected and a model form has been established, a data sample can be defined by using PROC SQL. This subset of data can then be accessed directly by the SAS procedure to execute an analysis that includes functions that do not execute in-database. Use of the SAS in-database procedures reduces the amount of data that is transferred and makes use of the resources of the Teradata system.

# Performance and Numerical Accuracy Issues with In-Database Computing

Consider the following issues when performing in-database computations:

- In-database computation can be either faster or slower than computation that uses SAS/ACCESS software, depending on client performance, network performance, server performance, and the type of computations involved.
- In-database computation tends to be slow when the number of variables in the analysis is large or when the number of BY groups is large.
- Additional overhead is involved with in-database computation, and therefore more memory is required on the client for in-database computation compared to out-of-database computation.
- The SSCP matrix is computed with less accuracy in-database when the variables have small coefficients of variation.

# Enabling and Controlling In-Database Computing

You control in-database computing with options in the LIBNAME statement and the OPTIONS statement.

**SQLGENERATION=NONE | DBMS | ALL**

specifies the type of in-database computing to be performed. The SQLGENERATION= option can be specified as either a LIBNAME statement option or as a system option in an OPTIONS statement. The value used for the SQLGENERATION= option depends on whether the option is specified in the LIBNAME statement, an OPTIONS statement, or both as follows:

- If the SQLGENERATION= option is specified in the LIBNAME statement for a data set, then with respect to that data set, any value of the SQLGENERATION= option that is specified in any OPTIONS statement is ignored.

- If the SQLGENERATION= option is not specified in a LIBNAME statement for a data set, then with respect to that data set, the value of the SQLGENERATION= option that is specified in the most recent OPTIONS statement is used.
- If the SQLGENERATION= option is not specified in either the LIBNAME statement for a data set or in an OPTIONS statement, then by default with respect to that data set, SQLGENERATION=DBMS.

You can specify different values of the SQLGENERATION= option for the DATA= and OUT= data sets by using different LIBNAME statements for the two data sets.

The values of the SQLGENERATION option are interpreted as follows:

**NONE** specifies that no in-database computation be performed. In this case, a SAS/STAT procedure uses SAS/ACCESS to copy the raw data from Teradata to the SAS Work directory, and the procedure performs all computations within the SAS System.

**DBMS** specifies that in-database computation be performed for data sets that are DBMS tables if no system options, LIBNAME options, data set attributes, or procedure options are incompatible with in-database computation. DBMS is the default value, and it automatically enables in-database processing.

When the value is DBMS, the procedure uses in-database processing (when possible) and uses conventional SAS processing when the specific procedure statements and options do not support in-database processing. For example, a REG procedure statement that contains a residual option computes the SSCP matrix in-database, but transfers the detail data to the SAS System to create residual analysis and output.

If the procedure attempts to perform a computation in-database but the SQL command fails, the procedure tries to perform that computation out-of-database.

**ALL** specifies that in-database computation be performed whenever possible.

In-database computation is not applicable to data sets that contain summary statistics, such as OUTEST= and OUTSTAT= data sets, or DATA= data sets for which the TYPE= attribute is CORR, EST, SSCP, FACTOR, and so on. (See the “Special SAS Data Sets” chapter in *SAS/STAT User’s Guide* for the complete list of TYPE= attributes.) Because the data in these tables are already summaries, no additional summarization can be performed in-database. The SQLGENERATION= option is ignored with respect to those data sets for which in-database computation is not applicable.

The following system options, which are specified in an OPTIONS statement, also control in-database computing for SAS/STAT, SAS/ETS, and SAS Enterprise Miner procedures:

**MSGLEVEL=N | I**

specifies the level of messages to be printed to the log. N is the default value.

**N** specifies that the following messages be printed in the log:

- a confirmatory note that states that SQL is used for in-database computations, when that is the case
- error messages if anything goes wrong with the SQL commands submitted for in-database computations

- note that states whether SQL is used, when there are SQL error messages

**I** specifies that everything printed by MSGLEVEL=N be printed plus the following notes:

- note that explains why SQL is not used for in-database computations, if that is the case (however, no note is printed when OPTIONS SQLGENERATION=NONE is specified)
- note that explains that the TYPE= attribute is not stored in DBMS tables when you try to create a special SAS data set (that is, a data set that has a non-blank value of the TYPE= attribute) as a DBMS table

**SQL_IP_TRACE=NOTE | SOURCE | ALL**

specifies the level of information to be included in the trace output. You can specify more than one value by enclosing the values in parentheses.

**NOTE** specifies that the trace output include the following information:

- all original messages from the DBMS (ordinarily, the SAS System intercepts certain confusing messages from the DBMS and replaces them with messages that can be more easily understood by SAS customers)
- note that indicates whether SQL is used for in-database computations, even when SQLGENERATION=NONE is specified. By default, a note is printed when SQL is used, but no note is printed when SQL is not used. If you specify SQL_IP_TRACE=NOTE, a note is always printed.
- notes that state the name of the data set engine, the name of the DBMS database, and so on
- notes that explain what kinds of SAS passwords apply to the data set, because the drivers used for in-database computation do not support SAS passwords
- note that indicates that no system options or data set attributes prevent using SQL for in-database computation, when that is the case (this note does not rule out the possibility that some procedure options might prevent in-database computation)

**SOURCE** specifies that trace output include the major SQL commands that are submitted for in-database computation, but not minor SQL commands such as those used to obtain metadata.

**ALL** specifies that trace output include the output for NOTE and SOURCE, plus all SQL commands that are submitted (including those used to obtain metadata and to verify that UDFs are published), plus additional details.

**SQLMAPPUTTO=NONE | SAS_PUT**

is not applicable to the SAS/STAT, SAS/ETS, or SAS Enterprise Miner procedures but is included here for completeness.

The Base SAS procedures use implicit pass-through to send SQL commands to Teradata. With implicit pass-through, the SQL commands are written in the SAS dialect of SQL and then the SAS System translates those commands to the Teradata dialect of SQL. The SAS dialect of SQL uses the PUT function to format variables. The Teradata dialect of SQL uses the SAS_PUT UDF to format variables. The SQLMAPPUTTO=SAS_PUT option specifies that the PUT function in SAS SQL be translated (mapped) to the SAS_PUT function in Teradata SQL.

The SAS/STAT, SAS/ETS, and SAS Enterprise Miner procedures write SQL commands directly in the Teradata dialect of SQL instead of using implicit pass-through. Therefore, this option has no effect on the SAS/STAT, SAS/ETS, and SAS Enterprise Miner procedures. See the section "SQLMAPPUTTO= System Option" in *SAS/ACCESS for Relational Databases: Reference*.

**FMTERR | NOFMTERR**

specifies whether to refrain from using unpublished formats.

If a procedure needs to format a BY variable but that format has not been published (that is, not installed on the Teradata server in SYSLIB or the current database), the procedure does not perform in-database computation unless you specify NOFMTERR. NOFMTERR tells the procedure to refrain from using unpublished formats and to use the unformatted values of the BY variables instead. If you specify NOFMTERR and a Teradata error occurs when the procedure tests a format, the procedure uses the unformatted values of the BY variables to define BY groups.

For each BY variable that has a format that cannot be used, a message is printed to that effect. That message is an error if FMTERR is specified or a note if NOFMTERR is specified. In addition, a note states whether SQL is used.

For information about publishing formats, see the section "Deploying and Using SAS Formats in Teradata" on page 43 in this document or the section "Deploying and Using SAS Formats in Teradata" in *SAS/ACCESS 9.3 for Relational Databases: Reference*.

**DBFMTIGNORE | NODBFMTIGNORE**

specifies which type of formats be used for DBMS columns.

Ordinarily, SAS formats control how data values are displayed rather than how data values are stored. However, SAS/ACCESS software works differently from other SAS products with respect to formats.

NODBFMTIGNORE (the default) causes SAS/ACCESS software, when it creates a DBMS table, to use the SAS formats assigned to SAS variables to decide which DBMS data types to assign to the corresponding DBMS columns. For example, the following SAS statements create a DBMS table named pi_table which contains a column named pi with a DBMS data type of decimal (3,2) with a total of 3 digits with 2 decimal places:

```
data dbms.pi_table;
   pi = 22/7;
   format pi 3.2;
run;
```

The value that SAS/ACCESS software inserts in the column pi is 3.14 rather than 3.142857142 … (that is, the actual numeric value of the column is truncated to the two decimal places that are specified in the SAS format). Computations performed with the DBMS column pi suffer from loss of precision and possibly numeric overflow.

If you specify OPTIONS DBFMTIGNORE, SAS/ACCESS software creates numeric DBMS columns with the data type DOUBLE PRECISION. This data type is the same as the default data type that the SAS System uses for numeric variables in ordinary SAS data sets.

The DBFMTIGNORE option does not apply to datetime or character formats.

See the section "Using SAS/ACCESS Software with Teradata: Some Cautionary Notes" for more details.

## LIBNAME Statement Options for In-Database Computing

To perform in-database computations, the SAS/STAT, SAS/ETS, and SAS Enterprise Miner procedures connect to the DBMS by using the credentials (username, password, and so on) in the LIBNAME statement for the DATA= data set. This connection is separate from any connection made by SAS/ACCESS software. Therefore, some SAS/ACCESS options such as the CONNECTION= option do not work with the SAS/STAT, SAS/ETS, and SAS Enterprise Miner in-database procedures. Also, LIBNAME concatenation cannot be used for in-database computation because the procedure cannot determine which credentials to use for the connection.

Although the DATABASE= and SCHEMA= options are aliases as data set options, they are not aliases in a LIBNAME statement. Table 1.2 describes the effects of the DATABASE= and SCHEMA= options in a LIBNAME statement.

**Table 1.2** DATABASE= and SCHEMA= Options for Teradata

| **Options specified in LIBNAME statement** | **Option that specifies name of database where the SAS System searches for DATA= table*** | **Option that specifies name of database where OUT= table is stored**** | **Option that specifies name of database where Teradata looks for UDFs***** |
|---|---|---|---|
| Neither DATABASE= nor SCHEMA= | USER= | USER= | USER= |
| DATABASE= but not SCHEMA= | DATABASE= | DATABASE= | DATABASE= |
| SCHEMA= but not DATABASE= | SCHEMA= | SCHEMA= | USER= |
| Both DATABASE= and SCHEMA= | SCHEMA= | SCHEMA= | DATABASE= |
| | * The DBMS user for the DATA= table must have the SELECT privilege for the named database. | ** The DBMS user for the DATA= table must have the CREATE TABLE, SELECT, and INSERT privileges for the named database. | *** If Teradata does not find the UDF in the named database, then Teradata looks in SYSLIB. The DBMS user for the DATA= table must have the EXECUTE FUNCTION privilege for the database where the UDF is found. |

Table 1.3 describes whether SAS/ACCESS LIBNAME options for Teradata work correctly with the DATA= and OUT= options for the SAS/STAT and SAS/ETS in-database procedures. The cell entries in the DATA= and OUT= columns have the following values:

- Yes—indicates that the option works for in-database computing
- No—indicates that the option does not work for in-database computing
- N/A (not applicable)—indicates that the option does not apply to in-database computing

Some cell entries in the DATA= and OUT= columns have superscripts that are intended to convey additional information. The meanings of the superscripts are as follows:

- A superscript of 1 (Yes$^1$ or No$^1$) indicates an option that is recommended for statistical applications and correctly affects in-database processing as expected.
- A superscript of 2 (Yes$^2$ or No$^2$) indicates an option that correctly affects in-database processing as expected.
- A superscript of 3 (Yes$^3$ or No$^3$) indicates an option that causes the procedure to use SAS/ACCESS for out-of-database computing.
- A superscript of 4 (Yes$^4$ or No$^4$) indicates an option that neither works for in-database computing nor prevents in-database computing, but is unlikely to cause a serious error. See the Comment column for more information.

