ODBC Driver for Teradata®

User Guide

Release 16.20

October 2018

B035-2526-108K
DOCS.TERADATA.COM

Copyright and Trademarks

Product Safety

Safety type	Description
NOTICE	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
⚠ CAUTION	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
⚠ WARNING	Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

Warranty Disclaimer

Except as may be provided in a separate written agreement with Teradata or required by applicable law, the information available from the Teradata Documentation website or contained in Teradata information products is provided on an "as-is" basis, without warranty of any kind, either express or implied, including the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.

The information available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The information available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in this information at any time without notice.

Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: teradata-books@lists.teradata.com

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

Contents

1

Overview

Introduction

This user guide contains information for the Windows, UNIX/Linux, and Apple OS X versions of ODBC Driver for Teradata. Unless a paragraph or section is specifically marked as Windows only, UNIX system only, or Apple OS X only, information pertains to the drivers for all supported operating systems.

ODBC standards have been developed through the efforts of an industry consortium called the SQL-Access Group (SAG), which consists of vendors that include Teradata, Hewlett-Packard®, Ingres, Oracle®, Informix, Sybase®, Microsoft®, and Novell®. The standards established by SAG define common database access mechanisms to simplify the exchange of data between client and server.

ODBC drivers connect applications with databases. This book describes how ODBC Driver for Teradata interfaces with Teradata Database. ODBC Driver for Teradata conforms to standard ODBC 3.8 Core-Level specifications.

Prerequisites

The following prerequisite knowledge is required for this product:

- Windows, UNIX, or Apple OS X concepts and commands
- ANSI Structured Query Language (SQL)
- ODBC and Teradata SQL
- ODBC Programmer's Reference

Supported Releases

This book supports the following releases:

- Teradata Database 16.20
- Teradata Tools and Utilities 16.20
- ODBC Driver for Teradata 16.20

Note:

Please note the following as you use this manual:

- The term "old" is used to refer to Teradata ODBC Driver, version 16.10 or earlier. For example, "old Teradata ODBC driver".
- The term "new" is used to refer to Teradata ODBC Driver, version 16.20 or newer. For example, "new driver".

Note:

See Determining the Installed Versions of ODBC Driver for Teradata to verify the ODBC Driver for Teradata version number.

Supported operating systems, supported Teradata Database versions, and product version numbers for all Teradata Tools and Utilities are available (at http://www.info.teradata.com) in a single spreadsheet titled *Teradata Tools and Utilities Supported Platforms and Product Versions* (B035-3119).

Key Changes in Product Behavior

The 16.20 version of Teradata ODBC Driver has a new code base, and many of the features found in previous versions have been completely redesigned. The new driver does not include all of the deprecated features found in previous versions of the Teradata ODBC Driver, and these features have been removed. For more information about the new driver and how its functionality compares to the previous versions of the Teradata ODBC Driver, see Deprecated Features for New Teradata ODBC Driver.

This version of ODBC Driver for Teradata includes the following characteristics:

- Deprecated features from the Teradata ODBC Driver are no longer functional and are not supported in the new driver. There are no workarounds for these procedures, as they are obsolete. For a complete list of obsoleted features which do not work in the new driver, see Deprecated Features for New Teradata ODBC Driver.
- Starting with version 15.10.01.x, the Side-by-Side feature requires unique driver names for DSN-less connecting Applications in order to reference the different versions of drivers that may be installed. For more information, see MultiVersion Support.
- ODBC install is self-contained and the installation of other Teradata Tools and Utilities products such as *Teradata ICU, Call-Level Interface* (CLI), and *TeraGSS* is not required.
- Installing the ODBC Driver no longer automatically migrates DSNs.

  DSNs belonging to the older versions of the driver will not be converted to the newly installed version of the ODBC Driver. See *Driver Migration* below for more information.
- Supports the use of unixODBC and iODBC driver managers. This is in addition to existing support of Progress DataDirect Driver Manager.

- Teradata ODBC Driver strictly adheres to ODBC specifications. Non-compliant ODBC functions found in old versions of TD ODBC driver are not supported by the new driver.
- As an important example of adherence to the ODBC specification, you must close the cursor before executing another query or else a cursor error is returned. Previously, you were able to bypass closing the cursor but this is not the correct behavior. This and other differences can be found in New Teradata ODBC Driver Compatibility Reference.
- Teradata ODBC driver conforms to standard ODBC 3.8 core level specifications.

Differences in Driver Implementation

There are differences between the old Teradata ODBC Driver (16.10 and older) and the new driver's SLOB implementation. The new driver includes the following changes for SLOB implementation:

- Caches, or attempts to cache, up to 2GB of all SLOBs in a row.
- Three configuration parameters:
    - 1-Enable SLOB Random Access
    - 2-Max size of one SLOB
    - 3-Max size of all SLOBs in a row

For more information, and a detailed list of features in the new driver, refer to New Teradata ODBC Features.

Driver Migration

If you are migrating to the new driver from an old Teradata ODBC Driver (16.10 or older), update the **Driver** key to point to the newly installed driver.

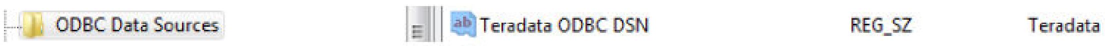
Note:

If a user installs the same version of new and old drivers, the current ODBC Driver installation overwrites existing DSNs. Before a new ODBC Driver installation, make backup copies of DSNs first and then proceed with the installation. Once installation is complete, copy back the old DSNs.

On Windows, if the desired driver is `Teradata ODBC DSN:`

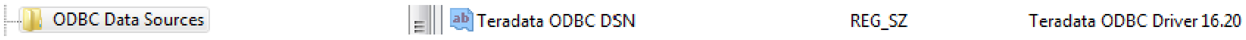
1. Change the key value under `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI\ODBC Data Sources` from **Teradata** to **Teradata ODBC Driver 16.xx** as shown in the figures below.

   From



ODBC Data Sources | Teradata ODBC DSN | REG_SZ | Teradata

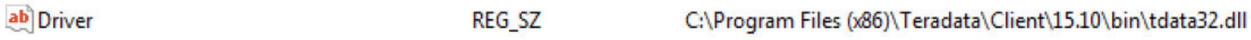
To:



ODBC Data Sources | Teradata ODBC DSN | REG_SZ | Teradata ODBC Driver 16.20

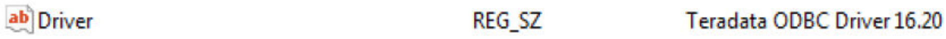
2. Also change the key `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBC.INI\Teradata ODBC DSN\Driver` from `C:\Program Files (x86)\Teradata\Client\15.10\bin\tdata32.dll` to **Teradata ODBC Driver 16.xx** as shown in the figures below.

