THE POWER TO KNOW.

SAS® 9.4 Data Quality Accelerator for Teradata: User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® 9.4 Data Quality Accelerator for Teradata: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® 9.4 Data Quality Accelerator for Teradata: User's Guide

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

November 2016

9.4-P1:dqaccltdug

Contents

What’s New in SAS 9.4 Data Quality Accelerator for Teradata

Overview

SAS 9.4 Data Quality Accelerator for Teradata contains general fixes and enhancements to in-database data quality operations. Additionally, the release numbering of the SAS Data Quality Accelerator for Teradata product has been changed from SAS Data Quality Accelerator 2.7 for Teradata to SAS 9.4 Data Quality Accelerator for Teradata. The new numbering is a result of the company’s recent integration of DataFlux Data Management Studio into the SAS suite of data quality, data integration, data governance, and master data management solutions. In keeping with this change, the *SAS Data Quality Accelerator 2.7 for Teradata: User’s Guide* has been retitled as *SAS 9.4 Data Quality Accelerator for Teradata: User’s Guide*.

Chapter 1

Introduction to SAS Data Quality Accelerator for Teradata

Introduction to the SAS Data Quality Accelerator for Teradata

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 a Teradata database, the transfer of large data sets to the computers that run the SAS System can cause a performance bottleneck. There are also possible unwanted security and resource management consequences for local data storage. SAS Data Quality Accelerator 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).

SAS Data Quality Accelerator for Teradata provides in-database data quality operations. The data quality operations are provided as Teradata stored procedures. A stored procedure is a subroutine that is stored in the database and is available to applications that access a relational database.

The stored procedures perform the following data quality operations:

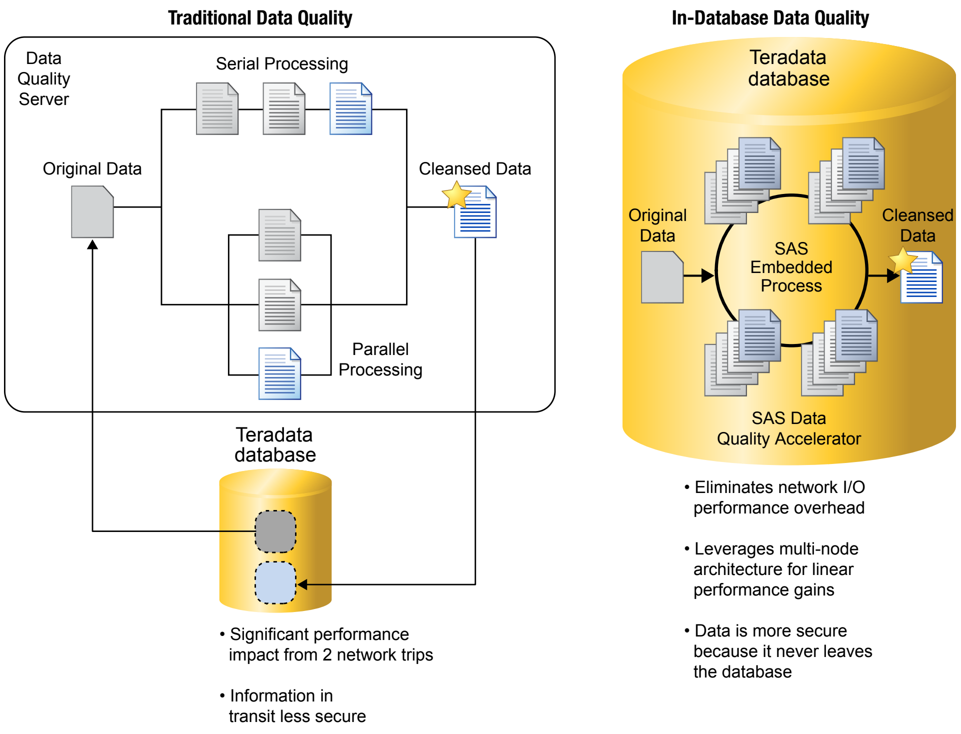
- upper-casing, lower-casing, and proper-casing
- attribute extraction
- gender analysis
- identification analysis
- matchcode generation
- parsing

- pattern analysis
- standardization

Benefits of In-Database Processing

The following diagram illustrates the difference between in-database and outside-of-database data quality operations.

Evolution of Data Quality


Traditional Data Quality
Data Quality Server
Serial Processing
Original Data
Cleansed Data
Parallel Processing
Teradata database
• Significant performance impact from 2 network trips
• Information in transit less secure
In-Database Data Quality
Teradata database
Original Data
SAS Embedded Process
Cleansed Data
SAS Data Quality Accelerator
• Eliminates network I/O performance overhead
• Leverages multi-node architecture for linear performance gains
• Data is more secure because it never leaves the database

Executing data quality operations inside of the database rather than as a separate utility outside of the database provides the following benefits:

- eliminates network I/O performance
- leverages multi-node architectures for linear performance gains
- makes information more secure because it never leaves the database

Benefits of Teradata Stored Procedure Technology

The availability of the in-database data quality operations as Teradata stored procedures offers the following benefits:

- Because they are self-contained, the data quality stored procedures provide a good way to enforce standards for commonly used processes and avoid same-code proliferation.
- Programmatically, the data quality stored procedures can be nested in the definition of other stored procedures.
- The data quality stored procedures can be run anywhere a Teradata stored procedure can be run. They can be executed interactively by a user with a command-line interface such as the Teradata BTEQ utility. Or they can be executed in a client program by any language that supports a Teradata connection or a generic ODBC connection. Examples include the Base SAS SQL procedure; the Teradata Tools and Utilities (TTTU); scripting languages, such as Perl or Python; and third-party ETL tools.

Comparison to SAS Data Quality Server Functions

The data quality stored procedures provide similar data quality operations to those available with the following SAS Data Quality Server functions.

Table 1.1 *Data Quality Stored Procedures and Comparable SAS Data Quality Server Functions*

Stored Procedure Name	Function Name	Description
DQ_EXTRACT()	DQEXTRACT()	Extracts specific entities or attributes from a text string.
DQ_GENDER()	DQGENDER()	Determines the gender of a person from their name or other information. The input value is a text string.
DQ_GENDER _PARSED()	DQGENDERPARSED()	Determines the gender of a person from their name or other information. The input consists of one or more parsed character values.
DQ_IDENTIFY()	DQIDENTIFY()	Determines the type of data that is represented by a text string.
DQ_LOWERCASE()	DQCASE()	Lowercases text.
DQ_MATCH()	DQMATCH()	Generates a matchcode for a text string.
DQ_MATCH _PARSED()	DQMATCHPARSED	Generates a matchcode for one or more parsed character values.
DQ_PARSE()	DQPARSE() DQPARSETOKENGET()	Segments a string into semantically atomic tokens.

Stored Procedure Name	Function Name	Description
DQ_PATTERN()	DQPATTERN()	Returns a simple representation of a text string's character pattern.
DQ_PROPERCASE()	DQCASE()	Applies uppercase and lowercase lettering using context-sensitive rules.
DQ_STANDARDIZE()	DQSTANDARDIZE()	Generates a preferred standard representation of a string.
DQ_STANDARDIZE _PARSED()	Not applicable	Generates a preferred standard representation from one or more parsed values.
DQ_UPPERCASE()	DQCASE()	Uppercases text.
DQ_SET_LOCALE()	Not applicable (the locale is set by other means)	Sets the locale for data quality operations in the SQL session.
DQ_LIST_LOCALES()	DQLOCALEINFOGET (lists the locales that are currently available in memory)	Lists all of the locales in the current SAS Quality Knowledge Base.
DQ_LIST_DEFNS()	DQLOCALEINFOLIST()	Lists the SAS Quality Knowledge Base definitions that are available for the specified operation and locale.
DQ_LIST_TOKENS()	Not applicable	Lists the input tokens that are available for a specified operation, Quality Knowledge Base definition, and locale.

The data quality stored procedures and SAS Data Quality Server functions differ in the following ways:

- SAS Data Quality Server functions operate outside of the database.
- SAS Data Quality Server functions operate on one data line at a time. The data quality stored procedures operate on the contents of an entire column of an input table.
- SAS Data Quality Server functions are applied dynamically. The data quality stored procedures can return output in a physical table or dynamically, as a *cursor*, which is a control structure that enables traversal over the records in a result set.
- A SAS language is required to use the SAS Data Quality Server functions. The data quality stored procedures can be run by any language that supports the Teradata SQL dialect.

Components of the Accelerator

SAS Data Quality Accelerator for Teradata consists of the following components:

1. SAS Embedded Process for Teradata. The SAS Embedded Process is a SAS server process that runs within Teradata to read and write data. Currently, the SAS Embedded Process for Teradata can be installed only on a Teradata Linux server.
2. SAS Quality Knowledge Base (QKB), a collection of data quality rules provided as part of the SAS Data Quality software offering. The QKB comes equipped with a standard collection of data quality rules, which you can modify for your needs in DataFlux Data Management Studio. The QKB can originate from a Windows or UNIX system.
3. Tools for packaging and deploying the QKB and SAS Embedded Process on the Teradata nodes.
4. Tools for creating and administering the data quality stored procedures in the Teradata database.
5. The data quality stored procedures.

The components require post-installation configuration in the Teradata database before the data quality stored procedures can be used. These steps must be performed by the Teradata database administrator. Instructions for performing these steps are provided in the “SAS Data Quality Accelerator for Teradata” chapter of the “Administrator’s Guide for Teradata” in the *SAS In-Database Products: Administrator’s Guide*.

The *SAS In-Database Products: Administrator’s Guide* also contains information about performing administrative tasks such as granting users authorization to run the stored procedures and upgrading the QKB.

About This Document

This document describes how to use the data quality stored procedures to improve data quality at your site.

Audience

The audience for this product is IT organizations that want to improve data quality so that their programs run on clean data.

Chapter 2

Overview of Data Quality Operations

Overview

This section illustrates the data quality operations that can be performed with the data quality stored procedures.

Casing

Casing applies context-sensitive case rules to text. It operates on character content, such as names, organizations, and addresses. Casing is applied with the DQ_LOWERCASE(), DQ_PROPERCASE(), and DQ_UPPERCASE() stored procedures.

Table 2.1 *Examples of Case Stored Procedure Inputs and Outputs*

Input	Procedure	Output
SAS INSTITUTE	DQ_LOWERCASE()	sas institute
	DQ_UPPERCASE()	SAS INSTITUTE
	DQ_PROPERCASE()	SAS Institute

Extraction

Extraction returns one or more extracted text values, or tokens, as output. Extraction is performed with the DQ_EXTRACT() stored procedure.

Table 2.2 *Example of Extraction*

Input	Output	
Blue men’s long-sleeved button-down collar denim shirt	Color:	Blue
	Material:	Denim
	Item:	Shirt

Gender Analysis

Gender analysis evaluates the name or other information about an individual to determine the gender of that individual. If the evaluation finds substantial clues that indicate gender, the function returns a value that indicates that the gender is female or male. If the evaluation is inconclusive, the stored procedure returns a value that indicates that the gender is unknown. The exact return value is determined by the specified gender analysis definition and locale.

