Paper 337-2009

In-Database Procedures with Teradata: How They Work and What They Buy You

David Shamlin and David Duling, SAS Institute, Cary, NC

ABSTRACT

SAS® applications are often built to work with large volumes of data in environments that demand rigorous IT security and management. When the data is stored in an external database, such as Teradata, the transfer of such large data sets to the computers that run SAS can cause a performance bottleneck and possible unwanted security and resource management consequences for local data storage. SAS® In-Database processing helps address these challenges by moving computational tasks closer to the data and by improving the integration between SAS and the database management system (DBMS). This paper examines opportunities for applying in-database techniques to SAS procedures for use in Teradata enterprise data warehouse environments.

Certain SAS procedures are being modified to leverage Teradata SQL functionality and, in some cases, to leverage SAS functions that are being deployed in the DBMS. In this paper, we review selected enhancements that are being made to SAS procedure statements in Base SAS®, SAS/STAT®, and SAS® Enterprise Miner™, and how these procedures affect the distribution of work between SAS and Teradata. A summary of the performance characteristics of these enhanced procedures is also discussed.

INTRODUCTION

Data lives in many forms—from flat text files, to SAS data sets, to highly secure relational databases. SAS users have long exploited relational database systems for exploration, analysis, and reporting. The division of data between data storage and data analysis systems is becoming increasing untenable at many institutions because of the following factors:

Data Movement

When data in a relational database system is accessed, resources are spent to transfer the data, to store the data in permanent or temporary SAS data sets, and to wait in real time for operations to execute. These operations yield an inefficiency that is negligible for small data sets, but can be very significant for large data sets. As the data that is managed by database systems increases, the effect of this inefficiency becomes more important to SAS users.

Security

SAS users have increased concerns over distribution of data that contains personal identify information. Many common data mining tasks require access to source data that contains this personal identity information before the data has been thoroughly cleaned and transformed. Examples of data include financial billing and receipt transactions, insurance claims, and medical treatment records. In these situations, institutions face increased scrutiny over security practices, which cause them to limit the transfer and duplication of this data.

Governance

Beyond security issues, corporate regulations often mandate controls over the distribution of data. Some regulations are mandated by governments (for example, Sarbanes-Oxley) or consortiums (for example, Basel II). In either case, a primary motivation is to ensure that corporate decisions are made using transparent and approved data and analytical practices. Even though centralized data management is often perceived to be a part of these regulated practices, the use of multiple redundant data stores represents data integrity risks.

Resources

Modern database systems often contain a large number of powerful processors, optimized disks, and fast network connections, along with highly developed software that optimizes query processing and manages security, users, queues, and priorities. Because organizations have invested large sums of money in the acquisition and maintenance of these systems, the organizations need to leverage their resources as much as possible.

To address these needs, SAS developers are working on methods to minimize data movement, while retaining the use of familiar SAS applications. SAS users can continue to use the SAS server to edit and run SAS programs.

WHO BENEFITS FROM SAS IN-DATABASE PROCEDURES

At SAS Global Forum 2008, an application specifically targeted at database users was previewed. SAS Formats for Teradata enables SQL expressions and where clauses that contain PUT statements to be passed through to the database. The pass-through ability can greatly reduce data movement in routine data building programs. Another enhanced product, SAS® Scoring Accelerator for Teradata, enables you to execute data mining model score code that is generated by SAS Enterprise Miner as native functions in the Teradata system. This code represents models that are more complex than models that can be run in pure SQL or when using the Predictive Model Markup Language (PMML). Scoring processes typically run more frequently and on larger quantities of data than do model building processes. Scoring processes must also run in production data environments.

In addition, other SAS products have functionality database users need. In Base SAS and SAS/STAT software. several SAS procedures have been updated 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. Teradata systems are a highly optimized mix of hardware and software and are used to store some of the largest and most secure data systems worldwide.