   From:

   Driver | REG_SZ | C:\Program Files (x86)\Teradata\Client\15.10\bin\tdata32.dll

   To:

   Driver | REG_SZ | Teradata ODBC Driver 16.20

   The DSN now points to the new driver.

New Driver Manager Support on Unix

The new Teradata ODBC Driver, starting with version 16.20, supports the industry standard iODBC Driver Manager and unixODBC Driver Manager on Unix platforms, in addition to the DataDirect Driver Manager.

Be aware that different driver managers may have different behaviors within the standards of ODBC. With the new Teradata ODBC Driver, this same driver can be used with any of the driver managers mentioned above. Applications and libraries no longer need to be rebuilt for specific Driver Managers, as the new driver now works with more driver managers. Some of the benefits of these driver managers are listed below.

Many applications are written to be used with common driver managers such as iODBC and unixODBC.

Performance gains in reopening connections using the Driver Managers' connection pooling is available through iODBC and unixODBC. See the respective driver manager documentation for more information.

Note:

> Be aware there are security risks with using connection pooling. For more information, see [Connection Pooling (Windows and Apple OS X)](#).

Also, both of these driver managers are open source and freely available, making them very popular. You can take advantage of these driver managers, which means it will be easier to connect to Teradata through languages like Perl, Python, PHP, R and so on.

Tools built to connect these languages to an ODBC driver were typically designed with the unixODBC Driver Manager or iODBC Driver Manager in mind. With the new Teradata ODBC driver, these no longer need to be rebuilt purposely for the DataDirect Driver Manager in order to access the Teradata database.

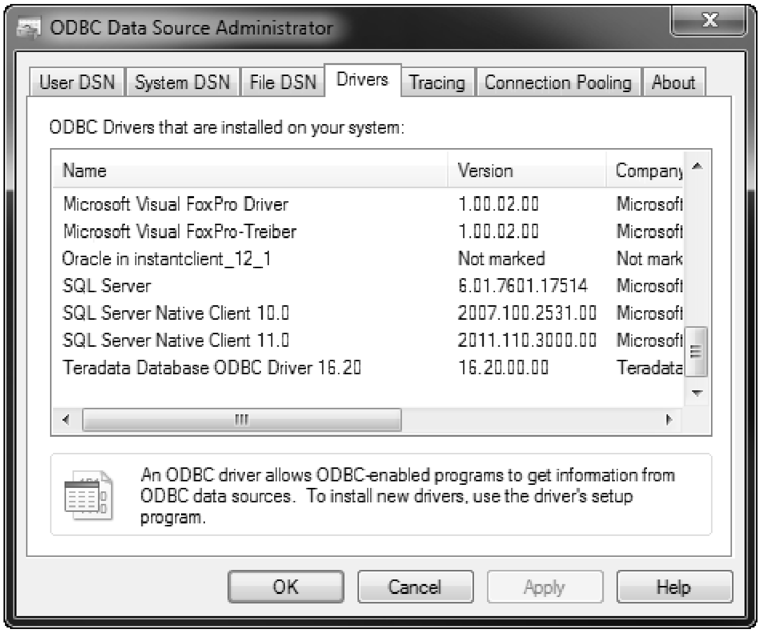
If you need more information about connecting with Perl, Python, and other supported languages, visit the Teradata Community at . Use keywords like ODBC PERL or ODBC Python to search on how to use Perl or Python through the new Teradata ODBC driver.

For installation and use of iODBC and unixODBC driver managers, see [Configuration for UNIX/Linux Systems](#).

Determining the Installed Versions of ODBC Driver for Teradata

Windows

To determine the currently installed versions of ODBC Driver for Teradata on Windows, open 32-bit ODBC Administrator or 64-bit ODBC Administrator. Select the Drivers tab and find the entries for Teradata Database ODBC Driver. The driver versions will be displayed.



ODBC Drivers that are installed on your system:

Name	Version	Company
Microsoft Visual FoxPro Driver	1.00.02.00	Microsoft
Microsoft Visual FoxPro-Treiber	1.00.02.00	Microsoft
Oracle in instantclient_12_1	Not marked	Not mark
SQL Server	6.01.7601.17514	Microsoft
SQL Server Native Client 10.0	2007.100.2531.00	Microsoft
SQL Server Native Client 11.0	2011.110.3000.00	Microsoft
Teradata Database ODBC Driver 16.20	16.20.00.00	Teradata

An ODBC driver allows ODBC-enabled programs to get information from ODBC data sources. To install new drivers, use the driver's setup program.

Linux/UNIX and Apple OS X Systems

To determine the currently installed version(s) of ODBC Driver for Teradata on a Linux, UNIX or Apple OS X system, use a command from the following table.

Operating System	Command
Oracle Solaris	`pkginfo -l | grep tdodbc`
IBM® AIX	`lslpp -R ALL -Lc | grep tdodbc`
Red Hat Linux	`rpm -qa | grep tdodbc`
SUSE Linux	`rpm -qa | grep tdodbc`
Apple OS X	TTU ListProducts application located in **Applications > Teradata Client 16.20 > ListProducts**
Ubuntu	`dpkg -l "tdodbc*" | grep ^ii`

Certification and Release Information

To view the releases supported by this document, see Supported Releases.

The new driver will replace the old Teradata ODBC driver as part of the TTU suite and ODBC mini-suite.

UNIX and Linux Information

This section includes directory, variable, library, and migration features specific to UNIX and Linux operating systems.

Teradata Tools and Utilities Directory Layout

The Teradata Tools and Utilities release directory is the combination of the user specified base directory, the Teradata Tools and Utilities fixed directory and the Teradata Tools and Utilities release directory. The installation package allows the user to select a starting location where a Teradata Tools and Utilities release will be installed. The default location is `/opt`. The tdodbc installation package includes both the 32-bit and 64-bit ODBC Driver.

For information about installing operating system-specific utilities, see the documentation listed in Related Documentation.

Changes to Runtime Environment Variable Settings

TTU products no longer depend on environment variables. For IBM AIX, only one version can be active, either 32-bit or 64-bit. A script is available to change 64-bit to 32-bit, and vice versa: `setusrliblinks.sh.` The script is installed in `<prefix>/teradata/client/16.20/`

`odbc_32.` To use the 32-bit ODBC driver, use **`setusrliblinks.sh.`** The 64-bit driver is enabled by default on a 64-bit system.

32-bit	use script **`setusrliblinks.sh.`** at `<prefix>/teradata/client/16.20/odbc_32.`
32-bit, 64-bit	.env files at `<prefix>/teradata/client/16.20/odbc_32` or `<prefix>/teradata/client/16.20/odbc_64.`
64-bit	Ubuntu at: `/usr/lib32` and `/usr/lib` (64-bit system libraries).