Two stored procedures are provided to perform gender analysis. The DQ_GENDER() stored procedure parses and analyzes an input text string in order to return a gender value.

Table 2.3 *Example of Gender Analysis of a Text String*

Input	Output
Jane Smith	F
Sam Adams	M
P. Jones	U

The DQ_GENDER_PARSED() stored procedure analyzes pre-parsed values and returns a gender value based on the concatenated values.

Table 2.4 *Example of Gender Analysis on Pre-parsed Values*

Input Value 1	Input Value 2	Output
Jane	Smith	F
Sam	Adams	M
P.	Jones	U

Identification Analysis

Identification analysis returns a value that indicates the category of the content in an input character string. The available categories and return values depend on your choice of identification definition and locale. Identification analysis is performed with the DQ_IDENTIFY() stored procedure.

Table 2.5 *Example of Identification Analysis*

Input	Output
John Smith	NAME
SAS Institute	ORGANIZATION

Matching

Matching analyzes the input data and generates a matchcode for the data. The matchcode represents a condensed version of the character value. Similar strings get identical matchcodes. You can specify a sensitivity value that indicates the degree of similarity that should be applied to consider something a match. For higher sensitivities, two values must be very similar to produce the same matchcode. At lower sensitivities, two values might produce the same matchcode despite considerable dissimilarities. Records can be clustered by sorting by matchcodes. Fuzzy lookups can be performed via matchcode searches.

Two stored procedures are provided to perform matching. The DQ_MATCH() stored procedure parses and analyzes an input text string and generates a matchcode for the string.

Table 2.6 *Example of Matching Text Strings*

Input	Output
Gidley, Scott A	XYZ$$$

Input	Output
Scotty Gidleigh	XYZ$$$
Mr Scott Gidlee Jr.	XYZ$$$
Mr Robert J Brauer	ABC$$$
Bob Brauer	ABC$$$

The DQ_MATCH_PARSED() stored procedure analyzes pre-parsed values and generates a matchcode based on the concatenated values.

Table 2.7 *Example of Matching Pre-Parsed Values*

Prefix	Given Name	Middle Name	Family Name	Suffix	Output
	Scott	A.	Gidley		XYZ$$$
	Scotty		Gidleigh		XYZ$$$
Mr.	Scott		Gidlee	Jr.	XYZ$$$
Mr.	Robert	J	Brauer		ABC$$$
	Bob		Brauer		ABC$$$

Data quality stored procedures currently do not perform clustering. You need to retrieve matchcodes and matchcode index keys from the DBMS and use a product such as SAS Data Quality Server to cluster data to determine possible matches and to determine which record from each cluster is the survivor. After the survivors have been determined, issue SQL statements to select the cleansed data of index keys of relevant survivors and write them to the DBMS.

Parsing

Parsing segments a string into semantically atomic tokens. Parsing is performed with the DQ_PARSE() stored procedure.

Table 2.8 *Example of Parsing*

Input	Output	
Mr. Roy G. Biv Jr	Prefix:	Mr.
	Given Name:	Roy
	Middle Name:	G.
	Family Name:	Biv
	Suffix:	Jr

Pattern Analysis

Pattern analysis returns a simple representation of a text string's character pattern, which can be used for pattern frequency analysis in profiling jobs. Pattern analysis identifies words or characters in the input data column as numeric, alphabetic, non-alphanumeric, or mixed. The choice of pattern analysis definition determines the nature of the analysis. Pattern analysis is performed with the DQ_PATTERN() stored procedure.

Table 2.9 *Example of Pattern Analysis*

Input	Output
919-677-8000	999-999-9999
NC	AA

Standardization

Standardization generates a preferred standard representation of data values. Standardization definitions are provided for character content such as dates, names, and postal codes. The available standardization definitions vary from one locale to the next. The return values are provided in the appropriate case, and insignificant blank spaces and punctuation are removed. The order of the elements in the return values might differ from the order of the elements in the input character values.

Two stored procedures are provided for performing standardization. The DQ_STANDARDIZE() stored procedure generates a preferred standard representation of a text string.

Table 2.10 *Example of Standardization of Different Text Strings*

Input	Output
N car	NC
919.6778000	(919) 677–8000
Smith, Mister James	Mr. James Smith

The DQ_STANDARDIZE_PARSED() stored procedure evaluates and generates a preferred standard representation of pre-parsed values. It reads a set of input tokens created in advance by using the DQ_LIST_TOKENS() and DQ_BIND_TOKEN() stored procedures and then creates a single standardized output string

Table 2.11 *Example of Standardization of Pre-Parsed Values*

Input			Output
Prefix	**Given Name**	**Family Name**	**Standardized**
Mister	James	Smith	Mr. James Smith

Chapter 3
Understanding the Stored Procedures

Overview of Using the SAS Data Quality Accelerator for Teradata

The SAS Data Quality Accelerator for Teradata includes four categories of stored procedures: data quality, informational, session management, and pre-parsed support. The stored procedures in the informational, session management, and pre-parsed support categories provide supporting functionality for the data quality stored procedures. For more information, see “Categorized List of Stored Procedures” on page 13.

Categorized List of Stored Procedures

The stored procedures, by category, are as follows:

Table 3.1 *Categorized List of SAS Data Quality Accelerator for Teradata Stored Procedures*

Category	Name	Description
Data Quality Operations	DQ_EXTRACT()	Extracts specific entities or attributes from a text string.
	DQ_GENDER()	Determines the gender of a person from their name or other information. The input value is a text string.
	DQ_GENDER_PARSED()	Determines the gender of a person from their name or other information. The input consists of one or more parsed character values.
	DQ_IDENTIFY()	Determines the type of data that is represented by a text string.
	DQ_LOWERCASE()	Lowercases text.
	DQ_MATCH()	Generates a matchcode for a text string.
	DQ_MATCH_PARSED()	Generates a matchcode for one or more parsed character values.
	DQ_PARSE()	Segments a string into semantically atomic tokens.
	DQ_PATTERN()	Returns a simple representation of a text string's character pattern.
	DQ_PROPERCASE()	Applies uppercase and lowercase lettering using context-sensitive rules.
	DQ_STANDARDIZE()	Generates a preferred standard representation of a string.
	DQ_STANDARDIZE_PARSED()	Generates a preferred standard representation from one or more parsed values.
	DQ_UPPERCASE()	Uppercases text.
Informational	DQ_LIST_DEFNS()	Lists the QKB definitions that are available for the specified locale and definition type.
	DQ_LIST_LOCALES()	Lists the names of the locales that are installed in the QKB.
Session Management	DQ_SET_LOCALE()	Sets the locale for SAS Data Quality Accelerator stored procedures as a session value.
	DQ_SET_OPTION()	Sets options for the SAS Data Quality Accelerator session as key, value pairs.
Pre-Parsed Support	DQ_BIND_TOKEN()	Creates a token-to-column mapping for the specified token in the SAS Data Quality Accelerator session.

Category	Name	Description
	DQ_CLEAR_BINDINGS()	Clears token-to-column mappings from the SAS Data Quality Accelerator session.
	DQ_LIST_BINDINGS()	Generates an output table containing the current list of token-to-column mappings in the SAS Data Quality Accelerator session.
	DQ_LIST_TOKENS()	Lists the input tokens that are available for the specified operation, QKB definition, and locale.

Invoking the Stored Procedures

All of the stored procedures are invoked with the CALL keyword from any product that supports the Teradata SQL dialect. For example, they might be run from within the Teradata command-line interface, BTEQ, or they might be provided to the database via an ODBC connection in any language that supports the Teradata SQL dialect.

Base SAS software supports ODBC connections to the Teradata database when SAS/ACCESS Interface to Teradata software is installed. You can use the SQL procedure to execute the data quality stored procedures. The stored procedures are executed through the SQL pass-through facility. The PROC SQL explicit pass-through facility connects to the Teradata database in ANSI mode by default. You must specify the MODE= option in the LIBNAME statement to switch to Teradata mode. Consult the SAS/ACCESS Interface to Teradata documentation for more information about the MODE= option. For an example of how the stored procedures are submitted with PROC SQL, see “Using Data Quality Stored Procedures in Base SAS” on page 36.

How the Data Quality Stored Procedures Work

Overview

The data quality stored procedures use data quality rules from the SAS Quality Knowledge Base in order to cleanse data. The rules, referred to as *QKB definitions*, are operation- and locale-specific. That is, the definition that you specify in a stored procedure must be appropriate for both the locale and the data quality stored procedure. For more information, see “Selecting a QKB Definition” on page 16.

In order to execute any data quality stored procedure, you must set a locale. There are two ways to set a locale for a data quality operation. For more information, see “Setting a Locale” on page 16.

The majority of data quality stored procedures operate on the content of a single table column. For gender, match, and standardize operations, which must parse input data in order to evaluate it, the SAS Data Quality Accelerator provides parsing and pre-parsed variants. When the data to be examined is stored as a text string, use the DQ_GENDER(), DQ_MATCH(), and DQ_STANDARDIZE() stored procedures. These stored procedures parse the input data internally before evaluating it. Use

DQ_GENDER_PARSED(), DQ_MATCH_PARSED(), and DQ_STANDARDIZE_PARSED() to evaluate pre-parsed data.

The stored procedures can return output in an output table or in a dynamic result set via a cursor. For more information, see "Controlling Stored Procedure Output" on page 16.

The stored procedures give you the option to include a primary key in the output. Including the primary key in the output enables you to join data quality stored procedure output tables.

Selecting a QKB Definition

You can list the QKB definitions that are available to your SAS Data Quality Accelerator session as follows:

1. Execute the DQ_LIST_LOCALES() stored procedure to list the installed locales.
2. Execute the DQ_LIST_DEFNS() stored procedure to list the definitions that are available for a specified locale and operation type.

DQ_LIST_DEFNS() returns a list of definition names. For more information about the definitions, see the QKB Help in DataFlux Data Management Studio. You can open the QKB Help in the original QKB installation on Windows or UNIX. The Help is typically installed at C:\Program Files\DataFlux\QKB\CI\release-number\doc\html. Or you can use the DataFlux Data Management Studio Customize program to view the Help.

When selecting a definition, be careful to choose a definition that is appropriate for the stored procedure. For example, do not use a case definition that includes the string LOWER in its name in the DQ_PROPERCASE() stored procedure.

For more information, see "DQ_LIST_LOCALES()" on page 30 and "DQ_LIST_DEFNS()" on page 29.

Locating Token Names

In order to use the DQ_*OPERATION*_PARSED() stored procedures, you must know the token names that are supported by the QKB definition that you want to use. The tokens supported for a definition depend on the operation and the locale. Execute the DQ_LIST_TOKENS() stored procedure to obtain a list of valid token names for the specified operation type and definition name. For more information, see "DQ_LIST_TOKENS()" on page 31.