SAS users who work with Teradata data stores and who write SAS programs that use basic summarization and statistical functions will benefit from the updated procedures.

TERADATA AND SAS

In a conventional environment, 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 and perform functions in the SAS system. This movement of data causes a performance penalty unless the equivalent functions could have been executed in the database fast enough to compensate for the time that was used to move the data.

The Access Module Processor (AMP) is the heart of the Teradata database. The AMP is a virtual process that controls the management of the disk subsystem to which it is assigned. Basically an AMP represents a single unit of parallelism within the Teradata environment. A query optimizer distributes the workload to the AMPs. The system implements a shared-nothing design in which little to no communication occurs between the AMPs. Each function that is distributed to the AMP must use only the data that is stored on the disks that are connected to that AMP. Performance improves when data and queries can be structured so that most of the processing can execute on the shared-nothing AMPs.

The trade-off in processing time versus data movement time depends on several dynamic factors, which include client and server workloads, network speed and workload, and data size. The benefit from using SAS In-Database procedures varies based on the particular environment. In general, the benefit increases as the number of rows increases.

SAS users have become proficient with the SQL procedure to write code that accesses data and to develop queries that explore and subset the data and that minimize data movement between systems. When you use SAS/ACCESS Interface to Teradata, you are using PROC SQL to pass SQL queries from a SAS process to Teradata. The SQL procedure can pass a query directly to the database; this is known as SQL pass-through. SQL procedure statements can be written to use native Teradata SQL, which is always passed to Teradata as is. This pass-through behavior is called explicit pass-through. SAS programmers often use explicit pass-through in SAS programs that are intended for use with only a specific database system, such as Teradata.

In other instances, a SAS program must be database-agnostic, in which case SAS SQL syntax is used. When possible, the SQL procedure automatically translates SAS SQL syntax into Teradata syntax that can be passed to the database. This pass-through behavior is called implicit pass-through. One of the primary design goals of the SQL procedure is to optimize the use of implicit pass-through.

The SQL procedure plays a key role in in-database processing. Basically, when a SAS procedure is enabled to run in the database, the procedure has been taught to generate custom SQL queries that represent part of the work to be done. After generating a SQL query, the in-database procedure passes control to the SQL procedure, which submits the query to Teradata. After the query executes in Teradata, the in-database procedure retrieves the results using the SAS/ACCESS Interface to Teradata software. The in-database procedure then performs any additional processing that is required to complete the PROC SQL step. The specific nature of the generated SQL queries varies by in-database procedure and how the SQL queries are invoked at any given time. This paper includes some specific in-database procedure examples.

CLASSES OF PROCEDURES

The following primary design patterns are used to develop in-database procedures. They are both based on the central concept of submitting SQL code to the database.

SQL Generation

The SAS 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 for a particular database, which 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. This is exactly the same end result if the user was using Base SAS tables. But, in this case, the relational database software is responsible for optimizing and executing the query. When the data to be analyzed is very large, this approach can result in significantly lower total processing lapse time and in reduced data movement between SAS and the database. The SQL generation pattern is shown in Figure 1.