System Library and Driver Directories

This section provides information about system libraries and drivers.

Operating System	32-bit	64-bit
Linux	`/usr/lib`	`/usr/lib64`
IBM AIX	`/usr/lib`	`/usr/lib`
Oracle Solaris on SPARC systems	`/usr/lib`	`/usr/lib/sparcv9`
Oracle Solaris on AMD Opteron systems	`/usr/lib`	`/usr/lib/amd64`
Ubuntu	`/usr/lib32`	`/usr/lib` (64-bit system libraries)

ODBC Driver Library

This section provides information about library references in operating system directories. This is accomplished using symbolic links to the actual library.

tdodbc (product)

- `libodbc.so` (or .a on AIX)
- `libodbcinst.so`
- `libivicu27.so` (32-bit)
- `libddicu27.so` (64-bit)

Simbaodbc (product)

- `tdataodbc_sb32.so` (32-bit)
- `tdataodbc_sb64.so` (64-bit)

ODBC Integrated Directories

ODBC contains release-independent directories which are denoted by ODBC_32 and ODBC_64.

Their sub-directories are symbolic links to the respective 32-bit and 64-bit Teradata Tools and Utilities release directories installed during the installation of the current ODBC.

ODBC_32 and ODBC_64 contain their respective **`odbc.ini`** and **`odbcinst.ini`** files which are free of any use of a Teradata Tools and Utilities release path. It is required that these templates be used when setting up the **`odbc.ini`** with the desired DSN settings. Otherwise, the user will be continuously updating the **`.ini`** files when different Teradata Tools and Utilities releases are installed. In particular, the path values in the odbc.ini template for InstallDir and Driver must be the ones defined in the template; they cannot be modified.

Whenever a new Teradata Tools and Utilities release is installed, the sub-directories are updated with the new symbolic links. This release will be the Active TTU of the system. See MultiVersion Support for more information.

The following table shows the release-independent directories.

On UNIX or Linux systems, the Teradata Tools and Utilities installation includes an option to uninstall previous versions of existing Teradata Tools and Utilities software. Any Teradata Tools and Utilities release version prior to 15.10.01 must be uninstalled before installing ODBC Driver for Teradata. This does not apply to efixes, because efixes are installed as an upgrade.

For information about installing operating system-specific utilities, see the documentation listed on http://www.info.teradata.com. Depending on the Teradata Tools and Utilities version installed, the actual value represented by <***`ttu version`***> might display as the version number.

`/opt`	`teradata/`
	`client/`
		`ODBC_32/`
		`include -> /opt/teradata/client/<TTU version>/include\`
		`lib -> /opt/teradata/client/<TTU version>/lib\`
		`locale -> /opt/teradata/client/<TTU version>/odbc_32/locale\`
		`odbc.ini`
		`odbcinst.ini`

`/opt`	`teradata/`
	`client/`
		`ODBC_64/`
		`include -> /opt/teradata/client/<TTU version>/include`
		`lib -> /opt/teradata/client/<TTU version>/lib64`
		`locale -> /opt/teradata/client/<TTU version>/odbc_64/locale\`
		`odbc.ini`
		`odbcinst.ini`

Apple OS X Information

This section describes the directory tree structure on Apple OS X.

Pathname	Description
`/Library/Application Support/teradata/client/ODBC/lib/libtdsso.dylib`	symbolic link to `'/Library/Application Support/teradata/client/<TTU version>/lib/libtdsso.dylib`'
`/Library/Application Support/teradata/client/ODBC/lib/tdataodbc_sbu.dylib`	symbolic link to `'/Library/Application Support/teradata/client/16.20/lib`
`/Library/Application Support/teradata/client/ODBC/lib/TeradataODBCSetup.bundle`	symbolic link to `'/Library/Application Support/teradata/client/<TTU version>/lib/TeradataODBCSetup.bundle'`
`/Library/Application Support/teradata/client/ODBC/lib/TdConnectionDialog.bundle`	symbolic link to `'/Library/Application Support/teradata/client/<TTU version>/lib/TdConnectionDialog.bundle'`
`/Library/Application Support/teradata/client/ODBC/include`	symbolic link to `'/Library/Application Support/teradata/client/<TTU version>/include'`
`/Library/Application Support/teradata/client/ODBC/lib`	symbolic link to `'/Library/Application Support/teradata/client/<TTU version>/lib'`
`/Library/Application Support/teradata/client/<TTU version>/odbc/bin/tdxodbc`	ODBC Connection test tool

Pathname	Description
`/Library/Application Support/teradata/client/<TTU version>/include/tdsql.h`	Teradata ODBC Driver custom definitions
`/Library/Application Support/teradata/client/<TTU version>/lib/libtdsso.dylib`	Teradata SSO library
`/Library/Application Support/teradata/client/<TTU version>/lib/tdataodbc_sbu.dylib`	Teradata ODBC driver
`/Library/Application Support/teradata/client/<TTU version>/odbc/lib/TdConnectionDialog.bundle`	Teradata ODBC Driver connection dialog
`/Library/Application Support/teradata/client/<TTU version>/lib/TeradataODBCSetup.bundle`	Teradata ODBC Setup dialog
`/Library/Application Support/teradata/client/<TTU version>/odbc/odbc.ini`	Sample `odbc.ini` file
`/Library/Application Support/teradata/client/<TTU version>/odbc/README`	README
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C/adhoc`	Sample ODBC application binary
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C/adhoc.c`	Sample ODBC application code
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C/common.includes`	Sample ODBC application code
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C/error.c`	Sample ODBC application code
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C/Makefile`	Sample ODBC application code
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C++/adhoc`	Sample ODBC application binary
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C++/adhoc.cpp`	Sample ODBC application code

Pathname	Description
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C++/common.includes`	Sample ODBC application code
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C++/error.cpp`	Sample ODBC application code
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples/C++/Makefile`	Sample ODBC application code
`/Library/Application Support/teradata/client/ODBC/odbc.ini`	Sample `odbc.ini` file

2

Configuration for UNIX/Linux Systems

Overview

This chapter provides information on data sources and variables used with the UNIX OS version of ODBC Driver for Teradata and how to configure the driver using the `odbc.ini` file.

For information about configuring the Windows version of ODBC Driver for Teradata, see Configuration for Windows.

For information about configuring ODBC Driver for Teradata for Apple OS X, see Configuration for Apple OS X.

ODBC Directories

During installation on 64-bit systems, ODBC directories are created for 32-bit, 64-bit, and their respective release-independent directories. On 32-bit systems, only 32-bit directories are created. The 32-bit and 64-bit directories are placed under the Teradata Tools and Utilities release directory. Release independent directories are created at the same directory level as the Teradata Tools and Utilities release directories. Release independent directories provide a way to set up the `odbc.ini` and `odbcinst.ini` files without having to change them when future Teradata Tools and Utilities releases are installed.