Setting a Locale

There are two ways to set a locale for a data quality operation:

- Specify a locale value in the *locale* parameter of the data quality stored procedure.
- Execute the DQ_SET_LOCALE() stored procedure at the start of the SAS Data Quality Accelerator session, and specify NULL in the stored procedure's *locale* parameter.

Valid locale values can be obtained by executing the DQ_LIST_LOCALES() stored procedure.

Controlling Stored Procedure Output

The stored procedures can return output in an output table or in a dynamic result set via a cursor. A *cursor* is a control structure that enables traversal over the records in a result set. It facilitates subsequent processing (such as retrieval, addition, and removal of database records) by enabling the rows in the result set to be processed sequentially.

The output that is returned by a data quality stored procedure is controlled with the *out-table* parameter. The stored procedures create an output table when you specify a table name in the *out-table* parameter. They return a cursor when NULL is specified as the output table name. Not all languages that support SAS Data Quality Accelerator for Teradata support cursors. The SAS SQL procedure is one language that does not support returned cursors.

For best results when creating an output table, execute the DQ_SET_OPTION() stored procedure at the start of the session to set the DQ_OVERWRITE_TABLE option, or specify the name of a non-existent table in *out-table*. By default, all of the stored procedures append data if the output table exists. In the case of append, the existing table must have the exact same schema as the output table, or an error is returned. The Append operation also fails if the same _PK_ value is specified. Executing the DQ_SET_OPTION() stored procedure with the DQ_OVERWRITE_TABLE option changes the default session behavior to overwrite existing tables. The setting lasts for the duration of the session or until you change it by executing DQ_SET_OPTION('DQ_OVERWRITE_TABLE', '0').

Using the DQ_OPERATION_PARSED() Stored Procedures

The DQ_*OPERATION*_PARSED() stored procedures use token-to-column mappings to identify the data to be operated on by the stored procedure instead of a column name. The use of token-to-column mappings enables multiple columns to be submitted as input to the stored procedures while providing flexibility in the number of columns that can be submitted.

You identify the tokens that need mapping by executing the DQ_LIST_TOKENS() stored procedure. You create the token-to-column mappings by executing the DQ_BIND_TOKEN() stored procedure. The DQ_BIND_TOKEN() stored procedure maps one token to one column. The stored procedure must be run at least once for each token that will be included in the data quality evaluation before a DQ_*OPERATION*_PARSED() stored procedure is run. Ideally, DQ_BIND_TOKEN() is executed for all of the input tokens supported for the QKB definition that will be used to cleanse the data. Unmapped tokens are treated as empty strings by the stored procedures.

DQ_BIND_TOKEN() is a session-based stored procedure. The mappings that it creates remain in session memory until you remove them, or until you log off. You can check for mappings in a session by executing the DQ_LIST_BINDINGS() stored procedure. You can remove them by executing the DQ_CLEAR_BINDINGS() stored procedure.

For best results, execute DQ_CLEAR_BINDINGS() to clear any previously used token-to-column mappings before creating new mappings.

Content of the Stored Procedure Output Tables

An output table is created when the SAS Data Quality Accelerator stored procedures run without error.

Data Quality Stored Procedures

The output tables returned by the DQ_*OPERATION*() stored procedures have the following columns:

- Column _INPUT_ shows what the data in the column specified in the *data_column* parameter of the stored procedure call looked like before the data quality stored procedure was run.
- Column _ERR_ contains a blank value or an error message. A blank value indicates that the stored procedure ran successfully. Error messages are returned for memory errors and errors caused by internal processes.
- Column *Result* holds the result of applying the data quality operation to each input found in the column specified in the *data_column* parameter of the stored procedure call. This column is named after the operation (for example, **`Gender`**, **`Identified`**, **`Matchcode`**, **`Lowercase`**, **`Pattern`**, **`Propercase`**, **`Standardized`**, and **`Uppercase`**).

  The DQ_EXTRACT() and DQ_PARSE() stored procedures return a result column for each token present in the QKB definition named in the *definition* parameter of the stored procedure call. The columns are named after the tokens.
- Column _PK_ contains the values from the data column that was specified in the *pk_column* parameter of the stored procedure call. If the value of the *pk_column* parameter is NULL in the stored procedure call, then column _PK_ is not created in the output table. When column _PK_ is created, it is created with the PRIMARY KEY constraint, permitting joins with the source table.

The output tables created by the DQ_*OPERATION*_PARSED() stored procedures include _ERR_, *Result*, and can include_PK_. The *Result* column contains a string of the concatenated, cleansed input column values.

All of the columns in the output tables except column _PK_ and the result column for DQ_GENDER() have data type VARCHAR(1024). Column _PK_ has the same data type as the column that was specified in the *pk-column* parameter in the stored procedure call. The data type for the Gender column is VARCHAR(1).

Informational Stored Procedures

The DQ_LIST_LOCALES() and DQ_LIST_DEFNS() stored procedures return output tables consisting of two columns each. One column is specific to the stored procedure. The other column is_ERR_.

- The name of the column created by DQ_LIST_LOCALES() is **`Locale Name`**.
- The name of the column created by DQ_LIST_DEFNS() is **`Definition Name`**.

The DQ_LIST_TOKENS() stored procedure returns an output table consisting of two columns. The columns are named **`Order`** and **`Token Name`**. **`Order`** specifies a number that indicates the order in which the definition uses the token. The **`Order`** column is omitted when output is written to a cursor.

The DQ_LIST_BINDINGS() stored procedure returns an output table consisting of two columns. These columns are named **`Token Name`** and **`Column Name`**.

Pre-Parsed Stored Procedures

The DQ_BIND_TOKEN() and DQ_CLEAR_BINDINGS() stored procedures do not return an output table. DQ_BIND_TOKEN() creates token-to-column mappings in the SAS Data Quality Accelerator session that can be listed with DQ_LIST_BINDINGS().

Chapter 4

Stored Procedure: Reference

Overview

All of the data quality stored procedures have the same general syntax, except for DQ_MATCH(), which has an additional, operation-specific parameter (*sensitivity*). The DQ_*OPERATION*_PARSED() stored procedures take one fewer parameter than the DQ_*OPERATION*() stored procedures. (The _PARSED procedures do not have a *data-column* parameter.) For more information about the syntax for the data quality stored procedures, see "DQ_OPERATION() Syntax" on page 19 and "DQ_OPERATION_PARSED() Syntax" on page 22.

For syntax information about the informational, session management, and pre-parsed support stored procedures, see the stored procedure.

DQ_*OPERATION*() Syntax

Executes the specified data quality operation on the specified data column, internally parsing the input string if necessary.

Interaction

Applies to the DQ_EXTRACT(), DQ_GENDER(), DQ_IDENTIFY(), DQ_LOWERCASE(), DQ_MATCH(), DQ_PARSE(), DQ_PATTERN(), DQ_PROPERCASE(), and DQ_STANDARDIZE() stored procedures.

Syntax

DQ_*OPERATION* ('*definition*', <'*sensitivity*',> '*in-table*', '*data-column*', '*pk-column*', '*out-table*', '*locale*');

Parameters

OPERATION
specifies the name of the data quality operation. Valid values are EXTRACT, GENDER, IDENTIFY, LOWERCASE, MATCH, PARSE, PATTERN, PROPERCASE, or STANDARDIZE. Preface the completed stored procedure name with the name of the database in which it is located. For example, SAS_SYSFNLIB.DQ_OPERATION(). SAS_SYSFNLIB is the required database name.

'*definition*'
specifies the name of a SAS Quality Knowledge Base definition. The definition must be valid for the locale. Use "DQ_LIST_DEFNS()" on page 29 to identify the definitions that are available for the specified operation and locale.

'*sensitivity*'
specifies an integer that represents the degree of similarity that should be applied to consider something a match. The *sensitivity* parameter is used by the DQ_MATCH() operation only, for which it is required. Valid values range from 50 to 95. The recommended sensitivity setting is 85. A higher sensitivity value requires greater similarity between input values for a match to be obtained. In general, higher sensitivity leads to fewer matches than lower sensitivity.

'*in-table*'
specifies the name of an existing database table containing data to be processed. Preface the table name with the database name.

'*data-column*'
specifies the name of the column in *in-table* that contains the data to be processed. This column must be a VARCHAR type.

Note: Any column name that is a reserved word in Teradata needs to be quoted when passed into a data quality stored procedure. For example:

```
call sas_sysfnlib.dq_parse('Date (MDY)', 'mydb.mytable', '"date"',
'_PK_', 'mydb.parsed', 'null')
```

'*pk-column*'
specifies the name of the column in *in-table* that contains the primary key or a null value. When this parameter is passed as NULL, then no primary key is transferred to the output table.

'*out-table*'
specifies the name of the database table where the result rows are written or a null value. Preface the table name with the database name. If the table does not exist, it is created.

For the best results, specify a non-existent table name, or use the DQ_SET_OPTION() stored procedure to set the DQ_OVERWRITE_TABLE option at the start of the session. By default, SAS Data Quality Accelerator stored procedures append results if a table with the name of the specified output table already exists. The DQ_OVERWRITE_TABLE option changes the default behavior to overwrite any existing tables. For more information, see "DQ_SET_OPTION()" on page 33.

When the *out-table* parameter is passed as NULL, the result set is dynamic and is returned as a cursor rather than being written to a specific table. For more information, see "Controlling Stored Procedure Output" on page 16.

'*locale*'
specifies an uppercase string specifying the five-letter ISO code name of the QKB locale to use for the session (for example, ENUSA) or a null value. When the *locale* parameter is passed as NULL, the locale value set by DQ_SET_LOCALE() stored procedure is used. You can use the DQ_LIST_LOCALES() stored procedure to identify the available locales. For more information, see "DQ_LIST_LOCALES()" on page 30.

Details

The strings passed to the data quality stored procedures need to be encoded in the ISO-8859-1 (Latin-1) encoding or some compatible encoding. This means that any table names or column names containing non-Western European characters are not supported. This restriction does not apply to the actual data in the columns. Unicode is supported within the columns, just not in the table metadata.

All parameters, except *sensitivity*, accept a string of up to 256 characters.

Example 1: DQ_GENDER()

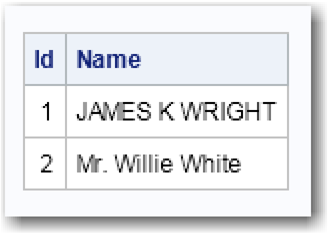
This example uses the DQ_GENDER() stored procedure to return a gender value based on the text in the Name column of a table named People. The stored procedure uses a QKB definition named Name. The output is stored in a table named People_Gend.

```
call sas_sysfnlib.dq_gender(
      'Name',
      'mydb.People',
      'Name',
      'Id',
      'mydb.People_Gend',
      'ENUSA');

select * from mydb.People_Gend;
```

This is the content of the source table, People.