**Table 1.3** SAS/ACCESS LIBNAME Options for Teradata

| LIBNAME option | DATA= | OUT= | Comments |
|---|---|---|---|
| ACCESS=READONLY | Yes$^2$ | Yes$^2$ | For an OUT= data set, you correctly receive a SAS/ACCESS error message. |
| BULKLOAD= | N/A | N/A | For out-of-database I/O. |
| CAST= | N/A | N/A | For out-of-database I/O. |
| CAST_OVERHEAD_MAXPERCENT= | N/A | N/A | For out-of-database I/O. |
| CONNECTION= | No$^4$ | No$^4$ | In-database SAS/STAT and SAS/ETS procedures use separate connections from SAS/ACCESS software and therefore work as if CONNECTION=UNIQUE is specified. |
| CONNECTION_GROUP= | No$^3$ | No$^3$ | In-database SAS/STAT and SAS/ETS procedures use separate connections from SAS/ACCESS software. |
| DBCOMMIT= | N/A | N/A | For out-of-database I/O. |
| DBCONINIT= | No$^3$ | No$^3$ | In-database SAS/STAT and SAS/ETS procedures use separate connections from SAS/ACCESS software. |

| LIBNAME option | DATA= | OUT= | Comments |
|---|---|---|---|
| DBCONTERM= | No[3] | No[3] | In-database SAS/STAT and SAS/ETS procedures use separate connections from SAS/ACCESS software. |
| DBCREATE_TABLE_OPTS= | N/A | Yes[2] | |
| DBINDEX= | N/A | N/A | For out-of-database I/O. |
| DBLIBINIT= | N/A | N/A | Executed when LIBNAME connection is made. |
| DBLIBTERM= | N/A | N/A | Executed when LIBNAME is disconnected. |
| DBMSTEMP=YES | No[3] | No[3] | In-database SAS/STAT and SAS/ETS procedures use separate connections from SAS/ACCESS software. Temporary tables can be accessed only within the connection in which they are created, so DBMSTEMP=YES causes Teradata errors that say the table does not exist. The default value NO works. |
| DBPROMPT= | N/A | N/A | Executed when LIBNAME connection is made. |
| DBSASLABEL= | Yes[1] | N/A | DBSASLABEL=NONE is recommended for correct procedure output. |
| DBSLICEPARM= | N/A | N/A | For out-of-database I/O. |
| DEFER=NO | No[4] | No[4] | In-database SAS/STAT and SAS/ETS procedures use separate connections from SAS/ACCESS software and therefore work as if DEFER=YES is specified. |
| DIRECT_EXE=DELETE | N/A | N/A | |

| LIBNAME option | DATA= | OUT= | Comments |
|---|---|---|---|
| DIRECT_SQL= | N/A | N/A | PROC SQL only. |
| ERRLIMIT= | N/A | N/A | For out-of-database I/O. |
| LOGDB= | N/A | N/A | For out-of-database I/O. |
| MODE= | N/A | N/A | Applies to explicit pass-through in PROC SQL. |
| MULTISTMT= | N/A | N/A | For out-of-database I/O. |
| MULTI_DATASRC_OPT= | N/A | N/A | For out-of-database I/O with PROC SQL. |
| PASSWORD= | Yes[2] | Yes[2] | |
| PREFETCH= | N/A | N/A | For out-of-database I/O. |
| PRESERVE_COL_NAMES=NO | No[3] | No[3] | The default value YES works. |
| PRESERVE_NAMES=NO | No[3] | No[3] | The default value YES works. |
| PRESERVE_TAB_NAMES=NO | No[3] | No[3] | The default value YES works. |
| READ_ISOLATION_LEVEL= | No[4] | No[4] | The option is ignored. |
| READ_LOCK_TYPE= | No[4] | No[4] | The option is ignored. |
| READ_MODE_WAIT= | No[4] | No[4] | The option is ignored. |
| REREAD_EXPOSURE= | N/A | N/A | |
| SERVER= | Yes[2] | Yes[2] | Same as TDPID=. |
| SPOOL= | N/A | N/A | For out-of-database I/O. |
| SQL_FUNCTIONS= | N/A | N/A | For PROC SQL. |
| SQL_FUNCTIONS_COPY= | N/A | N/A | For PROC SQL. |
| SQLGENERATION= | Yes[2] | Yes[2] | If SQLGENERATION= is specified in the LIBNAME statement, any value specified in an OPTIONS statement is ignored. |

| LIBNAME option | DATA= | OUT= | Comments |
|---|---|---|---|
| TDPID= | Yes[2] | Yes[2] | Same as SERVER=. |
| UPDATE_ISOLATION_LEVEL= | No[4] | No[4] | The option is ignored. |
| UPDATE_LOCK_TYPE= | No[4] | No[4] | The option is ignored. |
| UPDATE_MODE_WAIT= | No[4] | No[4] | The option is ignored. |
| USER= | Yes[2] | Yes[2] | |
| UTILCONN_TRANSIENT= | N/A | N/A | In-database SAS/STAT and SAS/ETS procedures use separate connections from SAS/ACCESS software. |

## Using SAS/ACCESS Software with Teradata: Some Cautionary Notes

See the chapter "SAS/ACCESS Interface to Teradata" in *SAS/ACCESS for Relational Databases: Reference* for the complete documentation regarding the use of SAS/ACCESS software with Teradata.

- When a SAS/STAT or SAS/ETS procedure performs in-database computation, SAS/ACCESS software is not used for downloading the rows of the DATA= data set from the DBMS or for uploading the rows of the OUT= data set to the DBMS. Rather, SAS/ACCESS software is used as follows:

  - When the SAS System parses the statements of a SAS/STAT or SAS/ETS procedure, SAS/ACCESS software looks up the variable names in the DATA= data set. Or, when you do not specify an explicit variable list, SAS/ACCESS software obtains the names of the variables from the DBMS.
  - SAS/ACCESS software tells the procedure which SAS data types and SAS formats are assigned to the variables in both the DATA= and OUT= data sets when those data sets are DBMS tables.
  - When a SAS/STAT or SAS/ETS procedure attempts to create an OUT= data set, SAS/ACCESS software generates the CREATE TABLE command and sends it to the DBMS.

  Because SAS/ACCESS software is used for these purposes, numerous SAS/ACCESS options affect in-database computation as described in this and subsequent sections. In addition, SAS/ACCESS options are important for using the DATA step or out-of-database procedures to create DBMS tables, and these options can have important consequences for subsequent analyses that are performed using in-database procedures.

- In statistical applications, you should exercise extreme caution with the BULKLOAD=YES option in the LIBNAME statement. The BULKLOAD=YES option omits duplicate rows. It is not unusual for statistical data sets to have duplicate rows, and omitting them produces incorrect answers. This behavior affects the DATA step and procedures that do not perform in-database computation. This behavior does not affect OUT= data sets that are created in-database by the PRINCOMP or SCORE procedures.

- You should usually specify the following SAS/ACCESS options regardless of whether you are performing in-database computations:
  - Either specify OPTIONS NOLABEL or specify the DBSASLABEL=NONE option in a LIBNAME statement to prevent SAS/ACCESS software from assigning default labels and thus adding spurious variable labels to procedure listings, ODS tables, and output SAS data sets. (Teradata tables do not store SAS variable labels. Variable labels must be specified in the PROC step using a LABEL or ATTRIB statement. However, when SAS/ACCESS software creates a SAS data set from a DBMS table, it assigns the variable names as labels by default.)
  - Specify DBFMTIGNORE in an OPTIONS statement to prevent SAS/ACCESS software from assigning data types to output variables based on their SAS formats. Without the DBFMTIGNORE option, output data sets in the DBMS (including ODS tables) might suffer from loss of precision or numeric overflow, causing loss of data. See the subsection “LIBNAME Statement Data Conversions” in the section “Data Types for Teradata” in *SAS/ACCESS 9.3 for Relational Databases: Reference*.
- When SAS/ACCESS software reads a Teradata table or creates an ordinary SAS data set from a Teradata table, it assigns formats to the SAS variables based on the Teradata data types. These formats are often inappropriate and can interfere with the proper operation of SAS/STAT, SAS/ETS, and Base SAS procedures such as PRINT, PRINCOMP, and TRANSREG, especially for character variables. When using such a data set, you might want to specify a FORMAT statement to remove the formats, such as the following:

```
FORMAT _CHARACTER_;
```

- If you use a RENAME= data set option, SAS/ACCESS software assigns the old variable names as variable labels.
- Most SAS data set options are not stored in Teradata tables. In particular, the TYPE= data set option is not stored in Teradata tables. For example, if you run the CORR procedure to create a TYPE=CORR data set as a Teradata table, the data set TYPE= is lost and you must specify the TYPE= data set option every time you use that table.
- Teradata has nulls instead of missing values. SAS software has 28 kinds of numeric missing values, but SAS/ACCESS software converts all of them to null when it creates a Teradata table from SAS input. SAS character missing values are the same as blank strings, but Teradata nulls are different from blank strings. When SAS/ACCESS software creates a Teradata table from SAS input, blank strings are converted to nulls. Conversely, when a Teradata table is converted to a SAS data set, numeric nulls are converted to ordinary SAS numeric missing values, and character nulls are converted to blank strings.

- Comparisons with Teradata nulls work differently from comparisons with SAS missing values. Teradata uses three-valued logic, so a comparison with a null produces an "unknown" logical value. In the SAS System, numeric missing values are regarded as less than any nonmissing value, and character missing values are (equal to) blank strings. Comparisons between two SAS missing values that the SAS System considers equal might yield either a null or a true value on Teradata, depending on how the SAS System translates the syntax.

- When the SAS System copies a column from a DBMS table to a new DBMS table, the new DBMS data type can differ from that of the original table. This is true for both in-database and out-of-database processing and is a consequence of the manner in which SAS/ACCESS software handles DBMS data types and SAS formats. See the subsection "LIBNAME Statement Data Conversions" in the section "Data Types for Teradata" in *SAS/ACCESS 9.3 for Relational Databases: Reference*. The change in data types happens in a DATA step with a SET or MERGE statement and in procedure statements that copy variables from the DATA= data set to the OUT= data set.

  You can exercise limited control over data types in output tables by using FORMAT or ATTRIB statements to assign formats. For out-of-database processing, you can use the DBTYPE= data set option to specify data types for output tables. However, the DBTYPE= data set option is not supported for in-database computation by the SAS/STAT and SAS/ETS procedures.

  SAS/ACCESS software does not change the following data types:

  - BYTE
  - CHAR
  - DATE
  - FLOAT (also known as REAL or DOUBLE PRECISION)
  - TIME
  - TIMESTAMP

Table 1.4 shows how data types are changed when the SAS System copies a Teradata column.

**Table 1.4** Data Type Changes

| Original Data Type | New Data Type |
|---|---|
| **Byte Data Types** | |
| VarByte(*n*) | Byte(*n*) |
| **Character Data Types** | |
| VarChar(*n*) | Char(*n*) |
| **Numeric Data Types** | |
| BYTEINT | SMALLINT |
| SMALLINT | INTEGER |
| INTEGER | DECIMAL(11) |
| DECIMAL | INTEGER |
| DECIMAL(1) | BYTEINT |
| DECIMAL(1,0) | BYTEINT |
| DECIMAL(1,1) | DECIMAL(2,1) |
| DECIMAL(2) | SMALLINT |
| DECIMAL(2,0) | SMALLINT |
| DECIMAL(2,1) | DECIMAL(3,1) |
| DECIMAL(2,2) | DECIMAL(3,2) |
| DECIMAL(3) | SMALLINT |
| DECIMAL(3,0) | SMALLINT |
| DECIMAL(3,1) | DECIMAL(4,1) |
| DECIMAL(3,2) | DECIMAL(4,2) |
| DECIMAL(3,3) | DECIMAL(4,3) |
| DECIMAL(4) | INTEGER |
| DECIMAL(4,0) | INTEGER |
| DECIMAL(4,1) | DECIMAL(5,1) |
| DECIMAL(4,2) | DECIMAL(5,2) |
| DECIMAL(4,3) | DECIMAL(5,3) |
| DECIMAL(4,4) | DECIMAL(5,4) |
| DECIMAL(5) | INTEGER |
| DECIMAL(5,0) | INTEGER |
| DECIMAL(5,1) | DECIMAL(6,1) |
| DECIMAL(5,2) | DECIMAL(6,2) |
| DECIMAL(5,3) | DECIMAL(6,3) |
| DECIMAL(5,4) | DECIMAL(6,4) |
| DECIMAL(5,5) | DECIMAL(6,5) |
| DECIMAL(6) | INTEGER |
| DECIMAL(6,0) | INTEGER |
| DECIMAL(6,1) | DECIMAL(7,1) |
| DECIMAL(6,2) | DECIMAL(7,2) |
| DECIMAL(6,3) | DECIMAL(7,3) |
| DECIMAL(6,4) | DECIMAL(7,4) |