Subdirectories for the 32-bit and 64-bit directories are listed in the following table.

Directory	Contents
`bin`	Home for executables and scripts used to manage and support the ODBC Driver for Teradata
`include`	Driver Manager and ODBC Driver for Teradata include files
`lib`	Driver Manager and ODBC Driver for Teradata libraries
`locale`	Driver Manager message file
`ErrorMessages`	Teradata ODBC Driver message catalog
`samples`	C and C++ sample application

The release-independent directories are given the upper case names ODBC_32 and ODBC_64. Their sub-directories are symbolic links to their respective 32-bit and 64-bit Teradata Tools and Utilities release directories. ODBC_32 and ODBC_64 installations contain their respective `odbc.ini` and `odbcinst.ini` files which are free of any use of a Teradata Tools and Utilities release path. It is required that these templates be used when setting up the `odbc.ini` with the desired DSN settings. Otherwise, the user will be continuously updating the `.ini` files when different Teradata Tools and Utilities releases are installed. In particular, the path values in the odbc.ini template for InstallDir and Driver must be the ones defined in the template; they cannot be modified.When a new Teradata Tools and Utilities release is installed, the sub-directories are updated with new symbolic links.

Note:

> If there are multiple versions of the driver (such as 16.00 and 16.10) installed on the system, then the symbolic links will be pointing to whichever version was installed most recently.

Coexistence of Different Version Drivers on the Same Machine

For ODBC Driver for Teradata starting with 15.10.10.00, for UNIX bundle, Windows, and OS X Suites, multiple releases can co-exist on the same system. For example, 15.10.10.00, 16.10.00.00 and later versions can co-exist on the system at the same time.

If a specific version of the driver that is not the active TTU version is needed by an application, then the ODBCINI environment variable must be set to `<InstallDir>/Teradata/client/<desired version>/odbc_32/odbc.ini` for 32 bit or `<InstallDir>/Teradata/client/odbc_64/odbc.ini` for 64-bit, and that odbc.ini file must be updated accordingly.

.env Files

Another option is to use the .env files provided for each OS. The .env files are located in `<prefix>/teradata/client/16.20/<etc>`.

For UNIX and OS X:

- ttu_1620_bash.env
- ttu_1620_csh.env

For AIX:

- ttu32_1620_bash.env
- ttu32_1620_csh.env
- ttu64_1620_bash.env
- ttu64_1620_csh.env

The .env file can be sourced in a terminal window so that release is active in that session.

Verifying the TCP/IP Connection

Verify that the client system can connect over TCP/IP to the server using a domain name server (DNS) or a local `hosts` file. Use the `tdxodbc` or `tdxodbc64` program in the `<InstallDir>/Teradata/client/<TTU version>/bin` directory to verify the connection and successful loading of the Teradata ODBC driver.

The local `hosts` file contains the names and IP addresses of system nodes that support the Teradata network communication interface.

Configuring a UNIX System

NOTICE

It is strongly recommended you do not update LIBPATH.

The driver provides a script to switch between 64-bit and 32-bit. As another option, the .env file can be sourced for 32-bit or 64-bit.

Setting Additional Environment Variables (optional)

The following table lists additional optional environment variables.

Variable Name	Function
ODBCINI	Specifies a different pathname for the `odbc.ini` file
LANG	Enables a supported character set to work with ODBC Driver for Teradata

Setting LANG

To enable Kanji, Chinese, or Korean support, the LANG environment variable must be set to the appropriate character setting. For example, use the following to set the language for Japanese:

```
export LANG=japan
```

Note:

Please refer to your Unix system for supported locales and correct values.

For the UTF8 session character set, set the environment variables to UTF8 for the UTF8-based text processing.

- setenv LANG en_US.UTF8
- setenv LC_ALL en_US.UTF8

For additional information about international language support, see International Character Set Support.

Configuring DSN in odbc.ini

For information on configuring a DSN, see Configuration of odbc.ini in UNIX/Linux and Apple OS X.

iODBC and unixODBC Driver Manager Installation

The Teradata installer comes with the DataDirect driver manager installed, but you can follow the steps in the procedures below to use a different driver manager.

- Installing your Chosen Driver Manager
- Selecting the Chosen Driver Manager

Installing your Chosen Driver Manager

1. Download your driver manager of choice from their official website.
   Typical driver managers used on Unix are iODBC or unixODBC.

   Note:

   iODBC is the choice made by Apple for their Mac Operating System, and unixODBC is a popular choice on Unix systems.

   The web sites are provided below:

   - http://www.iodbc.org
   - http://www.unixodbc.org

2. Based on your application bitness, you need to match the bitness (32- or 64-bit) of your installed Teradata ODBC driver and the Driver Manager bitness you install.
   a. Download the correct bitness or compile the source for the right bitness (typically based on the CFLAGS settings).

      Note:

      You can have both a 32-bit and 64-bit version of the driver manager installed. They default to different locations and your odbc.ini, and odbcinst.ini should reference them correctly.

Tip:

When using your application or testing the connection, if you encounter errors related to “ELF”, it is likely due to you mismatching the bitness between application, driver manager and/or Teradata ODBC driver.

You can typically check the bitness of a file on Unix systems by using the “file” command in a shell, and pass in the filename with path if needed.

3. Install your driver manager.
   You may need root access depending on where you install. Consult the driver manager's documentation for more information.

Selecting the Chosen Driver Manager

1. Update your LD_LIBRARY_PATH with a path to the Driver Manager chosen.
   For example, if you installed the Driver Manager in `/usr/local/lib`, you can run the following command to set the LD_LIBRARY_PATH for the current user session:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

iODBC and unixODBC Driver Manager Connection Testing

To test the connection, you can use an ODBC-enabled client application. For a basic connection test, you can also use the test utilities that are packaged with your driver manager installation.

For example, the iODBC driver manager includes simple utilities called iodbctest and iodbctestw. Similarly, the unixODBC driver manager includes simple utilities called isql and iusql.

Using the iODBC Driver Manager

You can use the iodbctest and iodbctestw utilities to establish a test connection with your driver. Use iodbctest to test how your driver works with an ANSI application, or use iodbctestw to test how your driver works with a Unicode application.

Note:

There are 32-bit and 64-bit installations of the iODBC driver manager available. If you have only one or the other installed, then the appropriate version of iodbctest (or iodbctestw) is available. However, if you have both 32- and 64-bit versions installed, then you need to make sure that you are running the version from the correct installation directory.

For more information about using the iODBC driver manager, see http://www.iodbc.org.