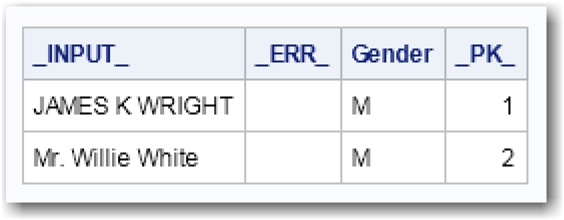
Figure 4.1 *People Table*



Id	Name
1	JAMES K WRIGHT
2	Mr. Willie White

This is the content of the table that is returned by the DQ_GENDER() stored procedure. The gender value is in the Gender column.

Figure 4.2 *Results of the DQ_GENDER() Stored Procedure*



INPUT	_ERR_	Gender	_PK_
JAMES K WRIGHT		M	1
Mr. Willie White		M	2

Example 2: DQ_STANDARDIZE()

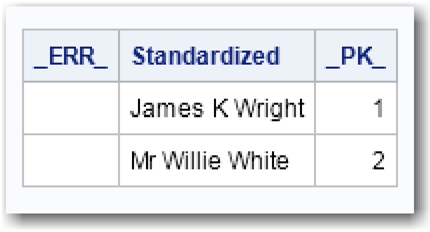
This example uses the DQ_STANDARDIZE() stored procedure to standardize the values in the Name column of the People table. The stored procedure also uses a QKB definition named Name. The output is stored in a table named People_Std.

```
call sas_sysfnlib.dq_standardize(
      'Name',
      'mydb.People',
      'Name',
      'Id',
      'mydb.People_Std',
      'ENUSA');

select * from mydb.People_Std;
```

This is the content of table returned by the DQ_STANDARDIZE() stored procedure.

Figure 4.3 *Results of the DQ_STANDARDIZE() Stored Procedure*



ERR	Standardized	_PK_
	James K Wright	1
	Mr Willie White	2

DQ_*OPERATION*_PARSED() Syntax

Executes the specified data quality operation on pre-parsed input tables. This syntax is available only for gender, match, and standardize operations.

Interaction

Applies to the DQ_GENDER_PARSED(), DQ_MATCH_PARSED(), and DQ_STANDARDIZE_PARSED() stored procedures.

Requirements

You must create token-to-column mappings with the DQ_LIST_TOKENS() and DQ_BIND_TOKEN() stored procedures before executing a DQ_*OPERATION*_PARSED() stored procedure.

Syntax

DQ_*OPERATION*_PARSED ('*definition*', <'*sensitivity*',> '*in-table*', '*pk-column*', '*out-table*', '*locale*');

Parameters

OPERATION
: specifies the name of the data quality operation. Valid values are GENDER, MATCH, or STANDARDIZE. Preface the completed stored procedure name with the name of the database in which it is located. For example, SAS_SYSFNLIB.DQ_GENDER_PARSED(). SAS_SYSFNLIB is the required database name.

'*definition*'
: specifies the name of a SAS Quality Knowledge Base definition. The definition must be valid in the current locale. Use "DQ_LIST_DEFNS()" on page 29 to identify the definitions that are available for the specified operation and locale.

'*sensitivity*'
: specifies an integer that represents the degree of similarity that should be applied to consider something a match. The *sensitivity* parameter is used by the DQ_MATCH_PARSED() operation only, for which it is required. Valid values range from 50 to 95. The recommended sensitivity setting is 85. A higher sensitivity value requires greater similarity between input values for a match to be obtained. In general, higher sensitivity leads to fewer matches than lower sensitivity.

'*in-table*'
: specifies the name of an existing database table containing data to be processed. Preface the table name with the database name.

'*pk-column*'
: specifies the name of the column in *in-table* that contains the primary key or a null value. When this parameter is passed as NULL, then no primary key is transferred to the output table.

'*out-table*'
: specifies the name of the database table where the result rows are written or a null value. Preface the table name with the database name. If the table does not exist, it is created.

  For the best results, specify a non-existent table name, or use the DQ_SET_OPTION() stored procedure to set the DQ_OVERWRITE_TABLE option at the start of the session. By default, SAS Data Quality Accelerator stored procedures append results if a table with the name of the specified output table already exists. The DQ_OVERWRITE_TABLE option changes the default behavior to overwrite existing tables. For more information, see "DQ_SET_OPTION()" on page 33.

When the *out-table* parameter is passed as NULL, the result set is dynamic and is returned as a cursor rather than being written to a specific table. For more information, see “Controlling Stored Procedure Output” on page 16.

'locale'
specifies an uppercase string specifying the five-letter ISO code name of the QKB locale to use for the session (for example, ENUSA) or a null value. When the *locale* parameter is passed as NULL, the locale value set by the DQ_SET_LOCALE() stored procedure is used. You can use the DQ_LIST_LOCALES() stored procedure to identify the available locales. For more information, see “DQ_LIST_LOCALES()” on page 30.

Details

The strings passed to the data quality stored procedures need to be encoded in the ISO-8859-1 (Latin-1) encoding or some compatible encoding. This means that any table names or column names containing non-Western European characters are not supported. This restriction does not apply to the actual data in the columns. Unicode is supported within the columns, just not in the table metadata.

All parameters except for *sensitivity* accept a string of up to 256 characters.

Example 1: DQ_GENDER_PARSED()

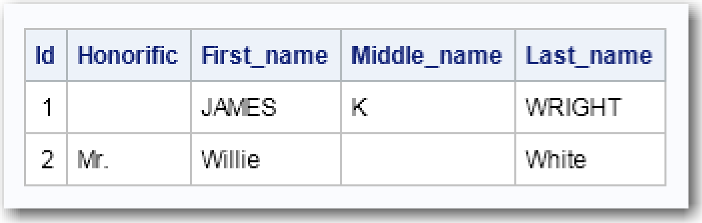
This example uses the DQ_GENDER_PARSED() stored procedure to return a gender value based on the values in the columns Honorific, First Name, and Middle Name from a table named PeopleParsed. The stored procedure uses a QKB definition called Name, which has Pre-Parsed support enabled in the QKB. The stored procedure reads a set of tokens that are created in advance by using the DQ_LIST_TOKENS() and DQ_BIND_TOKEN() stored procedures and then creates a single standardized output string based on those tokens. The output of the DQ_GENDER_PARSED() operation is stored in a table named PeopleParsed_Gend.

```
/* Execute DQ_LIST_TOKENS(), DQ_BIND_TOKEN() and DQ_LIST_BINDINGS() first */
call sas_sysfnlib.dq_gender_parsed(
     'Name',
     'mydb.PeopleParsed',
     'Id',
     'mydb.PeopleParsed_Gend',
     'ENUSA');

select * from mydb.PeopleParsed_Gend;
```

This is the content of the source table, PeopleParsed. Note that the name values are split across four columns.

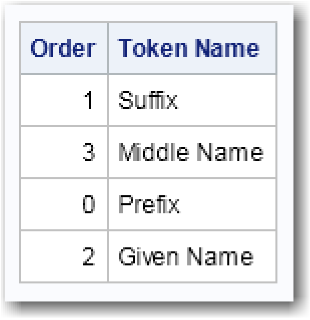
Figure 4.4 *PeopleParsed Table*



Id	Honorific	First_name	Middle_name	Last_name
1		JAMES	K	WRIGHT
2	Mr.	Willie		White

These are the input tokens returned by DQ_LIST_TOKENS() for a gender operation in the ENUSA locale. See "DQ_LIST_TOKENS()" on page 31 for an example of the code that was used to obtain the input tokens.

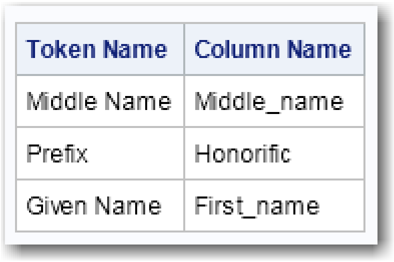
Figure 4.5 *Tokens Supported for a Gender Operation in the ENUSA Locale*



Order	Token Name
1	Suffix
3	Middle Name
0	Prefix
2	Given Name

These are the column mappings. See "DQ_BIND_TOKEN()" on page 27 for the code to create the token-to-column maps. See "DQ_LIST_BINDINGS()" on page 28for the code to display the token-to-column mappings.

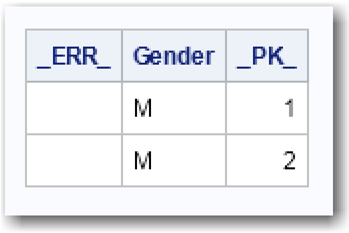
Figure 4.6 *Column Mappings Defined for the PeopleParsed Gender Operation*



Token Name	Column Name
Middle Name	Middle_name
Prefix	Honorific
Given Name	First_name

These are the results of the DQ_GENDER_PARSED() stored procedure.

Figure 4.7 *DQ_GENDER_PARSED() Output Table*



ERR	Gender	_PK_
	M	1
	M	2

Example 2: DQ_STANDARDIZE_PARSED()

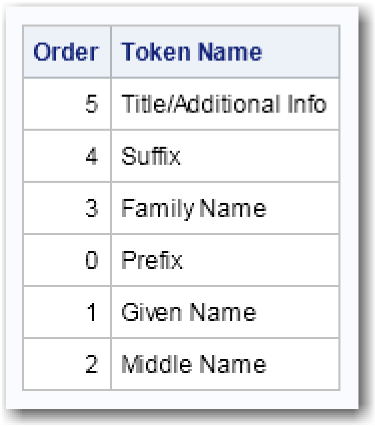
This example uses the DQ_STANDARDIZE_PARSED() stored procedure to standardize the values in the Honorific, Middle_name, Given_name, and Family_name columns of the PeopleParsed table. The stored procedure uses a QKB definition called Name, which has Pre-Parsed support enabled in the QKB. The stored procedure reads a set of tokens that are created in advance by using the DQ_LIST_TOKENS() and DQ_BIND_TOKEN() stored procedures and then creates a single standardized output string based on those tokens. The output is stored in a table named PeopleParsed_Std.

```
/* Execute DB_LIST_TOKENS, DB_BIND_TOKEN() and DQ_LIST_BINDINGS() first */
call sas_sysfnlib.dq_standardize_parsed(
     'Name',
     'mydb.PeopleParsed',
     'Id',
     'mydb.PeopleParsed_Std',
     'ENUSA');

select * from mydb.PeopleParsed_Std;
```

These are the input tokens returned by the DQ_LIST_TOKENS() stored procedure for the standardization operation in the ENUSA locale. See "DQ_LIST_TOKENS()" on page 31 for the code to obtain the tokens.

***Figure 4.8** Tokens Supported for Standardizing Names in the ENUSA Locale*

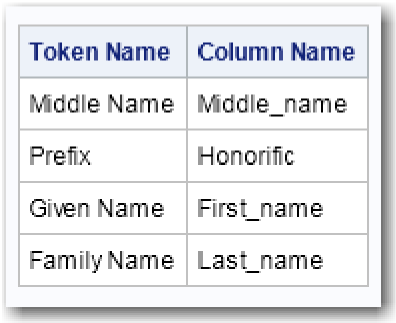


Order	Token Name
5	Title/Additional Info
4	Suffix
3	Family Name
0	Prefix
1	Given Name
2	Middle Name

The Name definition provided for standardization operations supports several more input tokens than the Name definition for gender operations. It is not necessary to create a column mapping for every token. See "DQ_BIND_TOKEN()" on page 27 for the code to create the column mappings and "DQ_LIST_BINDINGS()" on page 28 for the code to display them.