| Original Data Type | New Data Type |
|---|---|
| DECIMAL(6,5) | DECIMAL(7,5) |
| DECIMAL(6,6) | DECIMAL(7,6) |
| DECIMAL(7) | INTEGER |
| DECIMAL(7,0) | INTEGER |
| DECIMAL(7,1) | DECIMAL(8,1) |
| DECIMAL(7,2) | DECIMAL(8,2) |
| DECIMAL(7,3) | DECIMAL(8,3) |
| DECIMAL(7,4) | DECIMAL(8,4) |
| DECIMAL(7,5) | DECIMAL(8,5) |
| DECIMAL(7,6) | DECIMAL(8,6) |
| DECIMAL(7,7) | DECIMAL(8,7) |
| DECIMAL(8) | INTEGER |
| DECIMAL(8,0) | INTEGER |
| DECIMAL(8,1) | DECIMAL(9,1) |
| DECIMAL(8,2) | DECIMAL(9,2) |
| DECIMAL(8,3) | DECIMAL(9,3) |
| DECIMAL(8,4) | DECIMAL(9,4) |
| DECIMAL(8,5) | DECIMAL(9,5) |
| DECIMAL(8,6) | DECIMAL(9,6) |
| DECIMAL(8,7) | DECIMAL(9,7) |
| DECIMAL(8,8) | DECIMAL(9,8) |
| DECIMAL(9) | DECIMAL(10) |
| DECIMAL(9,0) | DECIMAL(10) |
| DECIMAL(9,1) | DECIMAL(10,1) |
| DECIMAL(9,2) | DECIMAL(10,2) |
| DECIMAL(9,3) | DECIMAL(10,3) |
| DECIMAL(9,4) | DECIMAL(10,4) |
| DECIMAL(9,5) | DECIMAL(10,5) |
| DECIMAL(9,6) | DECIMAL(10,6) |
| DECIMAL(9,7) | DECIMAL(10,7) |
| DECIMAL(9,8) | DECIMAL(10,8) |
| DECIMAL(9,9) | DECIMAL(10,9) |
| DECIMAL(10) | DECIMAL(11) |
| DECIMAL(10,0) | DECIMAL(11) |
| DECIMAL(10,1) | DECIMAL(11,1) |
| DECIMAL(10,2) | DECIMAL(11,2) |
| DECIMAL(10,3) | DECIMAL(11,3) |
| DECIMAL(10,4) | DECIMAL(11,4) |
| DECIMAL(10,5) | DECIMAL(11,5) |
| DECIMAL(10,6) | DECIMAL(11,6) |
| DECIMAL(10,7) | DECIMAL(11,7) |
| DECIMAL(10,8) | DECIMAL(11,8) |
| DECIMAL(10,9) | DECIMAL(11,9) |
| DECIMAL(10,10) | DECIMAL(11,10) |
| DECIMAL(11) | DECIMAL(12) |
| DECIMAL(11,0) | DECIMAL(12) |

| Original Data Type | New Data Type |
|---|---|
| DECIMAL(11,1) | DECIMAL(12,1) |
| DECIMAL(11,2) | DECIMAL(12,2) |
| DECIMAL(11,3) | DECIMAL(12,3) |
| DECIMAL(11,4) | DECIMAL(12,4) |
| DECIMAL(11,5) | DECIMAL(12,5) |
| DECIMAL(11,6) | DECIMAL(12,6) |
| DECIMAL(11,7) | DECIMAL(12,7) |
| DECIMAL(11,8) | DECIMAL(12,8) |
| DECIMAL(11,9) | DECIMAL(12,9) |
| DECIMAL(11,10) | DECIMAL(12,10) |
| DECIMAL(11,11) | DECIMAL(12,11) |
| DECIMAL(12) | DECIMAL(13) |
| DECIMAL(12,0) | DECIMAL(13) |
| DECIMAL(12,1) | DECIMAL(13,1) |
| DECIMAL(12,2) | DECIMAL(13,2) |
| DECIMAL(12,3) | DECIMAL(13,3) |
| DECIMAL(12,4) | DECIMAL(13,4) |
| DECIMAL(12,5) | DECIMAL(13,5) |
| DECIMAL(12,6) | DECIMAL(13,6) |
| DECIMAL(12,7) | DECIMAL(13,7) |
| DECIMAL(12,8) | DECIMAL(13,8) |
| DECIMAL(12,9) | DECIMAL(13,9) |
| DECIMAL(12,10) | DECIMAL(13,10) |
| DECIMAL(12,11) | DECIMAL(13,11) |
| DECIMAL(12,12) | DECIMAL(13,12) |
| DECIMAL(13) | DECIMAL(14) |
| DECIMAL(13,0) | DECIMAL(14) |
| DECIMAL(13,1) | DECIMAL(14,1) |
| DECIMAL(13,2) | DECIMAL(14,2) |
| DECIMAL(13,3) | DECIMAL(14,3) |
| DECIMAL(13,4) | DECIMAL(14,4) |
| DECIMAL(13,5) | DECIMAL(14,5) |
| DECIMAL(13,6) | DECIMAL(14,6) |
| DECIMAL(13,7) | DECIMAL(14,7) |
| DECIMAL(13,8) | DECIMAL(14,8) |
| DECIMAL(13,9) | DECIMAL(14,9) |
| DECIMAL(13,10) | DECIMAL(14,10) |
| DECIMAL(13,11) | DECIMAL(14,11) |
| DECIMAL(13,12) | DECIMAL(14,12) |
| DECIMAL(13,13) | DECIMAL(14,13) |
| DECIMAL(14) | DECIMAL(15) |
| DECIMAL(14,0) | DECIMAL(15) |
| DECIMAL(14,1) | DECIMAL(15,1) |
| DECIMAL(14,2) | DECIMAL(15,2) |
| DECIMAL(14,3) | DECIMAL(15,3) |
| DECIMAL(14,4) | DECIMAL(15,4) |

| Original Data Type | New Data Type |
|---|---|
| DECIMAL(14,5) | DECIMAL(15,5) |
| DECIMAL(14,6) | DECIMAL(15,6) |
| DECIMAL(14,7) | DECIMAL(15,7) |
| DECIMAL(14,8) | DECIMAL(15,8) |
| DECIMAL(14,9) | DECIMAL(15,9) |
| DECIMAL(14,10) | DECIMAL(15,10) |
| DECIMAL(14,11) | DECIMAL(15,11) |
| DECIMAL(14,12) | DECIMAL(15,12) |
| DECIMAL(14,13) | DECIMAL(15,13) |
| DECIMAL(14,14) | DECIMAL(15,14) |
| DECIMAL(15) | DECIMAL(16) |
| DECIMAL(15,0) | DECIMAL(16) |
| DECIMAL(15,1) | DECIMAL(16,1) |
| DECIMAL(15,2) | DECIMAL(16,2) |
| DECIMAL(15,3) | DECIMAL(16,3) |
| DECIMAL(15,4) | DECIMAL(16,4) |
| DECIMAL(15,5) | DECIMAL(16,5) |
| DECIMAL(15,6) | DECIMAL(16,6) |
| DECIMAL(15,7) | DECIMAL(16,7) |
| DECIMAL(15,8) | DECIMAL(16,8) |
| DECIMAL(15,9) | DECIMAL(16,9) |
| DECIMAL(15,10) | DECIMAL(16,10) |
| DECIMAL(15,11) | DECIMAL(16,11) |
| DECIMAL(15,12) | DECIMAL(16,12) |
| DECIMAL(15,13) | DECIMAL(16,13) |
| DECIMAL(15,14) | DECIMAL(16,14) |
| DECIMAL(15,15) | DECIMAL(16,15) |
| DECIMAL(16) | DECIMAL(17) |
| DECIMAL(16,0) | DECIMAL(17) |
| DECIMAL(16,1) | DECIMAL(17,1) |
| DECIMAL(16,2) | DECIMAL(17,2) |
| DECIMAL(16,3) | DECIMAL(17,3) |
| DECIMAL(16,4) | DECIMAL(17,4) |
| DECIMAL(16,5) | DECIMAL(17,5) |
| DECIMAL(16,6) | DECIMAL(17,6) |
| DECIMAL(16,7) | DECIMAL(17,7) |
| DECIMAL(16,8) | DECIMAL(17,8) |
| DECIMAL(16,9) | DECIMAL(17,9) |
| DECIMAL(16,10) | DECIMAL(17,10) |
| DECIMAL(16,11) | DECIMAL(17,11) |
| DECIMAL(16,12) | DECIMAL(17,12) |
| DECIMAL(16,13) | DECIMAL(17,13) |
| DECIMAL(16,14) | DECIMAL(17,14) |
| DECIMAL(16,15) | DECIMAL(17,15) |
| DECIMAL(16,16) | DECIMAL(17,16) |
| DECIMAL(17) | DECIMAL(18) |

| Original Data Type | New Data Type |
|---|---|
| DECIMAL(17,0) | DECIMAL(18) |
| DECIMAL(17,1) | DECIMAL(18,1) |
| DECIMAL(17,2) | DECIMAL(18,2) |
| DECIMAL(17,3) | DECIMAL(18,3) |
| DECIMAL(17,4) | DECIMAL(18,4) |
| DECIMAL(17,5) | DECIMAL(18,5) |
| DECIMAL(17,6) | DECIMAL(18,6) |
| DECIMAL(17,7) | DECIMAL(18,7) |
| DECIMAL(17,8) | DECIMAL(18,8) |
| DECIMAL(17,9) | DECIMAL(18,9) |
| DECIMAL(17,10) | DECIMAL(18,10) |
| DECIMAL(17,11) | DECIMAL(18,11) |
| DECIMAL(17,12) | DECIMAL(18,12) |
| DECIMAL(17,13) | DECIMAL(18,13) |
| DECIMAL(17,14) | DECIMAL(18,14) |
| DECIMAL(17,15) | DECIMAL(18,15) |
| DECIMAL(17,16) | DECIMAL(18,16) |
| DECIMAL(17,17) | DECIMAL(18,17) |
| DECIMAL(*n*) for *n* > 17 | FLOAT |
| **Time Data Types** | |
| TIME with TIMEZONE | CHAR(21) |
| TIME(0) with TIMEZONE | CHAR(14) |
| TIME(1) with TIMEZONE | CHAR(16) |
| TIME(2) with TIMEZONE | CHAR(17) |
| TIME(3) with TIMEZONE | CHAR(18) |
| TIME(4) with TIMEZONE | CHAR(19) |
| TIME(5) with TIMEZONE | CHAR(20) |
| TIME(6) with TIMEZONE | CHAR(21) |
| **Timestamp Data Types** | |
| TIMESTAMP with TIMEZONE | CHAR(32) |
| TIMESTAMP(0) with TIMEZONE | CHAR(25) |
| TIMESTAMP(1) with TIMEZONE | CHAR(27) |
| TIMESTAMP(2) with TIMEZONE | CHAR(28) |
| TIMESTAMP(3) with TIMEZONE | CHAR(29) |
| TIMESTAMP(4) with TIMEZONE | CHAR(30) |
| TIMESTAMP(5) with TIMEZONE | CHAR(31) |
| TIMESTAMP(6) with TIMEZONE | CHAR(32) |
| **Interval Data Types** | |
| INTERVAL YEAR | CHAR(3) |
| INTERVAL YEAR(1) | CHAR(2) |
| INTERVAL YEAR(2) | CHAR(3) |
| INTERVAL YEAR(3) | CHAR(4) |
| INTERVAL YEAR(4) | CHAR(5) |

| Original Data Type | New Data Type |
|---|---|
| INTERVAL YEAR to MONTH | CHAR(6) |
| INTERVAL YEAR(1) to MONTH | CHAR(5) |
| INTERVAL YEAR(2) to MONTH | CHAR(6) |
| INTERVAL YEAR(3) to MONTH | CHAR(7) |
| INTERVAL YEAR(4) to MONTH | CHAR(8) |
| INTERVAL MONTH | CHAR(3) |
| INTERVAL MONTH(1) | CHAR(2) |
| INTERVAL MONTH(2) | CHAR(3) |
| INTERVAL MONTH(3) | CHAR(4) |
| INTERVAL MONTH(4) | CHAR(5) |
| INTERVAL DAY | CHAR(3) |
| INTERVAL DAY(1) | CHAR(2) |
| INTERVAL DAY(2) | CHAR(3) |
| INTERVAL DAY(3) | CHAR(4) |
| INTERVAL DAY(4) | CHAR(5) |
| INTERVAL DAY to HOUR | CHAR(6) |
| INTERVAL DAY(1) to HOUR | CHAR(5) |
| INTERVAL DAY(2) to HOUR | CHAR(6) |
| INTERVAL DAY(3) to HOUR | CHAR(7) |
| INTERVAL DAY(4) to HOUR | CHAR(8) |
| INTERVAL DAY to MINUTE | CHAR(9) |
| INTERVAL DAY(1) to MINUTE | CHAR(8) |
| INTERVAL DAY(2) to MINUTE | CHAR(9) |
| INTERVAL DAY(3) to MINUTE | CHAR(10) |
| INTERVAL DAY(4) to MINUTE | CHAR(11) |
| INTERVAL DAY to SECOND | CHAR(19) |
| INTERVAL DAY(1) to SECOND | CHAR(18) |
| INTERVAL DAY(2) to SECOND | CHAR(19) |
| INTERVAL DAY(3) to SECOND | CHAR(20) |
| INTERVAL DAY(4) to SECOND | CHAR(21) |
| INTERVAL HOUR | CHAR(3) |
| INTERVAL HOUR(1) | CHAR(2) |
| INTERVAL HOUR(2) | CHAR(3) |
| INTERVAL HOUR(3) | CHAR(4) |
| INTERVAL HOUR(4) | CHAR(5) |
| INTERVAL HOUR to MINUTE | CHAR(6) |
| INTERVAL HOUR(1) to MINUTE | CHAR(5) |
| INTERVAL HOUR(2) to MINUTE | CHAR(6) |
| INTERVAL HOUR(3) to MINUTE | CHAR(7) |
| INTERVAL HOUR(4) to MINUTE | CHAR(8) |
| INTERVAL HOUR to SECOND | CHAR(16) |
| INTERVAL HOUR(1) to SECOND | CHAR(15) |
| INTERVAL HOUR(2) to SECOND | CHAR(16) |
| INTERVAL HOUR(3) to SECOND | CHAR(17) |
| INTERVAL HOUR(4) to SECOND | CHAR(18) |
| INTERVAL MINUTE | CHAR(3) |

| Original Data Type | New Data Type |
|---|---|
| INTERVAL MINUTE(1) | CHAR(2) |
| INTERVAL MINUTE(2) | CHAR(3) |
| INTERVAL MINUTE(3) | CHAR(4) |
| INTERVAL MINUTE(4) | CHAR(5) |
| INTERVAL MINUTE to SECOND | CHAR(13) |
| INTERVAL MINUTE(1) to SECOND | CHAR(12) |
| INTERVAL MINUTE(2) to SECOND | CHAR(13) |
| INTERVAL MINUTE(3) to SECOND | CHAR(14) |
| INTERVAL MINUTE(4) to SECOND | CHAR(15) |
| INTERVAL SECOND | CHAR(10) |
| INTERVAL SECOND(1) | CHAR(9) |
| INTERVAL SECOND(2) | CHAR(10) |
| INTERVAL SECOND(3) | CHAR(11) |
| INTERVAL SECOND(4) | CHAR(12) |

## Indeterminate Row Order in a DBMS Table

In an ordinary SAS data set, the observations are ordered in a consistent way. For example, if you run the PRINT procedure several times, the observations are always printed in the same order. But in a DBMS table, the order of the rows is not defined. If you run PROC PRINT several times on a DBMS table, the rows might be printed in a different order every time. If you run a statistical analysis that depends on row order in a DBMS table (such as the Durbin-Watson statistic in the REG procedure or tables with ORDER=DATA in the FREQ procedure), the results might differ every time you run the analysis.