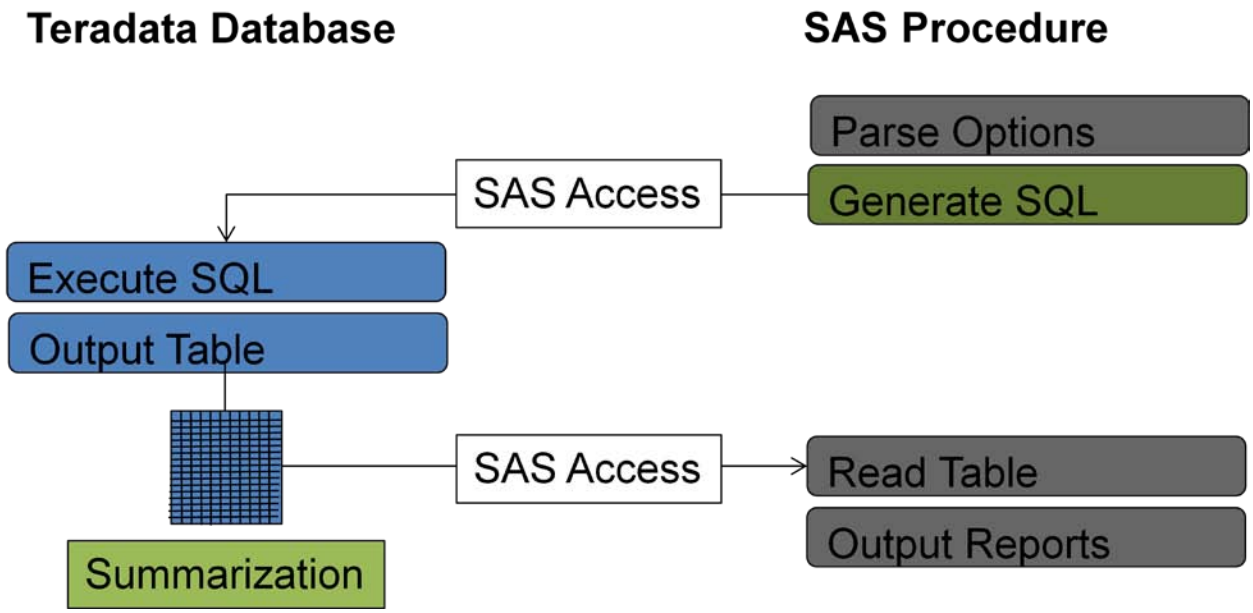

Figure 1. SAS Procedure Creates and Submits SQL Code to the Database

SAS In-Database Functions

In the second design pattern, the SAS procedure dynamically creates and submits SQL code. However, this code includes references to a new database function that was developed by SAS and is installed on the Teradata system. This new SAS In-Database function meets the standards of performance and accuracy that users expect from SAS software, including the handling of missing values and formatted values. The in-database function meets the reliability expectations of Teradata users. A good example of an algorithm implemented as a SAS In-Database function is the creation of a sum of squares and cross products (SSCP) matrix. The SSCP matrix is a condensed representation of the relationships between variables. It is a fundamental building block for many statistical procedures that are used in SAS products. 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. A data set that contains 100 numeric columns and 5 million rows of data will produce an SSCP matrix that contains, nominally, 100x100 cells. However, we need only the lower triangle at 100x50 cells. In this example, the data that was transferred from Teradata to SAS was reduced from 500,000,000 raw data cells to 5,000 SSCP data cells, which is a reduction of 1000 percent! As you can see, the benefits increase greatly as the number of rows increases. Large data applications benefit the most from in-database processing.

In this pattern, the SAS procedure constructs SQL code that uses the SAS In-Database function that creates the SSCP matrix. This function is not intended for general-purpose SQL queries, but it is useful for a SAS procedure. The elements of the SSCP are created and transferred to SAS. After the cross products have been transferred to SAS, the database is available to continue processing queries while the SAS procedure continues the analysis. An example of the use of in-database processing in PROC PRINCOMP in SAS/STAT is shown Figure 2:

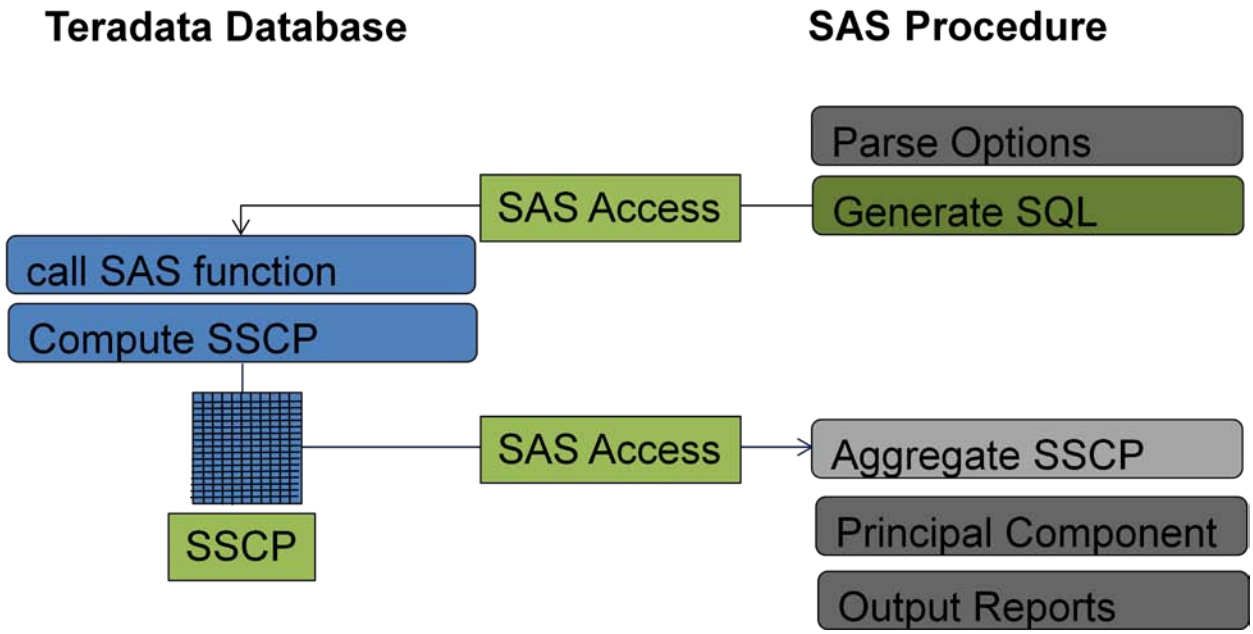

Figure 2. SAS Procedure Submits the SQL Code that Calls a SAS Function in the Database

The use of in-database processing is similar for any SAS procedure that relates to SSCP. The one exception is the substitution of the regression function for the principal components.

BASE SAS PROCEDURES

The R&D team working on SAS and Teradata is currently working on in-database implementations in the FREQ procedure, the SUMMARY procedure, the MEANS procedure, and the RANK procedure. These procedures are commonly used in reporting and data discovery applications. The in-database enhancements to these procedures apply the SQL generation design pattern to better leverage the processing power of Teradata and reduce data movement. More specifically, when in-database capabilities are used, they dynamically generate standard SQL queries that leverage intrinsic Teradata SQL descriptive statistical functions to summarize the data in the database before the procedure returns any information to the calling SAS process.

It is generally known that SAS procedure steps can directly or indirectly reference SAS formats. When this is the case, the referenced formats can be deployed inside Teradata to promote in-database processing. The SAS 9.2 release of SAS/ACCESS Interface to Teradata includes a run-time library of SAS formats that can be installed on your Teradata system. It includes tools to publish PROC FORMAT formats to your database system. The details of how to use these tools exceeds the scope of this paper. For information about deploying and using SAS formats in Teradata, see the SAS 9.2 documentation.

THE FREQ Procedure

The FREQ procedure produces one-way to *n*-way frequency and contingency (crosstabulation) tables. For two-way tables, PROC FREQ computes tests and measures of associations. For *n*-way tables, PROC FREQ provides stratified analysis by computing statistics across, as well as within, strata. When PROC FREQ runs in the database with Teradata, an SQL SELECT statement is generated that calculates the count of each distinct value of each classification variable specified. One global GROUP BY is added to the SELECT statement to cover all crossings in all of the specified FREQ TABLE statements.