These are the column mappings.

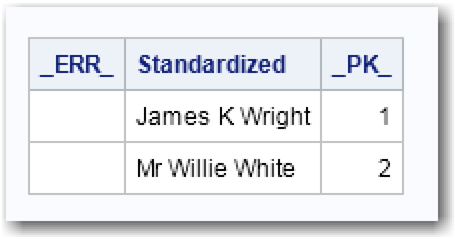
***Figure 4.9** Column Mappings Defined for the PeopleParsed Standardization Operation*



Token Name	Column Name
Middle Name	Middle_name
Prefix	Honorific
Given Name	First_name
Family Name	Last_name

These are the results of the DQ_STANDARDIZE_PARSED() stored procedure:

Figure 4.10 *DQ_STANDARDIZE_PARSED() Output Table*



ERR	Standardized	_PK_
	James K Wright	1
	Mr Willie White	2

Dictionary

DQ_BIND_TOKEN()

Creates a token-to-column mapping for the specified token in the SAS Data Quality Accelerator session.

Category: Pre-Parsed Support

Requirements: The DQ_BIND_TOKEN() stored procedure must be executed at least once before executing a DQ_*OPERATION*_PARSED() stored procedure.

DQ_BIND_TOKEN() does not have visible output. Use DQ_LIST_BINDINGS() to display available token-to-column mappings.

Syntax

DQ_BIND_TOKEN('*token-name*', '*data-column*')

Parameters

token-name
specifies the name of an input token. Use the DQ_LIST_TOKENS() stored procedure to obtain a list of valid token names. For more information, see "DQ_LIST_TOKENS()" on page 31.

data-column
specifies the name of the data column to which to bind the token. The data column must exist in the table that is specified in the *in-table* parameter of the DQ_*OPERATION*_PARSED() stored procedure that follows. This column must be a VARCHAR type.

Details

This stored procedure creates one token-to-column map. Execute the stored procedure for each valid input token that you want to include in the evaluation of the specified QKB definition and operation.

DQ_BIND_TOKEN() is a session-based stored procedure. The token-to-column mappings created with DQ_BIND_TOKEN() remain in the SAS Data Quality Accelerator session until the session ends, or until you remove them with the DQ_CLEAR_BINDINGS() stored procedure. It is a good idea to check for and clear token-to-column mappings from any previous DQ_*OPERATION*_PARSED() operations before creating token-to-column mappings for another operation. Otherwise, any irrelevant token-to-column mappings that remain in the session will have a column in the next DQ_*OPERATION*_PARSED() stored procedure's output table. For more information, see "DQ_LIST_BINDINGS()" on page 28 and "DQ_CLEAR_BINDINGS()" on page 28.

Unmapped tokens are treated as empty strings by the DQ_*OPERATION*_PARSED() stored procedures.

Examples

Example 1

The following code created the token-to-column mappings depicted in Figure 4.6 on page 25.

```
call sas_sysfnlib.dq_bind_token('Prefix', 'Honorific')
call sas_sysfnlib.dq_bind_token('Given Name', 'First_name')
call sas_sysfnlib.dq_bind_token('Middle Name', 'Middle_name')
```

Example 2

The following code created the token-to-column mappings in Figure 4.9 on page 26.

```
call sas_sysfnlib.dq_bind_token('Prefix', 'Honorific')
call sas_sysfnlib.dq_bind_token('Given Name', 'First_name')
call sas_sysfnlib.dq_bind_token('Middle Name', 'Middle_name')
call sas_sysfnlib.dq_bind_token('Family Name', 'Last_name')
```

DQ_CLEAR_BINDINGS()

Removes token-to-column mappings from the SAS Data Quality Accelerator session.

Category: Pre-Parsed Support

Syntax

DQ_CLEAR_BINDINGS()

Parameters

This stored procedure takes no parameters. It removes all token-to-column mappings in the SAS Data Quality Accelerator session.

Example

```
call sas_sysfnlib.dq_clear_bindings()
```

DQ_LIST_BINDINGS()

Lists the token-to-column mappings in the SAS Data Quality Accelerator session.

Category: Pre-Parsed Support

Interaction: Use DQ_LIST_BINDINGS() to list the token-to-column mappings that were created with DQ_BIND_TOKEN().

Syntax

DQ_LIST_BINDINGS('*out-table*')

Parameters

'out-table'

(optional) specifies the name of the database table where the result rows are written. Preface the table name with the database name. If the table does not exist, it is created.

For the best results, specify a non-existent table name, or use the DQ_SET_OPTION() stored procedure to set the DQ_OVERWRITE_TABLE option at the start of the session. By default, SAS Data Quality Accelerator stored procedures append results if a table with the name of the specified output table already exists. The DQ_OVERWRITE_TABLE option changes the default behavior to overwrite any existing tables. For more information, see "DQ_SET_OPTION()" on page 33.

When the *out-table* parameter is passed as NULL, the result set is dynamic and is returned as a cursor rather than being written to a specific table. For more information, see "Controlling Stored Procedure Output" on page 16.

Details

The token-to-column mappings remain in the SAS Data Quality Accelerator session until it ends, or until you use the DQ_CLEAR_BINDINGS() stored procedure to remove the column mappings from the session.

Examples

Example 1

The following code was used to display the token-to-column mappings in Figure 4.6 on page 25.

```
call sas_sysfnlib.dq_list_bindings('mydb.gendermaps')
```

Example 2

The following code was used to display the token-to-column mappings in Figure 4.9 on page 26.

```
call sas_sysfnlib.dq_list_bindings('mydb.standaridzemaps')
```

DQ_LIST_DEFNS()

Lists the QKB definitions available for the specified data quality operation type and locale.

Category: Informational

Syntax

DQ_LIST_DEFNS('*locale*', '*operation-type*', '*out-table*')

Parameters

'*locale*'
specifies an uppercase string specifying the five-letter ISO code name of the QKB locale to use for the session (for example, the ENUSA code name represents the “English, United States” locale). When the locale parameter is passed as NULL, the locale value set by DQ_SET_LOCALE() stored procedure is used. Use the DQ_LIST_LOCALES() stored procedure to identify the available locales. For more information, see “DQ_LIST_LOCALES()” on page 30.

'*operation-type*'
specifies a string that identifies the data quality operation for which to return definitions. Valid values are as follows:

- CASE
- EXTRACTION
- GENDER
- IDENTIFICATION
- MATCH
- PARSE
- PATTERN
- STANDARDIZATION

'*out-table*'
(optional) specifies the name of the database table where the result rows are written. Preface the table name with the database name. If the table does not exist, it is created.

For the best results, specify a non-existent table name, or use the DQ_SET_OPTION() stored procedure to set the DQ_OVERWRITE_TABLE option at the start of the session. By default, SAS Data Quality Accelerator stored procedures append results if a table with the name of the specified output table already exists. The DQ_OVERWRITE_TABLE option changes the default behavior to overwrite any existing tables. For more information, see “DQ_SET_OPTION()” on page 33.

When the *out-table* parameter is passed as NULL, the result set is dynamic and is returned as a cursor rather than being written to a specific table. For more information, see “Controlling Stored Procedure Output” on page 16.

Example

See “Using Data Quality Stored Procedures in Base SAS” on page 36 for more DQ_LIST_DEFNS() examples, and screen captures of the output tables.

```
call sas_sysfnlib.dq_list_defns('ENUSA', 'parse', 'mydb.parsedefs')
```

DQ_LIST_LOCALES()

Lists the locales available to SAS Data Quality Accelerator session.

Category: Informational

Syntax

DQ_LIST_LOCALES('*out-table*')

Parameters

'out-table'

(optional) specifies the name of the database table where the result rows are written. Preface the table name with the database name. If the table does not exist, it is created.

For the best results, specify a non-existent table name, or use the DQ_SET_OPTION() stored procedure to set the DQ_OVERWRITE_TABLE option at the start of the session. By default, SAS Data Quality Accelerator stored procedures append results if a table with the name of the specified output table already exists. The DQ_OVERWRITE_TABLE option changes the default behavior to overwrite any existing tables. For more information, see "DQ_SET_OPTION()" on page 33.

When the *out-table* parameter is passed as NULL, the result set is dynamic and is returned as a cursor rather than being written to a specific table. For more information, see "Controlling Stored Procedure Output" on page 16.

Example

See "Using Data Quality Stored Procedures in Base SAS" on page 36 for an example of a DQ_LIST_LOCALES() output table.

```
call sas_sysfnlib.dq_list_locales('mydb.loclist')
```

DQ_LIST_TOKENS()

Lists the input tokens that are available for a specified operation, Quality Knowledge Base definition, and locale.

Category: Pre-Parsed Support

Interaction: DQ_LIST_TOKENS() returns information needed by DQ_BIND_TOKEN().

Syntax

DQ_LIST_TOKENS('*locale*', '*operation-type*', '*definition*', '*out-table*')

Parameters

'locale'

is an uppercase string specifying the five-letter ISO code name of the QKB locale to use for the session (for example, ENUSA). Use the DQ_LIST_LOCALES() stored procedure to list the available locales. For more information, see "DQ_LIST_LOCALES()" on page 30.

'*operation-type*'
specifies a string that identifies the data quality operation for which the tokens are used. Valid values are as follows:

- GENDER
- MATCH
- PARSE
- STANDARDIZATION

'*definition*'
specifies the name of a SAS Quality Knowledge Base definition. The definition must be valid in the current locale. Use the DQ_LIST_DEFNS() stored procedure to identify the definitions that are available for the specified operation and locale. For more information, see "DQ_LIST_DEFNS()" on page 29.

'*out-table*'
(optional) specifies the name of the database table where the result rows are written. Preface the table name with the database name. If the table does not exist, it is created.

For the best results, specify a non-existent table name, or use the DQ_SET_OPTION() stored procedure to set the DQ_OVERWRITE_TABLE option at the start of the session. By default, SAS Data Quality Accelerator stored procedures append results if a table with the name of the specified output table already exists. The DQ_OVERWRITE_TABLE option changes the default behavior to overwrite any existing tables. For more information, see "DQ_SET_OPTION()" on page 33.

When the *out-table* parameter is passed as NULL, the result set is dynamic and is returned as a cursor rather than being written to a specific table. For more information, see "Controlling Stored Procedure Output" on page 16.

Details

DQ_LIST_TOKENS() identifies the input tokens that a QKB definition needs to make an evaluation for a specified operation and locale. You use this stored procedure to obtain the token names to specify in the DQ_BIND_TOKEN() stored procedure.