To get a consistent row order for a DBMS table, you can use the DBCONDITION= data set option to specify an ORDER BY clause with one or more variables that defines a unique order for the rows. However, the DBCONDITION= option works only for out-of-database processing. The DBCONDITION= option does not work with in-database processing.

Following is an example that uses the DBCONDITION= option for out-of-database processing to control the row order in PROC PRINT:

```
options ls=72 options nodate nostimer nonumber;
%let server  =sl91204;
%let user    =sas;
%let password=sas;
libname tera teradata server=&server user=&user password=&password;
```

The SAS System generates the following note:

```
NOTE: Libref TERA was successfully assigned as follows:
      Engine:        TERADATA
      Physical Name: sl91204
```

```
proc delete data=tera.test;
run;
```

The SAS System generates the following warning:

```
WARNING: File TERA.test.DATA does not exist.
```

```
data tera.test;
   do i=1 to 3;
      do j=1 to 4;
         x=rannor(12345);
         output;
      end;
   end;
run;
```

The SAS System generates the following note:

```
NOTE: The data set TERA.test has 12 observations and 3 variables.
```

```
title "Indeterminate Row Order";
proc print data=tera.test;
run;
```

The SAS System generates the following output:

```
              Indeterminate Row Order
                Obs    i    j        x
                  1    1    1    -0.04298
                  2    3    1     0.57221
                  3    1    2    -0.09999
                  4    3    2     0.17571
                  5    1    3    -0.24349
                  6    3    3    -1.44361
                  7    1    4    -0.22226
                  8    3    4     0.44887
                  9    2    1     0.07353
                 10    2    2     0.49937
                 11    2    3    -1.52119
                 12    2    4     0.79180
```

```
NOTE: There were 12 observations read from the data set TERA.test.
```

```
title "Order Rows by I and J";
proc print data=tera.test(dbcondition="order by i, j");
run;
```

The SAS System generates the following output:

```
              Order Rows by I and J
               Obs    i    j        x
                 1    1    1    -0.04298
                 2    1    2    -0.09999
                 3    1    3    -0.24349
                 4    1    4    -0.22226
                 5    2    1     0.07353
                 6    2    2     0.49937
                 7    2    3    -1.52119
                 8    2    4     0.79180
                 9    3    1     0.57221
                10    3    2     0.17571
                11    3    3    -1.44361
                12    3    4     0.44887

NOTE: There were 12 observations read from the data set TERA.test.
```

```
  title "Order Rows by X";
  proc print data=tera.test(dbcondition="order by x");
  run;
```

The SAS System generates the following output:

```
              Order Rows by X
               Obs    i    j        x
                 1    2    3    -1.52119
                 2    3    3    -1.44361
                 3    1    3    -0.24349
                 4    1    4    -0.22226
                 5    1    2    -0.09999
                 6    1    1    -0.04298
                 7    2    1     0.07353
                 8    3    2     0.17571
                 9    3    4     0.44887
                10    2    2     0.49937
                11    3    1     0.57221
                12    2    4     0.79180

NOTE: There were 12 observations read from the data set TERA.test.
```

```
  proc delete data=tera.test;
  run;
```

The SAS System generates the following note:

```
NOTE: Deleting TERA.test (memtype=DATA).
```

## SAS/ACCESS Data Set Options for Teradata

Table 1.5 describes whether SAS/ACCESS data set options for Teradata work correctly with the DATA= and OUT= SAS/STAT and SAS/ETS procedure options. The cell entries in the DATA= and OUT= columns have the following values:

- Yes—indicates that the option works for in-database computing
- No—indicates that the option does not work for in-database computing
- N/A (not applicable)—indicates that the option does not apply to in-database computing
- Sometimes—see Comment column for details

Some cell entries in the DATA= and OUT= columns have superscripts that are intended to convey additional information. The meanings of the superscripts are as follows:

- A superscript of 1 (Yes$^1$ or No$^1$) indicates an option that correctly affects in-database processing as expected.
- A superscript of 2 (Yes$^2$ or No$^2$) indicates an option that works differently for in-database computing compared to out-of-database computing.
- A superscript of 3 (Yes$^3$, No$^3$, N/A$^3$, or Sometimes$^3$) indicates an option that causes the procedure to use SAS/ACCESS for out-of-database computing.
- A superscript of 4 (Yes$^4$ or No$^4$) indicates an option that neither works for in-database computing nor prevents in-database computing, but is unlikely to cause a serious error. See the Comment column for more information.
- A superscript of 5 (Yes$^5$ or No$^5$) indicates an option that might cause incorrect answers or Teradata syntax errors.

**NOTE:** As a data set option, DATABASE= is an alias for SCHEMA=, but as LIBNAME options, DATABASE= and SCHEMA= are distinct options. The data set options DATABASE= and SCHEMA= work the same way as the LIBNAME option SCHEMA=.

If you specify the same option as both a data set option and as a LIBNAME option, the value of the data set option is used for that data set.

**Table 1.5** SAS/ACCESS Data Set Options for Teradata

| Data Set Option | DATA= | OUT= | Comments |
|---|---|---|---|
| BL_CONTROL= | N/A | N/A | For out-of-database I/O. |
| BL_DATAFILE= | N/A | N/A | For out-of-database I/O. |

| Data Set Option | DATA= | OUT= | Comments |
|---|---|---|---|
| BL_LOG= | N/A | N/A | For out-of-database I/O. |
| BUFFERS= | N/A | N/A | For out-of-database I/O. |
| BULKLOAD= | N/A | N/A | For out-of-database I/O. |
| CAST= | N/A | N/A | For out-of-database I/O. |
| CAST_OVERHEAD_MAXPERCENT= | N/A | N/A | For out-of-database I/O. |
| DATABASE= | No[5] | No[5] | DATABASE= is an alias for SCHEMA= as a data set option. DATABASE= as a data set option does not work like DATABASE= as a LIBNAME option. |
| DBCOMMIT= | N/A | N/A | For out-of-database I/O. |
| DBCONDITION= | No[3] | No[3] | |
| DBCREATE_TABLE_OPTS= | N/A | Yes[1] | |
| DBFORCE= | N/A | N/A | In-database computation works like DBFORCE=YES, regardless of what value is specified for DBFORCE=. SAS/ACCESS software defaults to DBFORCE=NO. However, DBFORCE= would have an effect on in-database computation only if DBTYPE= specified a Teradata data type that resulted in truncation, whereas specifying DBTYPE= prevents in-database computation. |
| DBINDEX= | N/A | N/A | For out-of-database I/O. |
| DBKEY= | N/A | N/A | For out-of-database I/O. |

| Data Set Option | DATA= | OUT= | Comments |
|---|---|---|---|
| DBLABEL= | N/A | N/A | DBLABEL= applies only when the DATA= data set is a SAS data set with variable labels and the OUT= data set is a DBMS table. In-database computation cannot be performed when the DATA= data set is not a DBMS table. |
| DBMASTER= | N/A | N/A | |
| DBNULL= | N/A[3] | No[3] | |
| DBSASLABEL= | Yes[1] | N/A | DBSASLABEL=NONE is recommended for correct procedure output, but it is normally easier to specify this in the LIBNAME statement. |
| DBSLICE= | N/A | N/A | For out-of-database I/O. |
| DBSLICEPARM= | N/A | N/A | For out-of-database I/O. |
| DBTYPE= | N/A[3] | Sometimes[3] | DBTYPE= prevents in-database computation because some Teradata types such as DATE and TIMESTAMP have not been implemented. |
| ERRLIMIT= | N/A | N/A | For out-of-database I/O. |
| MBUFSIZE= | N/A | N/A | For out-of-database I/O. |
| ML_CHECKPOINT= | N/A | N/A | For out-of-database I/O. |
| ML_ERROR1= | N/A | N/A | For out-of-database I/O. |
| ML_ERROR2= | N/A | N/A | For out-of-database I/O. |
| ML_LOG= | N/A | N/A | For out-of-database I/O. |
| ML_RESTART= | N/A | N/A | For out-of-database I/O. |
| ML_WORK= | N/A | N/A | For out-of-database I/O. |
| MULTILOAD= | N/A | N/A | For out-of-database I/O. |

| Data Set Option | DATA= | OUT= | Comments |
|---|---|---|---|
| MULTISTMT= | N/A | N/A | For out-of-database I/O. |
| NULLCHAR= | N/A[3] | No[3] | |
| NULLCHARVAL= | N/A[3] | No[3] | |
| PRESERVE_COL_NAMES=NO | No[3] | No[3] | The default value YES works. |
| READ_ISOLATION_LEVEL= | No[4] | No[4] | The option is ignored. |
| READ_LOCK_TYPE= | No[4] | No[4] | The option is ignored. |
| READ_MODE_WAIT= | No[4] | No[4] | The option is ignored. |
| SCHEMA= | Yes[1] | Yes[1] | |
| SET= | N/A | Yes[2] | If the DATA= table contains duplicate rows, the procedure attempts to insert duplicate rows into the OUT= table, and the results for in-database and out-of-database processing differ. With in-database processing, duplicate rows are discarded and do not cause errors. With out-of-database processing, duplicate rows cause errors and a ROLLBACK is issued. With MODE=TERADATA, the ROLLBACK produces an empty or incomplete table. With MODE=ANSI, the ROLLBACK produces an empty table. |
| SLEEP= | N/A | N/A | For out-of-database I/O. |
| TENACITY= | N/A | N/A | For out-of-database I/O. |
| UPDATE_ISOLATION_LEVEL= | No[4] | No[4] | The option is ignored. |
| UPDATE_LOCK_TYPE= | No[4] | No[4] | The option is ignored. |
| UPDATE_MODE_WAIT= | No[4] | No[4] | The option is ignored. |

## Base SAS Data Set Options

Table 1.6 describes whether the Base SAS data set options work with the DATA= and OUT= SAS/STAT and SAS/ETS procedure options for in-database computing. The cell entries in the DATA= and OUT= columns have the following values:

- Yes—indicates that the option works for in-database computing
- No—indicates that the option does not work for in-database computing
- N/A (not applicable)—indicates that the option does not apply to in-database computing

Some cell entries in the DATA= and OUT= columns have superscripts that are intended to convey additional information. The meanings of the superscripts are as follows:

- A superscript of 1 (Yes$^1$ or No$^1$) indicates an option that correctly affects in-database processing as expected.
- A superscript of 2 (Yes$^2$, No$^2$, or N/A$^2$) indicates an option that causes SAS/ACCESS syntax errors or warnings. When SAS/ACCESS software issues a warning message, the option is ignored by the procedure.
- A superscript of 3 (Yes$^3$, No$^3$, or N/A$^3$) indicates an option that causes the procedure to use SAS/ACCESS for out-of-database computing.