Testing Your Connection Using the iODBC Driver Manager

1. Run iodbctest or iodbctestw.

   Tip:

   Optionally, if you do not remember the DSN, then type a question mark (?) to see a list of available DSNs.

   ---

2. Type the connection string for connecting to your data store, and then press **Enter**.
   If the connection is successful, then the SQL> prompt appears.

Using the unixODBC Driver Manager

You can use the isql and iusql utilities to establish a test connection with your driver and your DSN. isql and iusql can only be used to test connections that use a DSN.

Use isql to test how your driver works with an ANSI application, or use iusql to test how your driver works with a Unicode application.

Note:

There are 32-bit and 64-bit installations of the unixODBC driver manager available. If you have only one or the other installed, then the appropriate version of isql (or iusql) is available. However, if you have both 32- and 64-bit versions installed, then you need to make sure that you are running the version from the correct installation directory.

For more information about using the unixODBC driver manager, see .

Testing Your Connection Using the unixODBC Driver Manager

1. Run isql or iusql by using the corresponding syntax:
   - `isql [DataSourceName]`
   - `iusql [DataSourceName]`

   where *`[DataSourceName]`* is the DSN that you are using for the connection.

   If the connection is successful, then the SQL> prompt appears.

   Note:

   For information about the available options, run isql or iusql without providing a DSN.

3

Configuration for Windows

Overview

This chapter provides instructions for configuring a data source and ODBC Driver for Teradata on Windows systems

For information about the UNIX system versions of ODBC Driver for Teradata, see Configuration for UNIX/Linux Systems.

For information about ODBC Driver for Teradata for Apple OS X, see Configuration for Apple OS X.

Windows ODBC Driver Directories

ODBC Driver for Teradata is installed on Windows following the instructions from *Teradata Tools and Utilities Installation Guide for Microsoft Windows* (B035-2407).

During installation of ODBC Driver for Teradata, the following directories are created:

```
<user_defined>\Teradata\Client\<version>\ODBC Driver for Teradata\
\ErrorMessages
\Help
\Samples
Readme.txt
tdodbcdsn.vbs
tdsql.h
```

`<user_defined>\Teradata\Client\<version>\bin\`

`libcrypto-1_1-x64.dll` or `libcrypto-1_1.dll`
`sbicudt53_64.dll` or `sbicudt53_32.dll`
`sbicuin53_64.dll` or `sbicuin53_32.dll`
`sbicuuc53_64.dll` or `sbicuuc53_32.dll`
`tdataodbc_sb64.dll` or `tdataodbc_sb32.dll`
`tdclientdir`
`tdxodbc`
`TeradataODBC.did`
`TeradataODBC64.man` or `TeradataODBC32.man`
`terasso.dll`

Configuring a Data Source

For ODBC Driver for Teradata, both 32-bit and 64-bit drivers can coexist on the same system.

For Windows 10 and Windows 2016, the location of the 32-bit and 64-bit ODBC Data Sources start menu item is located in *Windows Administrative Tools* for ODBC Data Sources (32-bit) and ODBC Data Sources (64-bit).

To set up a data source name (DSN) for each type of configuration:

- To set up DSNs for a 32-bit driver, open the 32-bit **ODBC Data Source Administrator** dialog box by selecting **Start > Programs > ODBC > 32-Bit ODBC Administrator**. Then follow the instructions from Configuring a Data Source using ODBC Data Source Administrator.
- To set up DSNs for a 64-bit driver, open the 64-bit **ODBC Data Source Administrator** dialog box by selecting **Start > Programs > ODBC > 64-Bit ODBC Administrator** Then follow the instructions from Configuring a Data Source using ODBC Data Source Administrator.

Note:

> The ODBC Data Source Administrator does not distinguish between 32-bit and 64-bit user DSNs as they both reside in the same location of the Windows registry.

Coexistence of Different Version Driver on the Same Machine

For ODBC Driver for Teradata starting with 15.10.10.00, for UNIX bundle, Windows, and OS X Suites, multiple releases can co-exist on the same system. For example, 15.10.10.00, 16.10.00.00 and later versions can co-exist on the system at the same time.

To set up a data source name (DSN) for the desired version, follow the instructions from Configuring a Data Source using ODBC Data Source Administrator.

Configuring a Data Source using ODBC Data Source Administrator

Use the **ODBC Data Source Administrator** dialog box to configure data sources for each Teradata system to be accessed. One or more data sources must be configured.

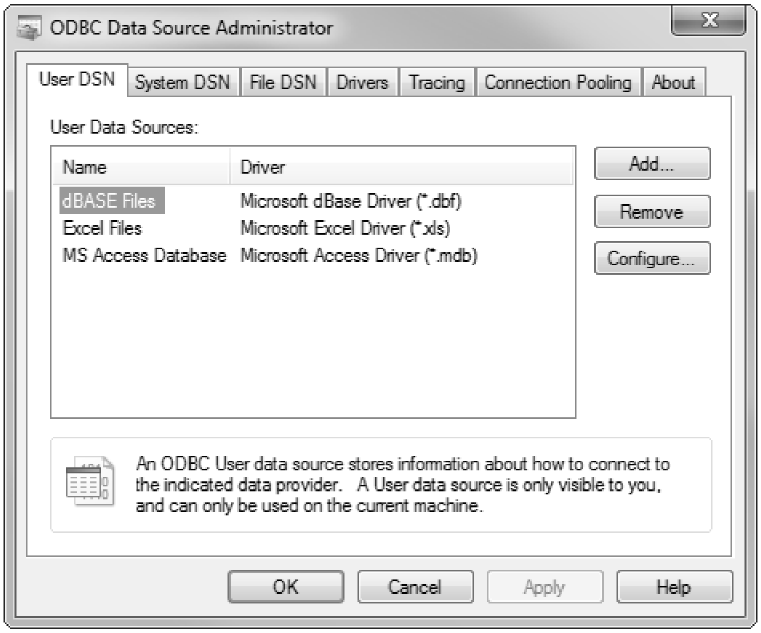
ODBC Driver for Teradata has established specific application uses for available DSN settings.

For DSN settings and their specific application use, see ODBC Driver for Teradata Application Development.

For information on disabling the ability to save the password in a DSN, see Security Considerations.

To configure a data source:

1. Open the Control Panel and navigate to **Control Panel > System and Security > Administrative Tools**.
2. In the Administrative Tools window, double-click **ODBC Data Sources** to open the **ODBC Data Source Administrator** dialog box.