The WEIGHT and WHERE statements also have an effect on the generated SQL statement. A WEIGHT statement is converted to a SQL SUM function reference, and a WHERE statement is included in the SQL statement. The result set that is created when the database executes the SELECT statement is converted to a data structure that is internal to SAS. After the data structure is built, statistical processing and other result processing continues as usual.

In-database processing with PROC FREQ reduces a single step's real-time execution because the initial summarization of the data can occur in parallel across all rows in the table in the Teradata warehouse. In cases where the cardinalities of the used classification variables are low relative to the total number of rows in the input table, the amount of data that is transferred from Teradata to SAS is also significantly reduced.

THE SUMMARY Procedure and the MEANS Procedure

The SUMMARY procedure and the MEANS procedure provide data summarization tools to compute descriptive statistics for variables across all rows and within groups of rows in the given input table. Similar to PROC FREQ, PROC SUMMARY and PROC MEANS generate an SQL query that is based on the statements that are used and the output statistics that are specified in the PROC step. If class variables are specified, the procedure creates a SQL GROUP BY clause that represents the *n*-way type. The result set that is created when the aggregation query executes in Teradata is read by SAS into the internal PROC SUMMARY or PROC MEANS data structure and all subsequent types are merged into the final analysis results. When SAS format definitions have been deployed in the Teradata database, then formatting of class variables occurs in Teradata. When this is not the case, the relevant formats are applied by SAS. Multi-label formatting is always done by SAS using the initially aggregated result set that is returned by Teradata. Support for the CLASS, TYPES, WAYS, VAR, BY, FORMAT, and WHERE statements are included in PROC SUMMARY and PROC MEANS in-database capabilities. Some PROC SUMMARY and PROC MEANS features are not yet supported, such as quintile statistics (MEDIAN, MODE, P25, P75), SKEWNESS, KURTOSIS, and the ID statement.

THE RANK PROCEDURE

The RANK procedure computes ranks for one or more numeric variables across the rows of a table. It sends the ranks to a new output table. Like PROC FREQ, PROC SUMMARY, PROC MEANS, PROC RANK generates SQL queries when it runs in the Teradata database. Computing ranks in Teradata is facilitated by ordered analytic functions that are available in Teradata SQL. In the first release of the PROC RANK implementation of in-database processing, all ranking methods are included except NORMAL. The structure of the SQL query generated during an in-database invocation of PROC RANK depends on several factors, including the ranking methods that are used, the number of variables that are ranked, the inclusion of BY and WHERE statements, and the PROC RANK options that are used, such as TIES= and DESCENDING. The generated SQL queries can be complex and can involve multiple nested JOIN operations. When the SAS library that is specified in the IN= option refers to the same database as in the OUT= option, no data is transferred from Teradata to SAS. Although the mathematical operations and the order in which they are performed are essentially the same for the SQL query that is executed in Teradata and for the PROC RANK code that is executed in SAS, small numerical differences in results are possible.

Using in-database processing with only these three procedures can provide customers with immediate benefits. In-database processing can add value to any organization that uses SAS with Teradata by reducing the run-time speeds of SAS applications, by minimizing data movement between the two environments, and by increasing the utilization of Teradata resources.

ANALYTICS PROCEDURES

Developers in the advanced analytics division are working on in-database versions of the PRINCOMP procedure, the VARCLUS procedure, the REG procedure, and the SCORE procedure. 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 processing is identical to the syntax and output for conventional SAS processing in most cases. In those cases, existing procedure steps in SAS programs will be able to run in-database without modification. Together with the in-database BASE procedures, they 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 database and SAS systems.

THE PRINCOMP Procedure

The PRINCOMP procedure computes a principal components transformation of the SSCP matrix into orthogonal components. The first component or dimension 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) is used to reduce 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 the database and is transferred to SAS for further processing. All options in the PRINCOMP procedure are supported for in-database processing, except the OUTPUT statement. You must use the SCORE procedure for in-database scoring of principle components.