**Table 1.6** Base SAS Data Set Options

| Data set option | DATA= | OUT= | Comments |
|---|---|---|---|
| ALTER= | No$^2$ | No$^2$ | SAS passwords are not supported on DBMSs. |
| BUFNO= | N/A | N/A | For out-of-database I/O. |
| BUFSIZE= | N/A | N/A | For out-of-database I/O. |
| CNTLLEV= | N/A | N/A | For SAS data sets only. |
| COMPRESS= | N/A | N/A | For SAS data sets only. |
| DLDMGACTION= | N/A | N/A | For SAS data sets only. |
| DROP= | Yes$^1$ | No$^3$ | |
| ENCODING= | No$^2$ | No$^2$ | |

| Data set option | DATA= | OUT= | Comments |
|---|---|---|---|
| ENCRYPT= | No$^2$ | No$^2$ | Requires SAS passwords. |
| FILECLOSE= | N/A | N/A | For tape data sets only. |
| FIRSTOBS= | No$^3$ | No$^3$ | DBMS tables have no inherent row order, and FIRSTOBS= is defined in terms of row order. |
| GENMAX= | No$^2$ | No$^2$ | Neither the Teradata nor TSSQL drivers support generation options that are explicitly specified in the procedure statement, and the procedure does not know whether a generation number is explicit or implicit. |
| GENNUM= | No$^2$ | No$^2$ | Neither the Teradata nor TSSQL drivers support generation options that are explicitly specified in the procedure statement, and the procedure does not know whether a generation number is explicit or implicit. |
| IDXNAME= | N/A | N/A | For out-of-database I/O. |
| IDXWHERE= | N/A | N/A | For out-of-database I/O. |
| IN= | N/A | N/A | For DATA step only. |
| INDEX= | N/A$^2$ | No$^2$ | For SAS data sets only. |
| KEEP= | Yes$^1$ | No$^3$ | |
| LABEL= | No$^2$ | No$^2$ | SAS data set attributes cannot be stored in Teradata. However, LABEL= should work on DATA=, just as TYPE= works on DATA=. |
| OBS= | No$^3$ | No$^3$ | DBMS tables have no inherent row order, and FIRSTOBS= is defined in terms of row order. |
| OBSBUF= | N/A | N/A | For SAS views only. |
| OUTREP= | N/A | N/A | For SAS data sets only. |
| POINTOBS= | N/A | N/A | For SAS data sets only. |
| PW= | No$^2$ | No$^2$ | SAS passwords are not supported on DBMSs. |

| Data set option | DATA= | OUT= | Comments |
|---|---|---|---|
| PWREQ= | No[2] | No[2] | SAS passwords are not supported on DBMSs. |
| READ= | No[2] | No[2] | SAS passwords are not supported on DBMSs. |
| RENAME= | No[3] | No[3] | SAS supervisor does not tell the procedure what the old names are. |
| REPEMPTY= | N/A | N/A | For SAS data sets only. |
| REPLACE= | N/A | N/A | For SAS data sets only. |
| REUSE= | N/A | N/A | For SAS data sets only. |
| SORTEDBY= | N/A | N/A | For SAS data sets only. |
| SPILL= | N/A | N/A | For SAS views only. |
| TOBSNO= | N/A | N/A | For out-of-database I/O. |
| TYPE= | No[3] | No[3] | SAS data set attributes cannot be stored in Teradata, and most TYPE= values indicate that the data set contains summary statistics. |
| WHERE= | Yes[1] | No[3] | |
| WHEREUP= | N/A | N/A | For updating a table, usually via out-of-database I/O. |
| WRITE= | No[2] | No[2] | SAS passwords are not supported on DBMSs. |

# Deploying and Using SAS Formats in Teradata

## Using SAS Formats

SAS formats are basically mapping functions that change an element of data from one format to another. For example, some SAS formats change numeric values to various currency formats and date-and-time formats.

The SAS System supplies many formats. You can also use the SAS FORMAT procedure to define custom formats that replace raw data values with formatted character values. For example, the following PROC FORMAT code creates a custom format called $REGION that maps ZIP codes to geographic regions.

```
proc format;
   value $region
   '02129', '03755', '10005' = 'Northeast'
   '27513', '27511', '27705' = 'Southeast'
   '92173', '97214', '94105' = 'Pacific';
run;
```

SAS programs frequently use both user-defined formats and formats that the SAS System supplies. Although they are referenced in numerous ways, using the PUT function in the SQL procedure is of particular interest for SAS in-database processing.

The PUT function takes a format reference and a data item as input and returns a formatted value. For example, this SQL procedure query uses the PUT function to summarize sales by region from a table of all customers:

```
select put(zipcode,$region.) as region,
   sum(sales) as sum_sales from sales.customers
   group by region;
```

The SAS SQL processor knows how to process the PUT function. Currently, SAS/ACCESS Interface to Teradata returns all rows of unformatted data in the SALES.CUSTOMERS table in the Teradata database to the SAS System for processing.

The SAS in-database technology deploys ("publishes") the PUT function implementation to Teradata as a new function named SAS_PUT(). Similar to any other programming language function, the SAS_PUT() function can take one or more input parameters and return an output value.

The SAS_PUT() function supports the use of SAS formats. You can specify the SAS_PUT() function in SQL queries that the SAS System submits to Teradata in one of two ways:

- implicitly by enabling the SAS System to automatically map PUT function calls to SAS_PUT() function calls
- explicitly by using the SAS_PUT() function directly in your SAS program

If you used the SAS_PUT() function in the previous example, Teradata formats the ZIP code values with the $REGION format and processes the GROUP BY clause using the formatted values.

By publishing the PUT function implementation to Teradata as the SAS_PUT() function to support you can realize these advantages:

- You can process the entire SQL query inside the database, which minimizes data transfer (I/O).
- The SAS format processing leverages the scalable architecture of the DBMS.
- The results are grouped by the formatted data and are extracted from the Teradata Enterprise Data Warehouse (EDW).

Deploying SAS formats to execute inside a Teradata database can enhance performance and exploit Teradata parallel processing. This is accomplished when you install the SAS/ACCESS Interface to Teradata software, which consists of the following three components:

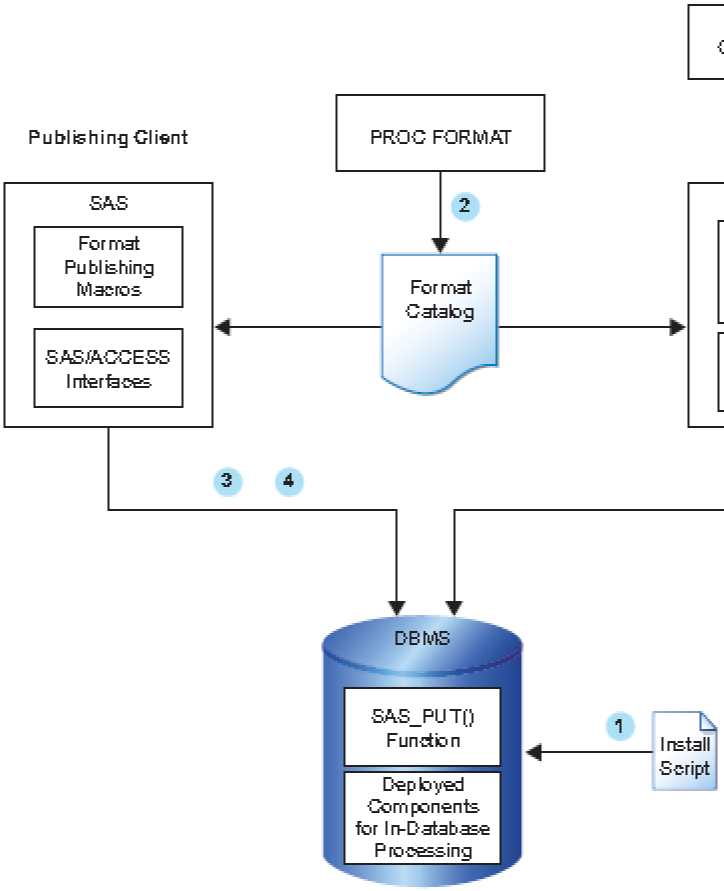
- SAS/ACCESS Interface to Teradata engine—controls the exchange of information between the SAS System and the Teradata EDW
- SAS Formats Library for Teradata—contains a library of SAS formats and the SAS_PUT UDF that maps the SAS PUT() function to the SAS_PUT function in Teradata
- SAS Accelerator Publishing Agent—contains macros that enable you to publish the SAS formats and user-defined formats to Teradata

## How It Works

By using a SAS formats publishing macro, you can generate a SAS_PUT() function that enables you to execute PUT function calls inside the Teradata EDW. You can reference the formats that the SAS System supplies and most custom formats that you create using the FORMAT procedure.

The SAS formats publishing macro takes a SAS format catalog and publishes it to the Teradata EDW. Inside the Teradata EDW, a SAS_PUT() function, which emulates the PUT function, is created and registered for use in SQL queries.

**Figure 1.1** Process Flow Diagram



Here is the basic process flow:

1. Install the SAS 9.3 Formats Library for Teradata in the Teradata EDW. This library contains many of the formats that are available in Base SAS software.

   **NOTE:** This is a one-time installation process.

   For more information, see the section "Formats That SAS Supplies in the Teradata EDW."

2. If necessary, create your custom formats by using PROC FORMAT and create a permanent catalog by using the LIBRARY= option.

   For more information, see the section "User-Defined Formats in the Teradata EDW" in *SAS/ACCESS 9.3 for Relational Databases: Reference, Second Edition* and the FORMAT procedure in the *Base SAS Procedures Guide*.

3. Start SAS 9.3 and run the %INDTD_PUBLISH_FORMATS macro. This macro creates the files that are needed to build the SAS_PUT() function and publishes those files to the Teradata EDW.

   The %INDTD_PUBLISH_FORMATS macro performs these tasks:

   - produces the set of .c and .h files that are necessary to build the SAS_PUT() function

  - produces a script of the Teradata commands that are necessary to register the SAS_PUT() function on the Teradata EDW

4. After the %INDTD_PUBLISH_FORMATS macro creates the script, SAS/ACCESS Interface to Teradata executes the script and publishes the files to the Teradata EDW.

5. Teradata compiles the .c and .h files and creates the SAS_PUT() function.

   The SAS_PUT() function is available to use in any SQL expression and to use typically wherever you would use Teradata built-in functions. For more information, see "Using the SAS_PUT() Function in the Teradata EDW" in *SAS/ACCESS 9.3 for Relational Databases: Reference, Second Edition*.

   **NOTE:** The SAS_PUT() function uses Latin-1 encoding. Any character that cannot be represented in Latin-1 might cause unexpected or unsuccessful behavior.

## Formats That SAS Supplies in the Teradata EDW

The SAS System supplies many formats for use in the SAS_PUT() function.

You must install the SAS 9.3 Formats Library for Teradata on the same machine where you have installed SAS Foundation. This is a one-time installation process.

After you install the SAS 9.3 Formats Library and run the %INDTD_PUBLISH_FORMATS macro, the SAS_PUT() function can call these formats.

For information about how to install and configure the SAS 9.3 Formats Library for Teradata, see the chapter on post-installation configuration for the SAS Accelerator Publishing Agent software in the *Configuration Guide for SAS 9.3 Foundation* for your operating environment.

**NOTE:** The SAS Scoring Accelerator for Teradata also uses these libraries. For more information about this product, see the *SAS Scoring Accelerator for Teradata: User's Guide*.

## User-Defined Formats in the Teradata EDW

You can use PROC FORMAT to create user-defined formats and store them in a format catalog. You can then use the %INDTD_PUBLISH_FORMATS macro to export the user-defined format definitions to the Teradata EDW where the SAS_PUT() function can reference them.

If you use the FMTCAT= option to specify a format catalog in the %INDTD_PUBLISH_FORMATS macro, these restrictions and limitations apply:

- Trailing blanks in PROC FORMAT labels are lost when publishing a picture format.

- Avoid using PICTURE formats with the MULTILABEL option. You cannot successfully create a CNTLOUT= data set when PICTURE formats are present. This is a known problem in PROC FORMAT.

- If you are using a character set encoding other than Latin1, picture formats are not supported. The picture format supports only Latin1 characters.
- If you use the MULTILABEL option, only the first label that is found is returned. For more information, see the PROC FORMAT MULTILABEL option in the *Base SAS Procedures Guide*.
- The %INDTD_PUBLISH_FORMATS macro rejects a format unless the LANGUAGE= option is set to English or is not specified.
- Although the format catalog can contain informats, the %INDTD_PUBLISH_FORMATS macro ignores the informats.
- User-defined formats that include a format that SAS supplies are not supported.

## Overview of the Publishing Process

The %INDTD_PUBLISH_FORMATS macro creates the files that are needed to build the SAS_PUT() function and publishes these files to the Teradata EDW.

The %INDTD_PUBLISH_FORMATS macro also publishes the formats that are included in the SAS 9.3 Formats Library for Teradata. This makes many formats that SAS supplies available inside Teradata.

In addition to SAS formats, you can also publish the PROC FORMAT definitions that are contained in a single SAS format catalog by using the FMTCAT= option. The process of publishing a PROC FORMAT catalog entry converts the range label pairs into embedded data in Teradata.

**NOTE:** If you specify more than one format catalog using the FMTCAT= option, the last format that you specify is published.