Not all tokens that are returned by DQ_LIST_TOKEN() must have a column mapped to them. However, at least one valid token-to-column mapping must exist in the SAS Data Quality Accelerator session for the DQ_*OPERATION*_PARSED() stored procedure to execute successfully.

Examples

Example 1

The following code created the token listing depicted in Figure 4.5 on page 25.

```
call sas_sysfnlib.dq_list_tokens(
     'ENUSA',
     'gender',
     'name',
     'mydb.Name_tokens');

select * from mydb.Name_tokens;
```

Example 2

The following code created the token listing depicted in Figure 4.8 on page 26.

```
call sas_sysfnlib.dq_list_tokens(
     'ENUSA',
     'standardization',
     'name',
     'mydb.Standardize_tokens');

select * from mydb.Standardize_tokens;
```

DQ_SET_LOCALE()

Establishes the locale for an in-database data quality stored procedure session.

Category: Session Management

Syntax

DQ_SET_LOCALE('*locale*')

Parameters

'locale'
is an uppercase string specifying the five-letter ISO code name of the QKB locale to use for the session (for example, ENUSA). You can obtain a listing of the valid locale values by using the DQ_LIST_LOCALES() stored procedures. For more information, see "DQ_LIST_LOCALES()" on page 30.

Details

DQ_SET_LOCALE is a session-based stored procedure that establishes the locale for the SAS Data Quality Accelerator session. The locale value set with this stored procedure is used when NULL is specified in the *locale* parameter of a SAS Data Quality Accelerator stored procedure.

Example

```
call sas_sysfnlib.dq_set_locale('ENUSA')
```

DQ_SET_OPTION()

Invokes optional behavior in the SAS Data Quality Accelerator session.

Category: Session Management

Syntax

DQ_SET_OPTION('*keyword*', '*value*')

Parameters

'*keyword*'
specifies the optional behavior to invoke. Currently one option is supported:

'DQ_OVERWRITE_TABLE'
specifies to overwrite existing stored procedure output tables instead of appending data to them.

'*value*'
specifies a Boolean value that controls the optional behavior. Specify 1 to turn on the optional behavior. Specify 0 (the default value) to turn it off. The option is session-based. If the default value is overridden, the session is reset to the default value when you log off.

Example

```
call sas_sysfnlib.dq_set_option('dq_overwrite_table', '1');
```

Chapter 5

Examples: Using the Stored Procedures in PROC SQL and BTEQ

Usage Overview

The traditional data quality process involves the following steps:

1. Extract uncleansed source data from the DBMS to the SAS Data Quality Server.
2. Standardize the data.
3. Publish the cleansed data to the DBMS.

With SAS Data Quality Accelerator for Teradata, the extraction step is removed, and the process consists only of connecting to the DBMS and calling stored procedures to standardize the uncleansed source data.

Overview of Examples

This section shows two ways that you can connect to a Teradata database and execute SAS Data Quality Accelerator for Teradata stored procedures. The first example uses the Base SAS SQL procedure to execute the stored procedures. The second example uses the Teradata BTEQ utility to execute the stored procedures.

The sample code creates a small source table named MyDB.Employees, which contains raw data. Then, it executes SAS Data Quality Accelerator stored procedures to cleanse the data and create a report with the cleansed data. In the examples:

- The DQ_LIST_LOCALES() store procedure identifies the locales that are available to the session.
- The DQ_LIST_DEFNS() stored procedure lists the gender, parse, and standardize definitions for the ENUSA locale.

- The DQ_SET_OPTION() stored procedure specifies to overwrite any existing stored procedure output tables.
- The DQ_GENDER() stored procedure determines the gender of the employees in the source table.
- The DQ_PARSE() stored procedure splits the employee’s first, middle, and last names.
- The DQ_STANDARDIZE() stored procedure standardizes the employees' addresses.
- SQL code joins the cleansed data.

The examples use the “English, United States” locale.

The Base SAS sample program contains extra code to print the stored procedure output tables.

Using Data Quality Stored Procedures in Base SAS

In a Base SAS session, you must use the SQL procedure to execute the data quality stored procedures. The stored procedures are executed through the SQL pass-through facility.

Note: You must have SAS Foundation software and SAS/ACCESS Interface to Teradata software installed on your client machine in order to make PROC SQL requests to the Teradata database.

```
proc sql;
  /* Establish the connection to Teradata. */
  connect to teradata
    (server=teraserver
     user=myid
     password=mypwd
     database=mydb
     mode=teradata); 1

/* Set option to overwrite output tables */ 2
execute (
  call sas_sysfnlib.dq_set_option('DQ_OVERWRITE_TABLE', '1')
  ) by teradata;

/* List the available locales */ 3
  execute (
    call sas_sysfnlib.dq_list_locales('mydb.loclist')
  ) by teradata;
title 'Available Locales';
select * from connection to teradata (select * from mydb.loclist);

/* List the available gender, parse, and standardize */ 4
/* definitions for the ENUSA locale */

  execute (
    call sas_sysfnlib.dq_list_defns('ENUSA', 'gender','mydb.gendefs')
  ) by teradata;
title 'Listing of ENUSA Gender Definitions';
```

```
select * from connection to teradata (select * from mydb.gendefs);

  execute (
    call sas_sysfnlib.dq_list_defns('ENUSA', 'parse','mydb.parsdefs')
  ) by teradata;
title 'Partial Listing of ENUSA Parse Definitions';
select * from connection to teradata (select * from mydb.parsdefs);

  execute (
    call sas_sysfnlib.dq_list_defns('ENUSA', 'standardization','mydb.standdefs')
  ) by teradata;
title 'Partial Listing of ENUSA Standardization Definitions';
select * from connection to teradata (select * from mydb.standdefs);

  /* Create an example source table. */
  execute (
    create table mydb.employees
      (empid      integer primary key not null,
       name       varchar(50),
       address    varchar(100),
       start_date varchar(9),
       birth_date varchar(9))
  ) by teradata;

  /* Populate the source table with some data. */
  execute (
    insert into mydb.employees values
        (203,
         'Ken Patrick Burke',
         '445 main street',
         '27JAN1994',
         '14JAN1950')
  ) by teradata;

  execute (
    insert into mydb.employees values
        (485,
         'Stella Mae Santorini',
         '160-a NORTH 6TH ST # 3',
         '18DEC2005',
         '17JUL1980')
  ) by teradata;

  execute (
    insert into mydb.employees values
        (1003,
         'Dale Michael Hayes',
         '1575 Jordon St, Suite 3 2nd floor',
         '01AUG2009',
         '09OCT1985')
  ) by teradata;

  execute (
    insert into mydb.employees values
        (460,
         'Angela Sarah Koehlepp',
           'Number 704 Martin Luther King Blvd',
           '14FEB2005',
           '12MAY1978')
    ) by teradata;

    execute (
      insert into mydb.employees values
          (255,
           'Ross Maxwell Petersen',
           'POBOX 4 SMITH STREET',
           '23NOV1996',
           '12MAR1968')
    ) by teradata;

  /* print the example table */
  title 'Content of Employees Table';
  select * from connection to teradata (select * from mydb.employees);

    /* Determine the gender of employees in the source table. */ 5
    execute (
      call sas_sysfnlib.dq_gender(
        'Name',
        'mydb.employees',
        'name',
        'empid',
        'mydb.employees_gend',
        'ENUSA')
    ) by teradata;

  /* print the output table */
  title 'Content of Employees_gend Table';
  select * from connection to teradata (select * from mydb.employees_gend);

    /* Run DQ_PARSE() to split the first, middle, and last name. */ 6
    execute (
      call sas_sysfnlib.dq_parse(
        'Name',
        'mydb.employees',
        'name',
        'empid',
        'mydb.employees_pars',
        'ENUSA')
    ) by teradata;

  /* print the output table */
  title 'Content of Employees_pars Table';
  select * from connection to teradata (select * from mydb.employees_pars);

    /* Standardize the employees' addresses. */ 7
    execute (
      call sas_sysfnlib.dq_standardize(
        'Address',
        'mydb.employees',
        'address',
        'empid',
        'mydb.employees_std',
      'ENUSA')
  ) by teradata;

/* print the output table */
title 'Content of Employees_std Table';
select * from connection to teradata (select * from mydb.employees_std);

  /* Aggregate the processed data into a report. */ 8
title 'Cleansed Employees Table';
  select * from connection to teradata (
    select
     e.empid,
     p."Given Name",
     p."Middle Name",
     p."Family Name",
     e.birth_date,
     e.start_date,
     g.Gender,
     s.Standardized
  from
     mydb.employees e,
     mydb.employees_gend g,
     mydb.employees_pars p,
     mydb.employees_std s
  where
    e.empid=g._PK_ and
    e.empid=p._PK_ and
    e.empid=s._PK_
  );
  /* Clean up. */
  execute (drop table mydb.employees) by teradata;
  execute (drop table mydb.employees_gend) by teradata;
  execute (drop table mydb.employees_pars) by teradata;
  execute (drop table mydb.employees_std) by teradata;
quit;
```

1 Note the use of the LIBNAME option MODE=TERADATA in the database connection statement. The data quality stored procedures require client programs to connect to the Teradata database with a database connection of session-mode Teradata. If this option is not specified, PROC SQL will connect in ANSI mode.

2 The DQ_SET_OPTIONS() stored procedure is executed to set the DQ_OVERWRITE_OPTION in the session. All output tables created with SAS Data Quality Accelerator stored procedures will be overwritten with new tables should any of the stored procedures be run again in the session.

3 The DQ_LIST_LOCALES() stored procedure is executed to identify the available locales.

4 The DQ_LIST_DEFNS() stored procedure is executed to list the gender, parse, and standardize definitions that are available for the ENUSA locale.

5 The DQ_GENDER() stored procedure call specifies to apply the QKB definition “Name” to data column Name. It specifies data column EmpID as the primary key. Stored procedure output is written to a table named MyDB.Employees_Gend.

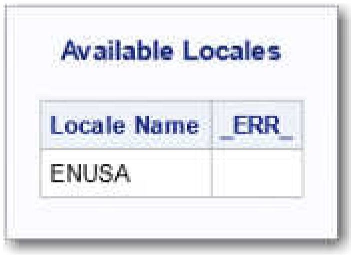
6 The DQ_PARSE() stored procedure call specifies to apply the QKB definition “Name” to data column Name as well. Although this definition has the same name as the definition specified in DQ_GENDER(), the content of the definitions is not the

same. The definitions contain different instructions. The stored procedure call specifies data column EmpID as the primary key. Stored procedure output is written to a table named MyDB.Employees_Pars.

7 The DQ_STANDARDIZE() stored procedure specifies to apply the QKB definition "Address" to data column Address. It specifies data column EmpID as the primary key. Stored procedure output is written to a table named MyDB.Employees_Stdz.

8 In the final step, a pass-through query is submitted to the Teradata database that specifies which columns from the source and stored procedure output tables to display and performs an equijoin on the primary key EmpID.