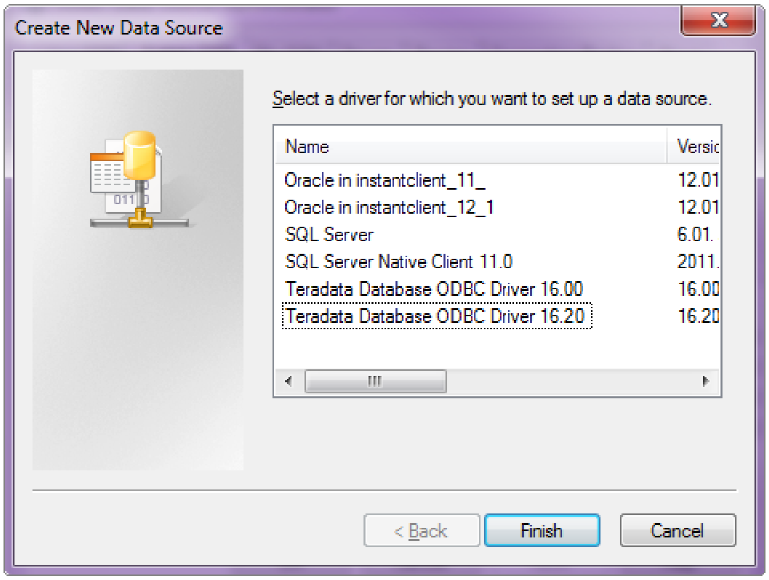


User Data Sources:

Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
MS Access Database	Microsoft Access Driver (*.mdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

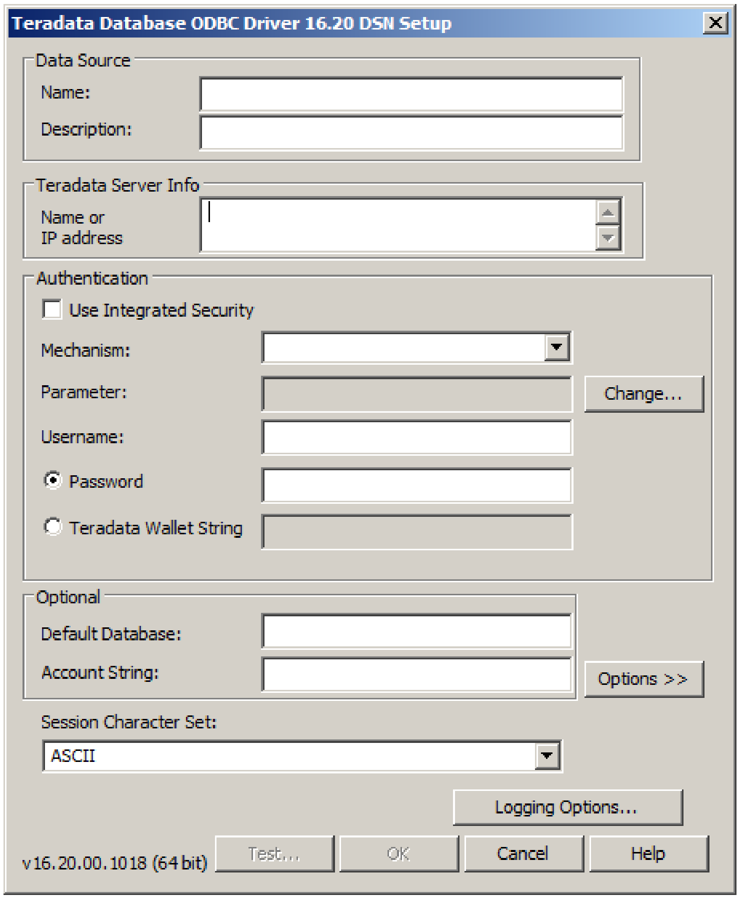
3. The **User DSN** and **System DSN** tabs list the names of existing data sources for the user and for all users of the system, respectively. To edit an existing data source, select it and click **Configure**. To create a new data source for Teradata Database, select the Teradata Database ODBC Driver with the desired version, click **Add**.



Select a driver for which you want to set up a data source.

Name	Versic
Oracle in instantclient_11_	12.01
Oracle in instantclient_12_1	12.01
SQL Server	6.01.
SQL Server Native Client 11.0	2011.
Teradata Database ODBC Driver 16.00	16.00
Teradata Database ODBC Driver 16.20	16.20

4. In the Data Source **Name** pane of the **Create New Data Source** dialog box, select **Teradata Database ODBC Driver 16.20**. Click **Finish** and the **Teradata16.20 ODBC Driver DSN Setup** dialog box appears.



☐ Use Integrated Security

● Password

○ Teradata Wallet String

Session Character Set: ASCII

v16.20.00.1018 (64 bit)

5. In the dialog box, fill in or modify the required information to identify the data source. The **OK** button for this dialog box does not become available unless the **Name** box in the **Data Source** group box and the **Teradata Server Info Name(s) or IP address(es)** group box are filled in.

ODBC Driver Setup Parameters

Field, Check Box, or Button	Description
Data Source group box	
Name	Enter the Data Source Name (DSN) that the application is to refer to when executing SQLConnect or SQLDriverConnect. The entered DSN is the

Field, Check Box, or Button	**Description**
	name that will appear in the **Data Sources** dialog box during a manual connection. See the “data-source-name=<driver>” option in ODBC Data Sources Section.
Description	[Optional] Enter descriptive text about the data source in this box. This is a comment field that is not retrievable by an SQL or ODBC command. See the “Description=<data-source-desc>” option in Data Source Specification Section.
Test Connection	
Username and password	Enter Username and password to test connection.
Buttons	
OK	Save changes to this dialog box.
Cancel	Cancel changes to this dialog box.
Help	Obtain help about this dialog box.
Teradata Server Info	
Name or IP address	Perform one of the following: • Enter the Teradata Database name (alias or FQDN) without COP suffix. • Enter the name or IP address of the LAN-connected node in your system, one per line. After providing the name, ODBC Driver for Teradata dynamically detects all associated COP entries. You must define COP names or the name without COP suffix in either a Domain Name Services (DNS) or the local hosts file. For more information, see Cop Discovery.
Authentication group box	

Field, Check Box, or Button	Description
Use Integrated Security	Default = Cleared Clear this check box to enable the user to connect to the database using Conventional Sign On (CSO). Select this check box to enable the user to connect to the database using Single Sign-On (SSO). In CSO from a network client, the user must provide a Teradata Database username and password. These are sent to Teradata Database for validation. If the username exists and the password correctly matches, access is allowed. Otherwise, the connection request is rejected. In SSO, the username and password are not submitted. The Teradata Database username is derived from the user identity on the client platform.
Mechanism	Default = determined by a configuration option set in an XML file by TeraGSS program, **`tdgssconfigure`**. Specify the desired security checking mechanism. Valid values are: • Empty – the same as omitting the keyword • TD2 – selects Teradata 2 as the authentication mechanism. Username and password are required. • TDNEGO – selects one of the Authentication Mechanisms automatically based on the policy, without user involvement. • LDAP – selects Lightweight Directory Access Protocol (LDAP) as the Authentication Mechanism. The application provides the username and password. • KRB5 – selects Kerberos (KRB5) on Windows clients working with Windows servers. The application provides the username and password. • JWT – selects JSON Web Token (JWT) as the Authentication Mechanism. An authentication mechanism based on the JWT needs to be provided in the form of "**`token=`*`<JWT token>`***" where ***`<JWT token>`*** is the actual JWT token. See complete descriptions at Authentication Mechanisms.
Parameter	Users cannot enter parameters in this field. Instead, click the **Change** button located to the right of the field, and a new dialog box opens, allowing the parameter to be entered. Once a parameter is entered, the text will be masked with * characters.