The %INDTD_PUBLISH_FORMATS macro performs the following tasks:

- creates .h and .c files, which are necessary to build the SAS_PUT() function.
- produces a script of Teradata commands that are necessary to register the SAS_PUT() function in the Teradata EDW.
- uses SAS/ACCESS Interface to Teradata to execute the script and publish the files to the Teradata EDW

For information about publishing formats, see the section “Deploying and Using SAS Formats in Teradata” in *SAS/ACCESS 9.3 for Relational Databases: Reference*.

## Using SAS Formats

To each variable in an ordinary SAS data set, you can assign a format, field width, and optional number of decimal places. The SAS System stores the format information for each variable in the SAS data set along with other attributes such as variable labels. Once you have published the SAS formats and any user-defined formats that you want to use, those formats can be used for in-database processing. However, a Teradata table cannot explicitly store the name of the SAS format, field width, and number of decimal places for each column. Instead, when SAS/ACCESS software opens a Teradata table, SAS/ACCESS assigns format information to each column based on the Teradata data type of the column (see the section "Data Types for Teradata" in *SAS/ACCESS 9.3 for Relational Databases: Reference*). If you want to use a format for any column that differs from the format that SAS/ACCESS software assigns to that column, you must specify the FORMAT statement in the same PROC step in which you want the format to be used.

## BY Groups and In-Database Computing

By default, SAS/STAT and SAS/ETS procedures that perform in-database computation define BY groups according to the formatted values of the BY variables. However, except for the $w. and $CHARw. formats, which do not truncate the data values, SAS formats must first be "published" before they can be used for in-database computing. If your SAS program uses PROC FORMAT to create one or more user-defined formats, you need to publish those user-defined formats as well. The process of publishing SAS formats is described in the section "Deploying and Using SAS Formats in Teradata" in this document. Additional details are also available in the section "Deploying and Using SAS Formats in Teradata" in *SAS/ACCESS 9.3 for Relational Databases: Reference*.

As indicated previously in the section "Using SAS Formats," after you have published the SAS formats and any user-defined formats that you want to use, those formats can be used for in-database computing. However, a Teradata table cannot explicitly store the name of the SAS format, field width, and number of decimal places for each column. Instead, when SAS/ACCESS software opens a Teradata table, it assigns format information to each column based on the Teradata data type of the column (see the section "Data Types for Teradata" in *SAS/ACCESS 9.3 for Relational Databases: Reference*). If you want to use a format for any column that differs from the format that SAS/ACCESS software assigns to that column, you must specify the FORMAT statement in the same PROC step in which you want the format to be used.

By default, if the SAS formats are not published or if any specific format used for a BY variable is not published, then the procedure prints error messages about the UDFs or specific formats that have not been published. If you want the procedure to run anyway without using the SAS_PUT UDF for unpublished formats, specify the following statement:

```
   OPTIONS NOFMTERR;
```

This statement causes the SAS/STAT or SAS/ETS procedure to define BY groups according to the unformatted values of the BY variables when the formats of the BY variables have not been published.

You should also be aware of the following conditions regarding BY groups and in-database computing:

- The DESCENDING option in the BY statement is supported for in-database computation.
- The NOTSORTED option in the BY statement is not supported for in-database computation, because the results with NOTSORTED depend on the order of the rows in a SAS data set, whereas the rows in a DBMS table do not have any defined order.

- For BY processing in-database, the SAS session encoding must be Latin-1. SAS_PUT uses the Latin-1 encoding, and any character that cannot be represented in Latin-1 can cause problems.
- SAS_PUT does not support Teradata columns with types of BYTE, VARBYTE, BLOB, GRAPHIC, or VARGRAPHIC. If you try to use SAS_PUT with any of these types, you are likely to get a Teradata error message saying, “Function sas_put does not exist”, even if SAS_PUT has been published.

There are also unusual situations in which the BY groups for in-database computation differ from the BY groups for out-of-database computation. This arises from the fact that BY groups for out-of-database computation depend on the order of the observations, in addition to the formatted values. The in-database behavior is a feature, not a defect. Here is an example:

```
title "Try To Use a Discontiguous Format for BY Groups";
options nolabel nodate nostimer;

%let server  =s191204;
%let user    =sas;
%let password=sas;

%let indconn=%str(user=&user password=&password server=&server);
libname dbms teradata &indconn;
```

The parity format assigns a formatted value of O to odd numbers and X to even numbers:

```
proc format;
   value parity
      1='O'
      2='X'
      3='O'
      4='X'
      5='O'
      6='X'
      7='O';
run;
```

Next, you publish the user-defined format on the DBMS:

```
%indtdpf;
%indtd_publish_formats(
   fmtcat=work.formats,
   action=create,
   mode=unprotected,
   outdir=sasuser,
   permdir=sasuser,
   fmttable=format_table );

proc print data=dbms.format_table;
where fmtname='PARITY';
run;
```

However, you cannot simply use the parity format to define BY groups based on odd and even numbers when using ordinary SAS data sets, as opposed to DBMS tables. The SAS System requires each BY group to be a contiguous sequence of observations, so you would have to create a new variable that contains the formatted value and then sort the data set by that new variable.

An out-of-database analysis of a DBMS table works the same way, because SAS/ACCESS software returns the rows ordered by the unformatted values of the BY variable. On the other hand, suppose you perform an in-database analysis. The rows in the DBMS table have no inherent order, so the BY groups are defined solely by the formatted values of the BY variable. To illustrate, suppose you create a data set that contains a variable named group with integer values from 1 to 7 and a variable named formatted_group with the formatted values of group.

```
proc delete data=dbms.par_test;
data dbms.par_test;
   retain random group value formatted_group;
   do group = 1 to 7;
      random = ranuni(123456);
      value  = ceil(group/2);
      formatted_group = put( group, parity1. );
      output;
   end;
run;

proc print data=dbms.par_test(dbcondition='order by "group"');
run;
```

The following analysis fails because the two groups defined by the formatted values are not contiguous sequences of observations when sorted by the unformatted values. There are many error messages because each BY group contains only one observation, which the PRINCOMP procedure considers to be an error.

```
title2 "Analysis 0: DATA out-of-db, OUT out-of-db, by group";
title3 "Should produce many error messages";
options sqlgeneration=none;
proc princomp data=dbms.par_test out=out;
   var random value;
   by group;
   format group parity1.;
run;
```

To get an out-of-database analysis to work, you have to use the explicitly formatted variable, formatted_group as in the following statements. Having both variables in the BY statement is redundant, but necessary, to get the variable group copied to the OUTSTAT= data set for comparisons.

```
title2 "Analysis 1: DATA out-of-db, OUT out-of-db, by formatted_group group";
title3 "This analysis is correct";
options sqlgeneration=none;
proc princomp data=dbms.par_test out=out1 outstat=stat1 cov;
   by formatted_group group;
   format group parity1.;
run;
```

Next, sort by the variable formatted_group first to get the comparisons to work:

```
proc sort data=stat1;
   by formatted_group group _type_ _name_;
   format _all_;
run;

proc print data=stat1;
run;

proc sort data=out1 out=sort1;
   by formatted_group group value;
   format _all_;
run;

proc print data=sort1;
run;
```

The in-database analysis works correctly because the BY groups are defined solely by the formatted values. Having both variables in the BY statement is redundant, but necessary, to get the variable group copied to the OUTSTAT= data set for comparisons.

```
title2 "Analysis 2: DATA in-db, OUT in-db, by group formatted_group";
title3 "This analysis is correct";
options sqlgeneration=dbms;
proc delete data=dbms.out2;
run;

proc princomp data=dbms.par_test out=dbms.out2 outstat=stat2 cov;
   var random value;
   by group formatted_group;
   format group parity1.;
run;
```

Next, swap the order of the BY variables to make the comparisons work:

```
proc sort data=stat2;
   by formatted_group group _type_ _name_;
   format _all_;
run;

proc print data=stat2;
run;

proc sort data=dbms.out2 out=sort2(LABEL='Principal Component Scores');
   by formatted_group group value;
   format _all_;
run;

proc print data=sort2;
run;

proc compare data=stat1 compare=stat2
   method=absolute criterion=1e-10 error;
run;

proc compare data=sort1 compare=sort2
   method=absolute criterion=1e-10 error;
```

```
run;
```

Finally, you perform a clean up, deleting unneeded data sets and dropping the formats:

```
proc delete data=dbms.par_test;
proc delete data=dbms.out2;
run;

%indtd_publish_formats(
   action=drop,
   outdir=sasuser,
   permdir=sasuser,
   fmttable=format_table );
```

# Support for Teradata Data Types

## Variables in the DATA= Data Set

When performing in-database computation with the SAS/STAT or SAS/ETS in-database procedures, all of the variables in the DATA= data set must be confined to the following Teradata data types, regardless of whether they are used in the analysis:

- BYTEINT
- SMALLINT
- INTEGER
- DECIMAL (ANSI NUMERIC)
- FLOAT (ANSI REAL or DOUBLE PRECISION)
- DATE
- TIME
- TIMESTAMP
- CHARACTER
- VARCHAR
- LONG VARCHAR
- CLOB

If the DATA= data set contains any variables of the following types, the procedure cannot perform in-database computation, but performs out-of-database computation instead:

- BYTE
- VARBYTE
- BLOB
- INTERVAL
- TIME WITH TIME ZONE
- TIMESTAMP WITH TIME ZONE

The following Teradata data types are not supported by SAS/ACCESS software:

- BIGINT
- GRAPHIC
- VARGRAPHIC
- LONG VARGRAPHIC

## Numeric Computations

SAS/STAT procedures that support in-database computing can perform numeric in-database computations for the following Teradata data types, all of which are converted to FLOAT before being used in any arithmetic computation:

- BYTEINT
- SMALLINT
- INTEGER
- DECIMAL (ANSI NUMERIC)
- FLOAT (ANSI REAL or DOUBLE PRECISION)
- DATE
- TIME
- TIMESTAMP

The following Teradata data types cannot be used for numeric in-database computations because they are character types:

- CHARACTER

- VARCHAR
- LONG VARCHAR
- CLOB

## BY Processing

Columns with the following Teradata data types can be used as BY variables:

- BYTEINT
- SMALLINT
- INTEGER
- DECIMAL (ANSI NUMERIC)
- FLOAT (ANSI REAL OR DOUBLE PRECISION)
- DATE
- TIME
- TIMESTAMP
- CHARACTER
- VARCHAR
- LONG VARCHAR
- CLOB

# Conditions That Prevent In-Database Processing

This section describes conditions that are checked by the SAS/STAT and SAS/ETS procedures that support in-database computing. Specific procedures might have other conditions that prevent in-database computing. One example is any option that requires all rows of a table to be downloaded without summarization. All specific conditions are discussed in detail in the following sections that are devoted to the individual SAS/STAT and SAS/ETS in-database procedures:

- CANCORR Procedure Options Affected by In-Database Computing
- FACTOR Procedure Options Affected by In-Database Computing

- PRINCOMP Procedure Options Affected by In-Database Computing
- REG Procedure Options and Statements Affected by In-Database Computing
- SCORE Procedure Options Affected by In-Database Computing
- TIMESERIES Procedure Options Affected by In-Database Computing
- VARCLUS Procedure Options Affected by In-Database Computing

## Options

In-database computation is not supported when any of the options described in Table 1.7 are specified.

**Table 1.7** Options That Do Not Support In-Database Computation

| Option | Option Value | Specified in OPTIONS Statement | Specified in LIBNAME Statement | Specified as Data Set Option | Reason |
|---|---|---|---|---|---|
| SQLGENERATION | NONE | x | x | | |
| VALIDVARNAME | V6 or UPCASE | x | | | DBMS identifiers are usually case-sensitive. |
| ENCODING | Anything other than Latin-1 when there is a BY statement | x | | | The SAS_PUT UDF supports only Latin-1. |
| OBS | Any | x | | x | DBMS tables have no inherent row order. |
| FIRSTOBS | >1 | x | | x | DBMS tables have no inherent row order. |
| RENAME | Any | | | x | |
| DROP or KEEP | Any | | | In OUT= | |

**Table 1.7** *continued*

| Option | Option Value | Specified in OPTIONS Statement | Specified in LIBNAME Statement | Specified as Data Set Option | Reason |
|---|---|---|---|---|---|
| WHERE | Any | | | In OUT= | |
| READ | Any | | | x | SAS passwords are not supported in SQL, which has a very different security system. |
| WRITE | Any | | | x | SAS passwords are not supported in SQL, which has a very different security system. |
| ALTER | Any | | | x | SAS passwords are not supported in SQL, which has a very different security system. |
| GENMAX | Any | | | x | |
| GENNUM | Any | | | x | |
| DBCONDITION | Any | | | x | |
| DBTYPE | Any | | | x | |
| DBNULL | Any | | | x | |
| DBGEN_NAME | SAS | | x | x | Causes SQL syntax errors. |
| PRESERVE_COL_NAMES | No | | x | x | Causes SQL syntax errors. |
| PRESERVE_TAB_NAMES | No | | x | x | Causes SQL syntax errors. |
| PRESERVE_NAMES | No | | x | x | Causes SQL syntax errors. |

**Table 1.7** *continued*

| Option | Option Value | Specified in OPTIONS Statement | Specified in LIBNAME Statement | Specified as Data Set Option | Reason |
|---|---|---|---|---|---|
| CONNECTION_GROUP | Any | | x | x | The in-database SAS/STAT and SAS/ETS procedures cannot share connections with SAS/ACCESS software. |
| DBMSTEMP | Yes | | x | x | The in-database SAS/STAT and SAS/ETS procedures cannot share connections with SAS/ACCESS software. |
| DBCONINIT | Any | | x | x | The in-database SAS/STAT and SAS/ETS procedures cannot share connections with SAS/ACCESS software. |
| DBCONTERM | Any | | x | x | The in-database SAS/STAT and SAS/ETS procedures cannot share connections with SAS/ACCESS software. |

## LIBNAME Properties

In-database computation is not supported when either of the following conditions is true:

- The engine is not TERADATA.
- The LIBNAME statement specifies a concatenated library.