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 analysis. The SSCP matrix is computed in the database, and is then transferred to SAS for further processing. All options in the VARCLUS procedure support in-database processing.

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. The REG procedure 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 exploratory analysis, in addition to its extensive model fitting and reporting functions. You might want to use the REG procedure to model compute candidate models in the database. You would do this before you extract a sample of data that contains only the final set of variables and a subset of rows to SAS for complete regression analysis. A full SSCP matrix is computed in Teradata and is transferred to SAS, in which the model fitting occurs. The REG procedure includes an option that uses a CORR or SSCP matrix as input, and that 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. This includes the general class of residual analysis, plot options, and confidence intervals.

THE SCORE PROCEDURE

The SCORE procedure uses the estimates table that is produced by the PRINCOMP procedure or the EG procedure to do the following:

- apply the model to an input data set
- produce an output data set with new columns that contains either the principal component values or the predicted and residual values

You can use the SCORE procedure to produce output in the form of a table in Teradata, which contains the score columns that are produced by the PRINCOMP procedure or the REG procedure. The SCORE procedure dynamically generates SQL code for the given model, and then submits the SQL code to the database, which produces a native Teradata table without having to extract any rows of data into SAS. All options in the SCORE procedure support in-database processing.

LIMITATIONS

Some features of SAS procedures do not execute in the database. The ability to perform in-database processing depends on the specific type of analysis that is provided by the statements, and the options that are used in a PROC step. The SAS documentation identifies the SAS procedures that support in-database processing. These limitations are considered reasonable for the use case of 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 the 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.

ENABLING IN-DATABASE PROCESSING

You can control in-database processing by using system options and LIBNAME statement options. The SQLGENERATION system option is used to control in-database processing. The supported values are NONE, DMBSMUST, and DBMS (the default), which 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 PROC REG step that contains a residual option computes the SSCP matrix in the database, but transfers the detail data to SAS to create residual analysis and output.
- When the value is DBMSMUST, only functions that support in-database processing are processed. If a function that does not support in-database processing (such as residual analysis) requests in-database

processing, a warning message is printed. Use this value only when you want a PROC step to completely fail when in-database processing is not supported.

When in-database processing technology is discussed with customers, questions about the impact on database workload are frequently asked. Using SAS In-Database processing with Teradata does have an impact on workload. The extent of that impact varies, based on the SAS procedure and how the procedure is used. However, because SAS In-Database procedures use design patterns that are based on SQL query execution and custom SAS functions that are invoked as part of the generated queries, you can monitor and manage the impact on your Teradata system by using standard workload management tools and best practices. Using such tools and the SQLGENERATION option provides a high degree of flexibility when tuning both your SAS and Teradata systems for in-database computing.

USE CASE AND EXAMPLE

A simple case illustrating the use of SAS In-Database procedures is provided in the following code. Starting with data that is already in Teradata, we first want to set up the LIBNAME statement that uses the Teradata SAS/ACCESS engine. If the Teradata LIBNAME engine has been correctly configured and the SAS Analytics Accelerator has been installed on the Teradata system, then successful in-database processing is possible.

```
/*--- credit data example ---*/
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;
```

Next, we want to 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. We print the list of variables for continued use.

```
/*--- Examine the data ---*/
title3 'Variables List' ;
proc contents data=tera.dmagecr out=c noprint ; run;
proc print data=c ; var name type format ; run ;
```

Now that we have the list of variables, we can write the SAS code steps for our 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 us to 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 on the results. As SAS users, we have not modified any code to enable in-database processing.

```
/*---- Get summary information ---*/
title3 'Frequencies of Class Variables' ;
proc freq data=tera.dmagecr  ;
table good_bad purpose ;
run ;
proc means data=tera.dmagecr ;
run ;
```