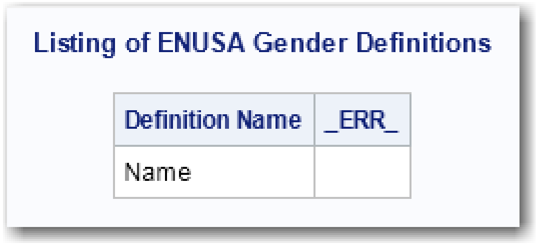
The following display shows the MyDB.Loclist table created by the DQ_LIST_LOCALES() stored procedure.



Available Locales

Locale Name	_ERR_
ENUSA	

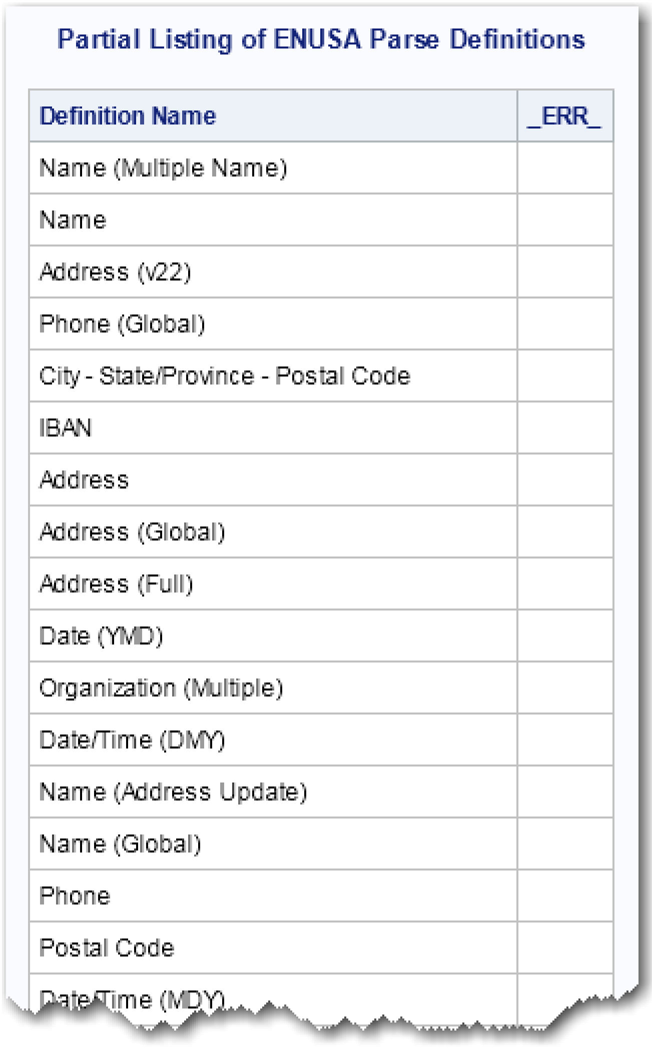
The following display shows the MyDB.Gendefs table created by the DQ_LIST_DEFNS() stored procedure.



Listing of ENUSA Gender Definitions

Definition Name	_ERR_
Name	

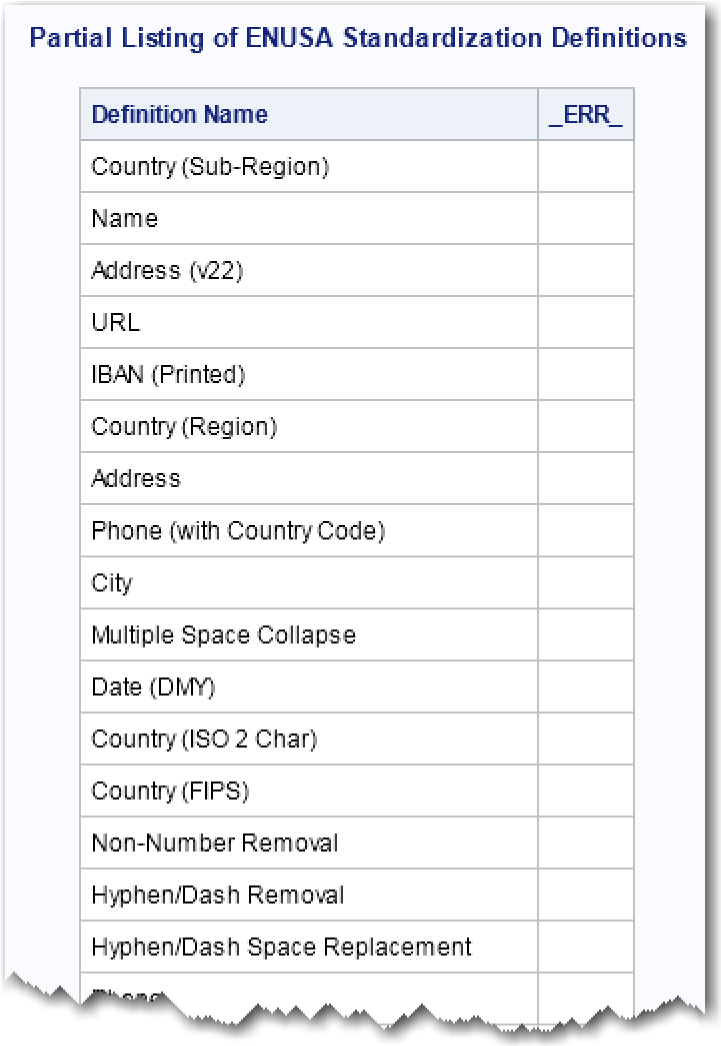
The following display shows a portion of the MyDB.Parsdefs table created by the DQ_LIST_DEFNS() stored procedure.



Partial Listing of ENUSA Parse Definitions

Definition Name	_ERR_
Name (Multiple Name)	
Name	
Address (v22)	
Phone (Global)	
City - State/Province - Postal Code	
IBAN	
Address	
Address (Global)	
Address (Full)	
Date (YMD)	
Organization (Multiple)	
Date/Time (DMY)	
Name (Address Update)	
Name (Global)	
Phone	
Postal Code	
Date/Time (MDY)	

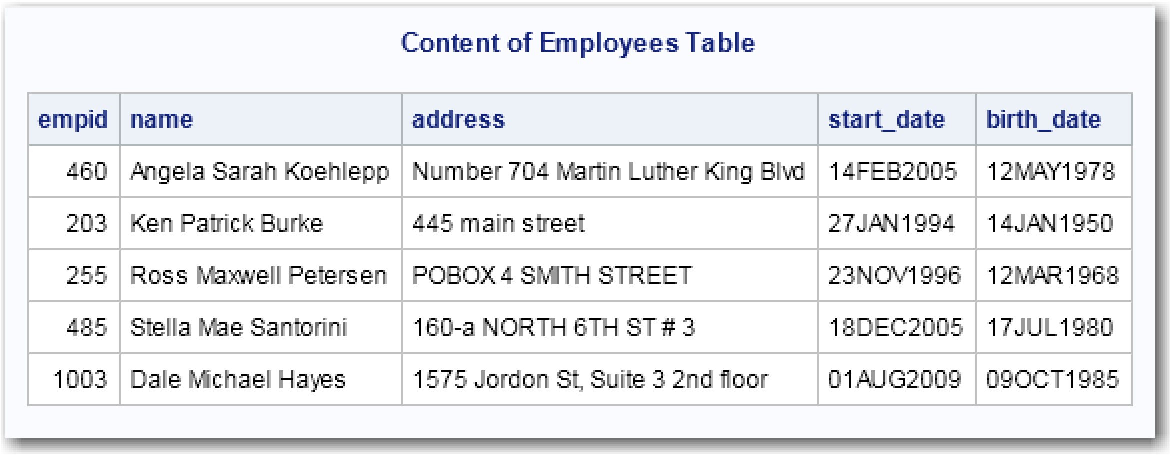
The following display shows a portion of the MyDB.Standdefs table created by the DQ_LIST_DEFNS() stored procedure.



Partial Listing of ENUSA Standardization Definitions

Definition Name	_ERR_
Country (Sub-Region)	
Name	
Address (v22)	
URL	
IBAN (Printed)	
Country (Region)	
Address	
Phone (with Country Code)	
City	
Multiple Space Collapse	
Date (DMY)	
Country (ISO 2 Char)	
Country (FIPS)	
Non-Number Removal	
Hyphen/Dash Removal	
Hyphen/Dash Space Replacement	

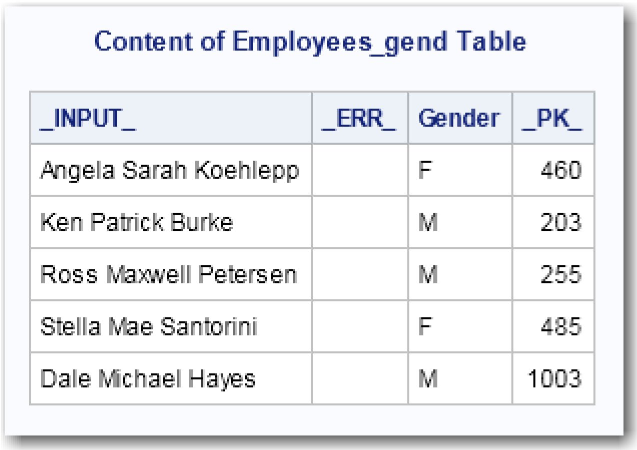
The following display shows contents of the MyDB.Employees table, before data cleansing:



Content of Employees Table

empid	name	address	start_date	birth_date
460	Angela Sarah Koehlepp	Number 704 Martin Luther King Blvd	14FEB2005	12MAY1978
203	Ken Patrick Burke	445 main street	27JAN1994	14JAN1950
255	Ross Maxwell Petersen	POBOX 4 SMITH STREET	23NOV1996	12MAR1968
485	Stella Mae Santorini	160-a NORTH 6TH ST # 3	18DEC2005	17JUL1980
1003	Dale Michael Hayes	1575 Jordon St, Suite 3 2nd floor	01AUG2009	09OCT1985

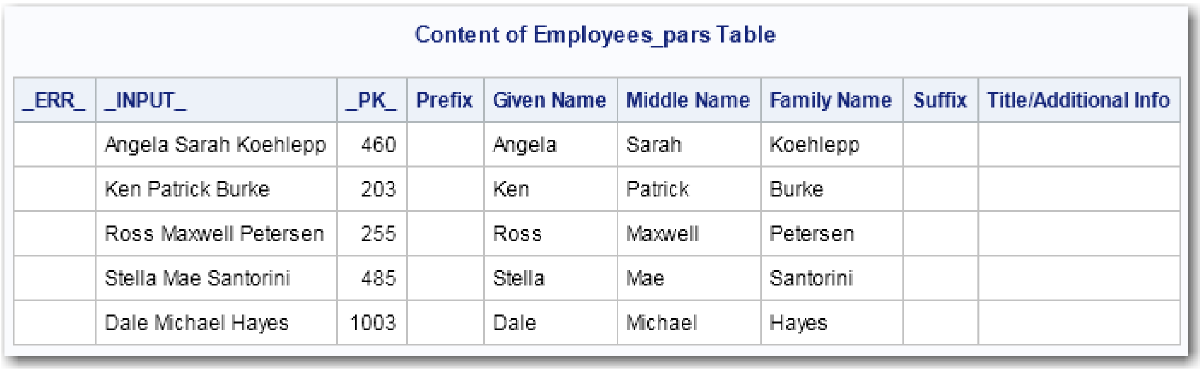
The following display shows the MyDB.Employees_Gend table:



Content of Employees_gend Table

INPUT	_ERR_	Gender	_PK_
Angela Sarah Koehlepp		F	460
Ken Patrick Burke		M	203
Ross Maxwell Petersen		M	255
Stella Mae Santorini		F	485
Dale Michael Hayes		M	1003

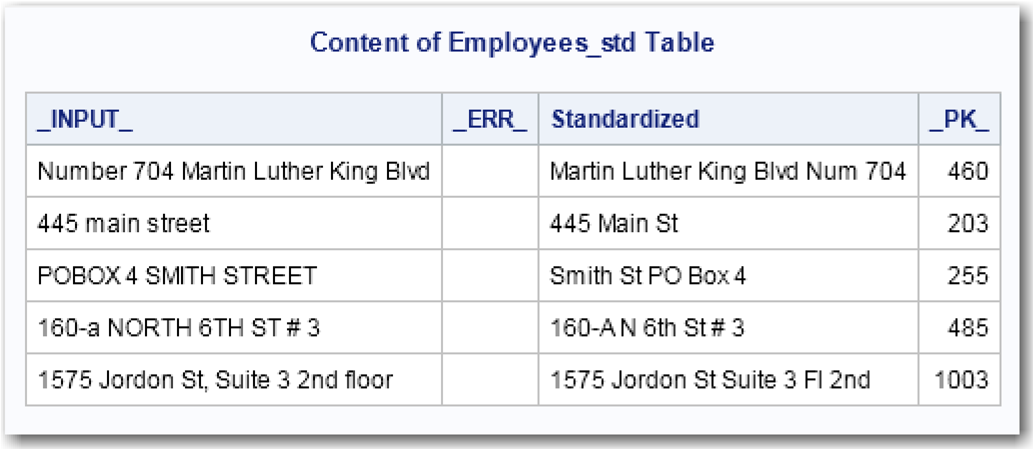
The following display shows the MyDB.Employees_Pars table:



Content of Employees_pars Table

ERR	_INPUT_	_PK_	Prefix	Given Name	Middle Name	Family Name	Suffix	Title/Additional Info
	Angela Sarah Koehlepp	460		Angela	Sarah	Koehlepp		
	Ken Patrick Burke	203		Ken	Patrick	Burke		
	Ross Maxwell Petersen	255		Ross	Maxwell	Petersen		
	Stella Mae Santorini	485		Stella	Mae	Santorini		
	Dale Michael Hayes	1003		Dale	Michael	Hayes		

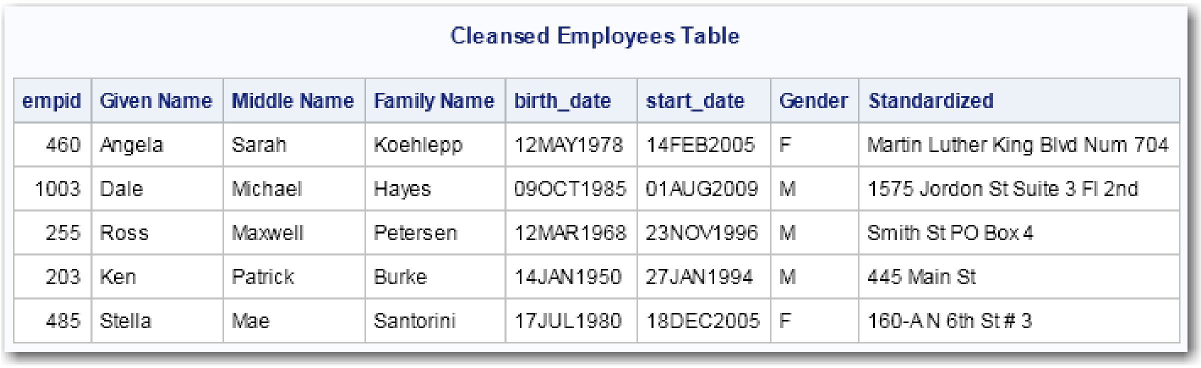
The following display shows the MyDB.Employees_Stdz table:



Content of Employees_std Table

INPUT	_ERR_	Standardized	_PK_
Number 704 Martin Luther King Blvd		Martin Luther King Blvd Num 704	460
445 main street		445 Main St	203
POBOX 4 SMITH STREET		Smith St PO Box 4	255
160-a NORTH 6TH ST # 3		160-A N 6th St # 3	485
1575 Jordon St, Suite 3 2nd floor		1575 Jordon St Suite 3 Fl 2nd	1003

The following display shows the content of the report after the data was cleaned with the data quality stored procedures:



Cleansed Employees Table

empid	Given Name	Middle Name	Family Name	birth_date	start_date	Gender	Standardized
460	Angela	Sarah	Koehlepp	12MAY1978	14FEB2005	F	Martin Luther King Blvd Num 704
1003	Dale	Michael	Hayes	09OCT1985	01AUG2009	M	1575 Jordon St Suite 3 Fl 2nd
255	Ross	Maxwell	Petersen	12MAR1968	23NOV1996	M	Smith St PO Box 4
203	Ken	Patrick	Burke	14JAN1950	27JAN1994	M	445 Main St
485	Stella	Mae	Santorini	17JUL1980	18DEC2005	F	160-A N 6th St # 3

Using Data Quality Stored Procedures in BTEQ

The following is BTEQ code that can be used to create and clean the content of the MyDB.Employees source table and create a report. BTEQ is a utility from the Teradata Utility Pack that allows users on a workstation to submit SQL statements to the Teradata database. This utility allows users to query, import, and export data from one or more Teradata RDBMS, and to produce reports. BTEQ stands for Basic Teradata Query. BTEQ connects to the Teradata database in Teradata mode by default.

```
.logon teraserver/myid,mypwd
.width 160

/* Set the option to overwrite output tables */
  call sas_sysfnlib.dq_set_option('DQ_OVERWRITE_TABLE', '1')

/* List the available locales. */
call sas_sysfnlib.dq_list_locales('mydb.loclist')

/* List the gender, parse, and standardization */
/* definitions available for the ENUSA locale. */
call sas_sysfnlib.dq_list_defns('ENUSA', 'gender','mydb.gendefs')
call sas_sysfnlib.dq_list_defns('ENUSA', 'parse','mydb.parsdefs')
call sas_sysfnlib.dq_list_defns('ENUSA', 'standardization','mydb.standdefs')

/* Create an example source table. */
create table mydb.employees
  (empid      integer primary key not null,
   name       varchar(50),
   address    varchar(100),
   start_date varchar(9),
   birth_date varchar(9));

/* Populate the source table with some data. */
insert into mydb.employees values
  (203,
   'Ken Patrick Burke',
   '445 main street',
   '27JAN1994',
   '14JAN1950');

insert into mydb.employees values
  (485,
   'Stella Mae Santorini',
   '160-a NORTH 6TH ST # 3',
   '18DEC2005',
   '17JUL1980');

insert into mydb.employees values
  (1003,
   'Dale Michael Hayes',
   '1575 Jordon St, Suite 3 2nd floor',
   '01AUG2009',
   '09OCT1985');

insert into mydb.employees values
  (460,
   'Angela Sarah Koehlepp',
   'Number 704 Martin Luther King Blvd',
   '14FEB2005',
   '12MAY1978');

insert into mydb.employees values
  (255,
   'Ross Maxwell Petersen',
   'POBOX 4 SMITH STREET',
   '23NOV1996',
   '12MAR1968');

/* Determine gender of employees in the source table. */
call sas_sysfnlib.dq_gender_loc(
  'Name',
  'mydb.employees',
  'name',
  'empid',
  'mydb.employees_gend',
  'ENUSA');

/* Run parse to split the first, middle, and last name. */
call sas_sysfnlib.dq_parse_loc(
  'Name',
  'mydb.employees',
  'name',
  'empid',
  'mydb.employees_pars',
  'ENUSA');

/* Standardize the employees' addresses. */
call sas_sysfnlib.dq_standardize_loc(
  'Address',
  'mydb.employees',
  'address',
  'empid',
  'mydb.employees_std',
  'ENUSA');
```

```
/* Aggregate the processed data into a report. */
select e.empid                               as "Employee ID",
       cast (p."Given Name"  as varchar(15)) as "Given Name",
       cast (p."Middle Name" as varchar(15)) as "Middle Name",
       cast (p."Family Name" as varchar(15)) as "Family Name",
       e.birth_date                          as "Birth Date",
       e.start_date                          as "Start Date",
       g.Gender                              as "Gender",
       cast (s.Standardized as varchar(30))  as "Address"
  from mydb.employees      e,
       mydb.employees_gend g,
       mydb.employees_pars p,
       mydb.employees_std s
where e.empid=g._PK_ and
       e.empid=p._PK_ and
       e.empid=s._PK_;

/* Clean up. */
drop table mydb.employees;
drop table mydb.employees_gend;
drop table mydb.employees_pars;
drop table mydb.employees_std;

.logoff
.quit
```

Chapter 6 Troubleshooting

Troubleshooting Problems with the Data Quality Stored Procedures

The data quality stored procedures can fail if one or more of the following are true:

- You do not have permission to use the stored procedures. Before you can use the data quality stored procedures, the Teradata administrator must grant you permission. If you receive a message similar to the following when you attempt to execute a stored procedure, you do not have permission. Contact your Teradata administrator to request permission.

  ```
  ERROR: Teradata execute: DQI_CALL_EP:The user does not have
  DELETE access to SAS_SYSFNLIB.dqt_codegen
  ```

- The request specifies an output location to which you do not have Write permission. Verify that you have access to the database that is specified in the output_table parameters.
- Your SQL request does not use the Teradata dialect. The stored procedures are invoked with the CALL keyword from any product that supports the Teradata SQL dialect. When you submit the data quality stored procedures in the SAS SQL procedure using explicit pass-through, the database connection is made in ANSI mode by default. You must specify the MODE= option in the Teradata LIBNAME statement to switch to Teradata mode. Consult the SAS/ACCESS Interface to Teradata documentation for more information about the MODE= option. Consult appropriate documentation for how to set Teradata mode in other client programs.

Recommended Reading

Here is the recommended reading list for this title:

- *SAS In-Database Products: Administrator's Guide*
- *SAS In-Database Products: User's Guide*
- *SAS Data Quality Server: Reference*
- *SAS/ACCESS for Relational Databases: Reference*
- *DataFlux Data Management Studio: User's Guide*
- SAS Quality Knowledge Base documentation that is available from SAS Documentation:
  - What's New in SAS Quality Knowledge Base
  - SAS Quality Knowledge Base Online Help

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books

Index