## Data Set and Variable Properties

In-database computation is not supported when any of the following conditions is true:

- The data set is not a DBMS table.
- The data set TYPE= attribute is not blank.
- The data set is a DBMS table and has variables named _TYPE_ and _NAME_, but TYPE=CORR, COV, or SSCP was not specified as a data set option.
- Any variable in the OUT= data set has a HEXw. format.
- The DATA= data set contains any variables with the following DBMS data types:
  - BYTE
  - VARBYTE
  - BLOB
  - INTERVAL
  - TIME WITH TIME ZONE
  - TIMESTAMP WITH TIME ZONE
  - BIGINT
  - GRAPHIC
  - VARGRAPHIC
  - LONG VARGRAPHIC

## BY Processing

In-database computation is not supported when any of the following conditions is true:

- The BY statement contains the NOTSORTED option.
- Any BY variable has a length or format width that exceeds 256 characters, the maximum supported by the SAS_PUT UDF.
- The total length of the BY variables exceeds 2,048 bytes, the maximum supported by the SAS_TACORR UDF.

- The total formatted width of the BY variables exceeds 2,048 bytes, the maximum supported by the SAS_TACORR UDF.
- One or more variables cannot be formatted because the SAS_PUT UDF has not been published in SYSLIB or in the current database and the NOFMTERR option was not specified.

## Compatibility of the DATA= and OUT= Data Sets

In-database computation is not supported for the DATA= data set when in-database computation is not used for the OUT= data set and there is a BY statement. This is because BY groups sometimes are not the same for both in-database and out-of-database computations.

In-database computation is not supported for the OUT= data set when in-database computation is not used for the DATA= data set.

In-database computation is not supported when the DATA= and OUT= data sets have different engines.

In-database computation is not supported when any variable has the following properties:

- is in both the DATA= and OUT= data sets
- does not have a DBMS character data type (CHAR, VARCHAR, CLOB)
- has a DBMS data type in the DATA= data set that belongs to a different group than the DBMS type of the corresponding variable in the OUT= data set, where the groups are as follows:
  - Numeric types:
    - BYTEINT
    - SMALLINT
    - INTEGER
    - DECIMAL
    - NUMERIC
    - FLOAT
    - REAL
    - DOUBLE PRECISION
  - Character types:
    - CHAR
    - VARCHAR
    - CLOB
  - DATE
  - TIME
  - TIMESTAMP

In most circumstances, if a procedure cannot perform in-database computation, it performs out-of-database computation instead. However, if the DBMS type of a variable belongs to different groups in the DATA= and OUT= data sets, the procedure issues an error message and quits. It might be possible to run the procedure by using SQLGENERATION=NONE.

If OPTIONS SQL_IP_TRACE=NOTE or OPTIONS SQL_IP_TRACE=ALL is specified, the procedure also issues a message if any variable has different DBMS types in the DATA= and OUT= data sets because Teradata often has trouble converting data from one type to another.

## Other Conditions

In-database computation is not supported when any of the following conditions is true:

- A variable is used more than once in a VAR, PARTIAL, or MODEL statement in the PRINCOMP, REG, or VARCLUS procedures.
- Two or more variables in the OUT= data set have the same name in the PRINCOMP or SCORE procedures.
- Any UDF required for in-database computation cannot be found in SYSLIB or the current database.

# In-Database Computing for the DATA= Data Set by the CANCORR, FACTOR, PRINCOMP, REG, and VARCLUS Procedures

The DATA= data set must be a Teradata table for in-database computing.

When the CANCORR, FACTOR, PRINCOMP, REG, or VARCLUS procedure performs in-database computation for the DATA= data set, the procedure generates an SQL query that computes the SSCP matrix. The query is passed to the DBMS and executed in-database. The results of the query are then passed back to the SAS System and stored in an ordinary SAS data set called Work._Mkxpx_. If there is a BY statement, Work._Mkxpx_ contains results for all of the BY groups. You do not need to understand the contents of Work._Mkxpx_. The contents are not documented and might change in future releases. Work._Mkxpx_ does not become the default input data set (_LAST_) even if the procedure creates no other data sets.

The SQL query requires the SAS_TACORR and SAS_TOVB UDFs to be published on the DBMS server. If these UDFs are absent, or if you do not have the privilege required for executing the UDFs, the SQL query fails. Usually, the database administrator installs these UDFs and grants the necessary privileges to people who want to use them.

## ODS Tables for the CANCORR, FACTOR, PRINCOMP, REG, and VARCLUS Procedures

ODS does not perform in-database processing. Most ODS tables contain statistics or other summaries of the data, so in-database processing would not be applicable. Although the statistics in an ODS table might have been computed using in-database processing, after the statistics are computed, there is nothing for ODS to do in-database.

When ODS creates an output data set, ODS assumes that it is an ordinary SAS data set. You can ask ODS to create tables on a DBMS by assigning a libref to the table in a LIBNAME statement and then using that libref to declare a two-level name for the ODS table in an ODS OUTPUT statement. ODS then uses SAS/ACCESS software to write the tables. Occasionally, ODS might fail to write a DBMS table for the following reasons:

- Some ODS tables are created with SAS formats assigned to some variables. SAS formats cannot be stored in a DBMS table. However, SAS/ACCESS Interface to Teradata sets the Teradata data type based on the SAS format. If SAS/ACCESS software assigns a Teradata data type with insufficient range or precision, the ODS table might fail because of numerical overflow or loss of precision. To avoid these problems, use OPTIONS DBFMTIGNORE to tell the SAS/ACCESS software to create all numeric columns with the Teradata data type FLOAT, which is equivalent to the ANSI DOUBLE PRECISION type.

- In some cases, ODS attempts to update an ODS table that has already been created. SAS/ACCESS software does not allow Teradata tables to be updated or replaced.

## In-Database Computing for the OUT= Data Set by the CANCORR, FACTOR, PRINCOMP, and SCORE Procedures

Both the OUT= and the DATA= data sets must be Teradata tables in order for in-database computation to be used for the OUT= data set.

When the CANCORR, FACTOR, PRINCOMP, or SCORE procedure computes the OUT= data set in-database, SAS/ACCESS software submits an SQL command to create an empty DBMS table. Then the procedure submits an SQL command to insert data into the DBMS table. If there is a BY statement, a separate SQL command is issued for each BY group.

The SAS/STAT procedures connect to the DBMS using the credentials (user, password, and so on) in the LIBNAME statement for the DATA= data set. This connection is separate from any connection made by SAS/ACCESS software. If in-database computations are also performed for the OUT= data set, the DBMS user specified in the LIBNAME statement for the DATA= data set must have the INSERT privilege for the database where the OUT= data set resides.

When the OUT= data set is computed in-database, the procedure does not print the usual note about the number of observations and variables in the data set, because the procedure cannot determine precisely how many observations are in the OUT= data set. Instead, the procedure prints a note that states how many rows were inserted as reported by the SQL command. If OPTIONS SQL_IP_TRACE=ALL is specified, a separate note is generated for each BY group, in addition to a note for the total of all the BY groups. The numbers reported by SQL seem to be accurate, but it is difficult to tell whether they are always exact. One possible situation that might cause a discrepancy is other users of the database inserting or deleting rows at the same time the procedure is running so that the total number of rows that SQL inserts is not necessarily the actual number of rows in the table.

In-database computation cannot be used for the DATA= data set if both of the following conditions exist:

- There is a BY statement.
- The OUT= data set is specified but is not computed in-database.

# SAS/STAT Procedure Options, Statements, and Features Affected by In-Database Computing

## CANCORR Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.8 shows the options for the CANCORR procedure that are affected by in-database computing.

**Table 1.8** CANCORR Procedure Options Affected by In-Database Computing

| Option | Comment |
|---|---|
| DATA= | The SSCP matrix can be computed in-database. The SSCP matrix is then used to compute the correlation or covariance matrix on the client. |
| OUT= | Canonical variable scores can be computed in-database provided that in-database computation is also used for the DATA= data set. |
| OUTSTAT= | In-database computation is not applicable. |

The PARTIAL statement is not currently supported for in-database computing.

## FACTOR Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.9 shows the options for the FACTOR procedure that are affected by in-database computing.

**Table 1.9** FACTOR Procedure Options Affected by In-Database Computing

| Option | Comment |
|---|---|
| DATA= | The SSCP matrix can be computed in-database. The SSCP matrix is then used to compute the correlation or covariance matrix on the client. |
| OUT= | Estimated factor scores can be computed in-database provided that in-database computation is also used for the DATA= data set. |
| OUTSTAT= | In-database computation is not applicable. |

The PARTIAL statement is not currently supported for in-database computing.

## PRINCOMP Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.10 shows the options for the PRINCOMP procedure that are affected by in-database computing.

**Table 1.10** PRINCOMP Procedure Options Affected by In-Database Computing

| Option | Comment |
|---|---|
| DATA= | The SSCP matrix can be computed in-database. The SSCP matrix is then used to compute the correlation or covariance matrix on the client. |
| OUT= | Principal component scores can be computed in-database provided that in-database computation is also used for the DATA= data set. |
| OUTSTAT= | In-database computation is not applicable. |

## REG Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.11 shows the options for the REG procedure that are affected by in-database computing.

**Table 1.11** REG Procedure Options and Statements Affected by In-Database Computing

| Option | Comment |
|---|---|
| COVOUT | In-database computation is not applicable. |
| DATA= | The SSCP matrix can be computed in-database. |
| EDF | In-database computation is not applicable. |
| OUTEST= | In-database computation is not applicable. |
| OUTSEB | In-database computation is not applicable. |
| OUTSSCP= | In-database computation is not applicable. |
| OUTSTB | In-database computation is not applicable. |
| OUTVIF | In-database computation is not applicable. |
| PCOMIT | In-database computation is not applicable. |
| PLOTS= | This option requires row-level access and therefore cannot be used in-database. |
| PRESS | This option requires row-level access and therefore cannot be used in-database. |
| RIDGE | In-database computation is not applicable. |
| RSQUARE | In-database computation is not applicable. |
| TABLEOUT | In-database computation is not applicable. |

The following statements require row-level access and therefore cannot be used in-database:

- OUTPUT
- PAINT
- PLOT

- REWEIGHT

The following MODEL statement options require row-level access and therefore cannot be used in-database:

- ACOV
- CLI
- CLM
- INFLUENCE
- LACKFIT
- P
- PARTIAL
- PARTIALDATA
- PRESS
- R
- SPEC
- WHITE

The following MODEL statement options are never available with Teradata because they depend upon the row order of the data, which is not guaranteed to be consistent:

- DW
- DWPROB

The following PRINT statement options require row-level access and therefore cannot be used in-database:

- ACOV
- CLI
- CLM
- DW
- INFLUENCE
- MODELDATA
- P
- PARTIAL
- R
- SPEC

## SCORE Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.12 shows the options for the SCORE procedure that are affected by in-database computing.

**Table 1.12** SCORE Procedure Options Affected by In-Database Computing

| Option | Comment |
|---|---|
| DATA= | The DATA= data set can be used for computing the OUT= data set in-database. |
| OUT= | Scores can be computed in-database provided that the DATA= data set is a table in the same DBMS. |
| SCORE= | In-database computation is not applicable. |

When performing in-database computation, the SCORE procedure does not know the number of observations in the DATA= data set. If the data set contains zero observations, PROC SCORE does not print the usual note that says that there are no observations in the DATA= data set. Instead, SCORE prints a note that says that SQL inserted zero rows in the OUT= data set.

## VARCLUS Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.13 shows the options for the VARCLUS procedure that are affected by in-database computing.

**Table 1.13** VARCLUS Procedure Options Affected by In-Database Computing

| Option | Comment |
|---|---|
| DATA= | The SSCP matrix can be computed in-database. The SSCP matrix is then used to compute the correlation or covariance matrix on the client. |
| OUTSTAT= | In-database computation is not applicable. |
| OUTTREE= | In-database computation is not applicable. |

# SAS/ETS Procedure Options, Statements, and Features Affected by In-Database Computing

## TIMESERIES Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.14 shows the options for the TIMESERIES procedure that are affected by in-database computing.