We continue the data exploration with a principle components analysis, which also runs in the database. We want to create plots, but we should use the ODS SELECT statement to list only those plots that are compatible with in-database processing. You can use the ODS TRACE option to find the graphics elements that are available for any procedure. After the unmodified SAS code runs, the ODS graphics output is displayed in the SAS session.

```
/*--- Principal Components Analysis ---*/
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 ;
```

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

```
/*--- save the Correlation matrix ---*/
title3 'Output from PCA';
data corr ; set pcastat(where=(_type_ eq "CORR")) ; run ;
proc print data=corr noobs ; run ;
```

Next, we want to run a regression analysis. However, by looking at the output of these procedures, we can see that the dependent variable GOOD_BAD is a character variable. To use PROC REG, we must transform the GOOD_BAD character variable to a numeric variable. We use PROC SQL to push SAS code to the database for this operation. Pushing code to the database eliminates data movement. We need to use the explicit pass-through syntax to force processing in the database. For the column named Foreign, we 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.

```
/*--- Transform the dependent variable ----*/
%let connopt=server=davetd  user=emdev pw='test' database=emdev ;
%let dbms=teradata;
Proc SQL noerrorstop;
     connect to &dbms. (&connopt.);
     execute ( ;
     create view tera.tdview_reg  as select
      age,  amount,    checking,    coapp,    depends,    duration,   employed,   existcr,
   'foreign'n,   history,   housing,   installp,   job,   marital,    other,
   property,   purpose,   resident,   savings,   telephon,
   case when  good_bad = 'good' then 0 else 1 end as target
    from tera.dmagecr ;
 ) by &dbms;
   execute (commit) by &dbms. ;
run ;
```

Finally, we 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 to use the efficient REG procedure for model selection and exploratory modeling steps. SAS output provides clues about how the factors are used in the model.

```
/*--- build regression 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 ;
```

Now, what if we wanted to run several model statements, selecting different combinations of variables and options? We can use the CORR matrix that we saved from the output of the PROC PRINCOMP step. PROC CORR and PROC REG can also produce this matrix as output. Because we saved the matrix data locally, we can execute any PROC REG step without accessing the raw data in Teradata.

```
/*--- run another reg step ---*
/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 ;
```

Finally, we 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 processing. However, the SCORE procedure does support in-database processing, and produces scores for a variety of models. We use the OUTEST data set that we 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 is transferred to SAS.

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

CONCLUSION

This paper introduced in-database procedure concepts and discussed the specific procedures that are being enhanced. The release of the enhanced SAS procedures that support in-database processing use the Teradata database system. These SAS procedures include PROC FREQ, PROC SUMMARY, PROC MEANS, PROC RANK, PROC PRINCOMP, PROC CORR, PROC VARCLUS, PROC REG, and PROC SCORE. The paper explained the key design patterns in in-database procedure processing, gave a general overview of how and when each procedure uses these design patterns, and explored their behaviors in detail through a use case example.

SAS In-Database processing with SAS procedures is a significant enhancement because of its ability to integrate with Teradata enterprise data warehouses. In-database procedures are a powerful and easy way to leverage more of Teradata's processing power by using existing SAS programming syntax. In-database procedures reduce data movement between SAS and Teradata environments and accelerate performance. Both of these benefits add real and immediate value to any organization's use of SAS and Teradata.

ACKNOWLEDGMENTS

The authors extend their heartfelt gratitude to the following individuals at SAS:

Robert Ray, Larry Dowty, Ashok Savasere, Mike Whitcher, Scott Mebust, Georges Guirguis, Warren Sarle, Randy Tobias, Robert Cohen, Pravin Boniface, Austin Swift, and Pat Buckley

Teradata:

Michael Watzke and Greg Otto

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

David Duling
R&D Director
SAS Institute Inc.
SAS Campus Dr.
Cary, NC 27513
E-mail: david.duling@sas.com

David Shamlin
Senior R&D Director
SAS Institute Inc.
SAS Campus Dr.
Cary, NC 27513
E-mail: david.shamlin@sas.com