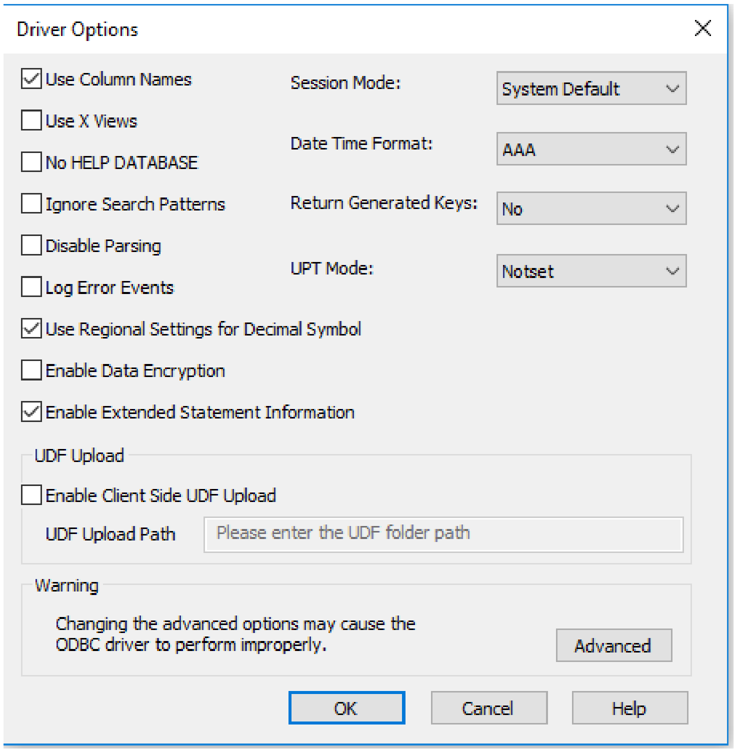
Field, Check Box, or Button	Description
	Indicate a string of characters to be regarded as a parameter of the authentication mechanism. The string is opaque to ODBC Driver for Teradata and is passed to the Teradata authentication software called to set the mechanism. Enclose characters [] {} () , ; ? * = ! @ in braces. You can use a Teradata Wallet reference string instead of a plain text parameter by specifying the **`$tdwallet()`** token. For example: **`$tdwallet(WalletRefString)`** For more information, see Teradata Wallet.
Username	Default = Cleared Specify a username for the default Teradata Database. Shows the default username that was specified during the data source(s) configuration of the driver. The default value can be overridden here. If required, the user is prompted for additional information. The username and password are Teradata-specific, not to be confused with Windows user ids and passwords. The username is interpreted in the context of the authentication mechanism. If, for example, the authentication mechanism is NTLM, then the username is assumed to be a Windows username. If the authentication mechanism allows fully qualified usernames, then the username can contain a domain or realm; for example, {***`judy@linedata`***}. Values containing a character such as @ must be enclosed in braces. SSO is indicated by the absence of a Username and Password. See the “Username=<name>” option in Data Source Specification Section.
Password	[Optional] The password required for the default Username. Not applicable on Apple OS X.

Field, Check Box, or Button	Description
	Note: Password stored locally (Windows registry) in encrypted form, but storing sensitive information in DSN is not 100% secure. Using this feature is not recommended, because it is deprecated. For more information, see Deprecated SQL Transformations. For information on disabling the ability to save the password in a DSN, see Security Considerations. Instead of storing the actual password in DSN, it is recommended to use a Teradata Wallet reference string as a password. Password and Teradata Wallet String fields are mutually exclusive, so a DSN can have either the Password or a Teradata Wallet String. For more information, see Teradata Wallet.
Teradata Wallet String	Enter and save a Teradata Wallet reference string as password for the user. Do not need to enclose the Teradata Wallet reference string in `$tdwallet()` token. It is recommended to use Teradata Wallet Reference string instead of saving the actual password in DSN. Password and Teradata Wallet String fields are mutually exclusive, so a DSN can have either the Password or a Teradata Wallet String. For more information, see Teradata Wallet.
Optional group box	
Default Database	Default = Cleared Specify a default database. See the “DefaultDatabase=<database-name>” option in Data Source Specification Section.
Account String	Default = Cleared Enter a user in Teradata Database while configuring the data source rather than having to provide account information during configuration of ODBC Driver for Teradata. This information can be used to help isolate users, to find out what application the users are running, or to restrict users from logging on.

Field, Check Box, or Button	**Description**
	See the "AccountStr=<account>" option in Teradata DSN Options.
Session Character Set	Default = ASCII Specify the character set for the session. It is strongly recommended to use the default ASCII session only for 7-bit ASCII characters. UTF8 is the recommended default session character set for all languages including US English. To use a different character set than is chosen by default, specify or select it here. The options available from the drop-down list are: • ASCII • UTF8 • UTF16 • LATIN1252_0A • LATIN9_0A • LATIN1_0A • Shift-JIS (Windows, DOS compatible, KANJISJIS_0S) • EUC (Unix compatible, KANJIEC_0U) • IBM Mainframe (KANJIEBCDIC5035_0I) • KANJI932_1S0 • BIG5 (TCHBIG5_1R0) • GB (SCHGB2312_1T0) • SCHINESE936_6R0 • TCHINESE950_8R0 • NetworkKorean (HANGULKSC5601_2R4) • HANGUL949_7R0 • ARABIC1256_6A0 • CYRILLIC1251_2A0 • HEBREW1255_5A0 • LATIN1250_1A0 • LATIN1254_7A0 • LATIN1258_8A0 • THAI874_4A0 Note: See the note in Session Character Sets and Translation DLLs for more information on restrictions. Note: For user-defined session character sets that are not shown on the above list, type the name of the user-defined session character set.

Teradata ODBC Driver Options

Click the **Options** button in the **ODBC Driver Setup for Teradata Database** dialog box for the **Driver Options** dialog box to appear. Use this dialog box to configure additional options and formats. Teradata ODBC Driver Options are listed in the table below.