**Table 1.14** TIMESERIES Procedure Options Affected by In-Database Computing

| Option | Comment |
|---|---|
| ACCUMULATE= | The time-stamped rows of the data table are accumulated to periodic time series in-database. The accumulated time series are then processed on the client. |

# Base SAS Procedure Options, Statements, and Features Affected by In-Database Computing

## CORR Procedure Options, Statements, and Features Affected by In-Database Computing

The ID statement requires row-level access and therefore cannot be used in-database.

Table 1.15 shows the options for the CORR procedure that are affected by in-database computing.

**Table 1.15** CORR Procedure Options, Statements, and Features Affected by In-Database Computing

| Option | Comment |
|---|---|
| EXCLNPWGT | If in-database computation is used, the EXCLNPWGT option is activated to exclude observations with nonpositive weights. |
| HOEFFDING | This option requires row-level access and therefore cannot be used in-database. |
| KENDALL | This option requires row-level access and therefore cannot be used in-database. |
| SPEARMAN | This option requires row-level access and therefore cannot be used in-database. |
| OUTH= | This option requires row-level access and therefore cannot be used in-database. |
| OUTK= | This option requires row-level access and therefore cannot be used in-database. |
| OUTS= | This option requires row-level access and therefore cannot be used in-database. |
| PLOTS | This option requires row-level access and therefore cannot be used in-database. |
| NOTSORTED | If you specify the NOTSORTED option in the BY statement, PROC CORR in-database computation ignores it and uses the default ASCENDING order for BY variables. |

## FREQ Procedure Options, Statements, and Features Affected by In-Database Computing

Table 1.16 shows the options for the FREQ procedure that are affected by in-database computing.

**Table 1.16** FREQ Procedure Options Affected by In-Database Computing

| Option | Comment |
|---|---|
| NOTSORTED | If you specify the NOTSORTED option in the BY statement, PROC FREQ in-database computation ignores it and uses the default ASCENDING order for BY variables. |
| ORDER=DATA | If you specify the ORDER=DATA option for input data in a DBMS table, PROC FREQ computation might produce different results for separate runs of the same analysis. In addition to determining the order of variable levels in crosstabulation table displays, the ORDER= option can also affect the values of many of the test statistics and measures that PROC FREQ computes. |

# SAS Enterprise Miner Procedure Options, Statements, and Features Affected by In-Database Computing

## DMDB Procedure Options, Statements, and Features Affected by In-Database Computing

PROC DMDB executes in-database with the following considerations:

- The number of VAR and CLASS terms cannot exceed the maximum number of columns in a Teradata table.
- If a WEIGHT statement is specified, then normal SAS computation is performed.
- For in-database computing, all interval variables are cast as float numbers.
- If a table exists in the database, it cannot be overwritten implicitly; it needs to be deleted explicitly.

Table 1.17 shows the options for the DMDB procedure that are affected by in-database computing.

**Table 1.17** DMDB Procedure Options, Statements, and Features Affected by In-Database Computing

| Option | Comment |
|---|---|
| MAXLEVEL= | This option is enforced on a per AMP (access module process) basis. |
| OUT= | SAS computation is used so that a DMDB data set can be created in SAS. |

## DMINE Procedure Options, Statements, and Features Affected by In-Database Computing

PROC DMINE executes in-database with the following considerations:

- R-square computation is supported. The in-database R-square computation uses either efficient SQL or UDF (user defined functions) depending on the number of required sufficient statistics (sums of squares and other sums). This number is computed by adding up all of the interval, class, class-interaction, and AOV16 effects. In particular, PROC DMINE uses the following formula to compute the number of these effects: TotalStat = 2*TotalClassLevels + 2*TotalClassInteractionLevels + TotalIntervalVars*35. If TotalStat is less than 2,030, then PROC DMINE uses SQL. Otherwise, PROC DMINE uses a UDF call to Teradata. These two approaches are used because Teradata allows a maximum of 2,048 columns in a table and the SQL approach is limited by this restriction.
- For stepwise forward variable selection, the **X'X** matrix is computed in-database using a UDF. In the current release, **X'X** is computed in-database only if the TotalStat value of effects that meet the minimum R-squared threshold (MINR2) is less than 2,800 and the sum of levels in all class effects that meet the MINR2 threshold is less than 500.
- Logistic regression for binary target variables is computed in-database by using an efficient SQL that performs one or two passes on the data depending on the number of effects in the model. In the current release, logistic regression is computed in-database only if the number of effects selected for logistic regression (after meeting MINR2 and STOPR2 thresholds) is less than 2,000.
- The status monitor is turned off when in-database computation is performed.

## DMREG Procedure Options, Statements, and Features Affected by In-Database Computing

PROC DMREG executes in-database with the following considerations:

- Linear and logistic models are supported. Other link functions in the MODEL statement are not supported.
- The CLASS statement is supported.
- The MODEL statement can contain only main effects and VAR interactions. Interactions composed of CLASS effects disable in-database computing.
- Model selection is supported using training (DATA=) statistics. Use of validation (VALIDATA=) or test (TESTDATA=) statistics is not supported in-database.
- The SCORE statement is not executed in-database. Data are transferred from the database to SAS.
- The default solution technique for logistic regression models is quasi-Newton. Methods that require Hessian or SSCP computations, such as Levenberg-Maquadt, are not computed in-database.

### Double Byte Character Sets

The in-database SAS Enterprise Miner procedures and macros can handle data sets in double byte character sets (DBCS) if SAS is configured to access double byte data sets from Teradata. Proper Teradata CLI (client level interface) configuration needs to be made on the client running SAS. Only DBCS data is supported; variable and table names must have single-byte encoding. See *SAS/ACCESS Interface to Teradata* for details.

## Miscellaneous Details

When SAS procedures perform out-of-database processing, SAS/ACCESS software runs SQL commands on Teradata in ANSI mode by default. When SAS/STAT, SAS/ETS, and SAS Enterprise Miner procedures perform in-database processing, the procedures run SQL commands on Teradata in Teradata mode. The difference in mode causes some differences in behavior as shown in Table 1.18.

**Table 1.18** Mode Differences for Data Insertion

| Procedures | ANSI Mode | Teradata Mode |
|---|---|---|
| Trying to insert a duplicate row when SET=YES | Teradata error and rollback | Duplicate row is dropped without error |
| Trying to insert a value that is too big for column | Teradata error and rollback | Value is truncated and inserted without error |

## Example

A simple case that illustrates the use of SAS in-database procedures is provided in the following statements:

1 Starting with data that is already in Teradata, the first step is to issue a LIBNAME statement that uses the Teradata SAS/ACCESS engine. If the Teradata LIBNAME engine has been correctly configured and SAS Analytics Accelerator 2.1 for Teradata has been installed on the Teradata system, then successful in-database computing is possible.

```
title1 "German Credit Data";
title2 "Teradata Pass-Through Example";
options ls=max nonumber nocenter;
ods html select default;
libname tera teradata server=davetd user=emdev password="test" database=emdev;
```

2 You can now examine the data by using standard SAS techniques. PROC CONTENTS reports only on the table information and does not download any rows of detail data. You might print the list of variables for continued use.

```
title3 'Variables List';
proc contents data=tera.dmagecr out=c noprint ;
run;

proc print data=c;
   var name type format;
run ;
```

3 Now that you have the list of variables, you can write the SAS statements for the analysis. The first use of SAS in-database procedures is in PROC FREQ and PROC MEANS. They are used to report summary measures that help you determine how to use these variables in the analysis. Both procedures are enabled for in-database processing. They dynamically generate SQL code that runs on Teradata and report only the results. As a SAS user, you have not modified any code to enable in-database computing.

```
title3 'Frequencies of Class Variables';
proc freq data=tera.dmagecr;
   table good_bad purpose;
run;

proc means data=tera.dmagecr ;
run;
```

4 You might continue the data exploration with a principal component analysis, which also runs in the database. Suppose you want to create plots. You should use the ODS SELECT statement to list only those plots that are compatible with in-database computing. (You can use the ODS TRACE option to find the graphics elements that are available for any procedure.) After the unmodified SAS statements run, the ODS graphics output is displayed in the SAS session.

```
ods select
   princomp.NObsNVar
   princomp.SimpleStatistics
   princomp.Corr
   princomp.EigenValues
   princomp.EigenVectors
   princomp.PrincompPatternPlot
   princomp.EigenvaluePlot;

proc princomp data=tera.tera.dmagecr outstat= work.pcastat;
   VAR age amount checking coapp depends duration employed existcr
      history housing installp job marital other property foreign
      resident savings telephon target;
run;
```

5 Next, save a copy of the output table in the SAS System because the output table contains several measures, such as the correlation matrix, which can be used for further analysis.

```
   title3 'Output from PCA';
   data corr;
      set pcastat(where=(_type_ eq "CORR"));
   run;


proc print data=corr noobs;
run;
```

**6** Now suppose you want to run a regression analysis. By looking at the output of these procedures, you can see that the dependent variable GOOD_BAD is a character variable. To use PROC REG, you must transform the GOOD_BAD character variable to a numeric variable. Use PROC SQL to push SAS code to the database for this operation. Pushing code to the database eliminates data movement. You need to use the explicit pass-through syntax to force processing in-database. For the column named Foreign, you must enclose the column name in quotation marks to prevent Foreign from being interpreted as a Teradata keyword. You must be careful when formatting your SQL code for in-database processing.

```
Proc SQL noerrorstop;
   connect using tera as teraConnect;
   execute (
   create view tera.tdview_reg as select
   age, amount, checking, coapp, depends, duration, employed, existcr,
   'foreign', history, housing, installp, job, marital, other,
   property, purpose, resident, savings, telephon,
   case when sas_dmnorm(good_bad) = sas_dmnorm('good') then 0 else 1 end as target
   from tera.dmagecr) by teraConnect;
   execute (commit) by teraConnect ;
quit;
```

**7** Finally, you build a regression model. Even though the conversion of a binary character dependent variable to a numeric variable for use in a linear regression model is not optimal, it is a common operation for using the efficient REG procedure for model selection and exploratory modeling steps. SAS output provides clues about how the factors are used in the model.

```
title3 'Linear Regression on raw data in teradata';
ods select SelectionSummary;
proc reg data=tera.tdview_reg;
   model target = age amount checking coapp depends
      duration employed existcr history housing installp
      job marital other property foreign resident savings
      telephon / selection = stepwise;
run;
quit;
```

**8** Suppose you want to run several model statements, selecting different combinations of variables and options. You can use the CORR matrix that you saved from the output of the PROC PRINCOMP step. PROC CORR and PROC REG can also produce this matrix as output. Because you saved the matrix data locally, you can execute any PROC REG step without accessing the raw data in Teradata.

```
title3 'Linear Regression on CORR from princomp';
ods trace on;
ods graphics on;
ods select SelectionSummary;

proc reg data= pcastat(type=corr) outset= est;
   model target = age amount marital other property
                  foreign resident savings telephon;
run;
quit;

ods graphics off trace off;
ods html close;
```

**9** Now suppose that you want to apply the model to a data set that contains new independent variable values to produce a new table that contains predicted values. This process is called scoring. The OUTPUT statement in PROC REG does not support in-database computing. However, the SCORE procedure does support in-database computing, and produces scores for a variety of models. You use the OUTEST data set that you saved locally in the PROC REG step to parameterize the model. PROC SCORE generates and pushes the appropriate SQL code to the database. Both the input and output tables are in Teradata, and no data are transferred to SAS.

```
proc score data= tera.newdata out= tera.scores score=est type=parms;
   var age amount marital other property foreign resident savings telephon;
run;
```

## Example: Using the %EM_INDBIN Macro

The following SAS statements demonstrate how to bin all of the interval variables in the Tera.Dmagecr table, output the binned variables, generate the binning code, and create a mapping information table. Each interval variable is grouped into five bins with each bin containing approximately 20 percent of the observations.

**1** First, you load the %EM_INDBIN macro:

```
%indtdaam;
```

**2** Next, you delete the output table if it exists:

```
proc sql;
```

```
      drop table tera.BinOut1;
quit;
```

**3** Name the location where you want to save your binning code:

```
filename _X catalog 'Work.Bin.BinningCode.source';
```

**4** Now, call the %EM_INDBIN macro:

The OUT= option creates a Teradata table that contains the binned variables. The BINNINGCODE= option creates a SAS file that contains the SQL code that is used to bin the variables. The BINMAPPINGS= option creates a mapping information table. The EVENTVAL= option defines the level of event for the target variable.The OPTIONS=WOEADJUST option creates the weight-of-evidence statistic for each bin.

```
title "Calling In-Database Binning Macro";

%em_indbin(DATA=Tera.Dmagecr,
           OUT=Tera.BinOut1,
           BINNINGCODE=_X,
           BINMAPPING=BinMapping,
           NUMBINS=5,
           TARGETVAR=good_bad,
           EVENTVAL=%str(bad),
           OPTIONS=WOEADJUST);
```

# Subject Index

# Syntax Index

# Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **`yourturn@sas.com`**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **`suggest@sas.com`**.