Driver Options

☑ Use Column Names
☐ Use X Views
☐ No HELP DATABASE
☐ Ignore Search Patterns
☐ Disable Parsing
☐ Log Error Events
☑ Use Regional Settings for Decimal Symbol
☐ Enable Data Encryption
☑ Enable Extended Statement Information

Session Mode: System Default
Date Time Format: AAA
Return Generated Keys: No
UPT Mode: Notset

UDF Upload
☐ Enable Client Side UDF Upload
UDF Upload Path: Please enter the UDF folder path

Warning
Changing the advanced options may cause the ODBC driver to perform improperly.

Field, Check Box, or Button	Description
Use Column Names	Default = Selected Determines whether ODBC Driver for Teradata returns column names or column titles. See the “DontUseTitles=[Yes|No]” option in Teradata DSN Options.
Use X Views	Default = Cleared Determines whether the X View is used. See the “UseXViews=[Yes|No]” option in Teradata DSN Options.
No HELP DATABASE	Default = Cleared Determines whether the Help Database is used.

Field, Check Box, or Button	Description
	See the "DontUseHelpDatabase=[Yes|No]" option in Teradata DSN Options.
Ignore Search Patterns	Default = Cleared Determines whether search pattern characters _ and % are used in search patterns or are passed as regular characters. See the "IgnoreODBCSearchPattern=[Yes|No]" option in Teradata DSN Options.
Disable Parsing	Default = Cleared Disables or enables parsing of SQL statements by ODBC Driver for Teradata. When parsing is enabled, the driver parses SQL statements and transforms ODBC escape sequences into SQL. • When selected, ODBC Driver for Teradata does *not* parse SQL statements. • When cleared, SQL statements are parsed. See the "NoScan=[Yes|No]" option in Teradata DSN Options.
Log Error Events	Default = Cleared When selected, ODBC Driver for Teradata performs error logging that appears in the Event Viewer. When cleared, error logging is not performed.
Use Regional Settings for Decimal Symbol	Default = Selected (Windows) Allows the application to decide which symbol is used as a decimal separator while retrieving decimal data. When selected, ODBC Driver for Teradata uses regional settings to determine the decimal symbol. If cleared, ODBC Driver for Teradata uses a "." character as the decimal symbol and ignores the regional settings.
Enable Data Encryption	Default = Cleared When Enable Data Encryption is checked – the option directs the ODBC Driver to use Data Encryption, causing ODBC Driver for Teradata and Teradata Database to communicate with each other in encrypted manner.

Field, Check Box, or Button	Description
	When Enable Data Encryption is cleared–Data Encryption is disabled.
Enable Extended Statement Information	Default = Selected Determines whether extended statement information is to be used by ODBC Driver for Teradata, provided that it is available from Teradata Database. Database versions from V2R6.2 and up support extended statement information, including metadata for parameters used in SQL requests and columns in result sets. When EnableExtendedStmtInfo is Selected – ODBC Driver for Teradata requests and uses extended statement information from the database if supported. If extended statement information is available, then the ODBC API function SQLDescribeParam is supported and SQLGetFunctions returns SQL_TRUE (supported) for SQL_API_SQLDESCRIBEPARAM. When EnableExtendedStmtInfo is cleared – ODBC Driver for Teradata does not request or use extended statement information from the database, even if supported. If extended statement information is unavailable, SQLDescribeParam is not supported and SQLGetFunctions returns SQL_FALSE (not supported) for SQL_API_SQLDESCRIBEPARAM.
Session Mode	Specifies the mode (Teradata or ANSI) for sessions on Teradata Database. The selected mode applies for the duration of the session. The default value is determined by the database based on the option used in the Teradata Database CREATE or MODIFY USER statement.
DateTime Format	Assigns the ANSI formats for DATE, TIME, and TIMESTAMP. The default setting is AAA. IAA is optional. Because the Integer data type has been deprecated for the TIME format, it is not recommended. For information, see Integer Time. See the “DateTimeFormat=[A|I]AA” option in Teradata DSN Options.
Return Generated Keys	Default = No Determines the result from requests that insert into identity columns (INSERT, INSERT … SELECT, UPSERT, MERGE-INTO). These requests can optionally return a result set containing identity column values (also known as auto-generated keys) for the inserted rows.

Field, Check Box, or Button	Description
	Auto-generated key retrieval is not supported in Teradata Database versions prior to V2R6.2. The setting of Return Generated Keys has no effect when using a pre-V2R6.2 database server. Valid values are No, Identity Column, and Whole Row: • No = Auto-generated key retrieval is disabled (default) • Identity Column = Retrieve identity column only • Whole Row = Retrieve entire row(s) When Return Generated Keys is set to Identity Column or Whole Row, a request that inserts into tables containing identity columns returns two results: a row count with the number of inserted rows and a result set containing either the auto-generated keys as a single column or the complete rows inserted. The insert request becomes similar to a macro that first inserts and then selects the identity column or all columns of the rows just inserted. When Return Generated Keys is set to No, the behavior of requests that insert into identity columns is not changed.
UPT Mode	Default = NOTSET Enables Unicode Pass Through Mode for the ODBC Application. The default value is NOTSET, which means that the UPT Mode set by the database is used. The ODBC Driver for Teradata does not send anything to the database when this option is set. UPTON: ODBC Driver for Teradata sends “SET SESSION CHARACTER SET UNICODE PASS THROUGH ON” to the database while connecting, thereby enabling UPT MODE for that session. UPTOFF: ODBC Driver for Teradata sends “SET SESSION CHARACTER SET UNICODE PASS THROUGH OFF” to the database while connecting, thereby disabling UPT MODE for that session. See the "UPTMode=[***NotSet***|***UPTOn***|***UPTOff***]" option in Teradata DSN Options.
UDF Upload	

Field, Check Box, or Button	**Description**
Enable Client Side UDF Source	Default=Cleared. If this is unchecked, the driver will not support UDF source file uploads. See the "EnableUDFUpload=[***Yes***|***No***] option in Teradata DSN Options.
UDF Upload Path	Fully qualified path where source files will be found. If defined, the driver looks at this location for files the database requests, unless the database gives a fully qualified path as part of the file name. No relative paths using ".." are allowed in this value. The default value displayed in the field (`Please enter the UDF folder path`) MUST be changed to the value you wish to use (either a valid path or empty). Delete the default value to leave the field empty. If **Enable Extended Statement Information** is checked: • The path you specify in **UDF Root Directory** will be prepended to all file names specified with an EXTERNAL NAME clause of a CREATE FUNCTION or REPLACE function. • To use fully qualified file names in EXTERNAL NAME clauses, leave this field empty. See "UDFUploadPath=<***path***>" option in Teradata DSN Options.
Warning group box	
Advanced	Click to bring up the dialog box. The dialog box contains further setting options available; however, it is *strongly recommended* NOT to change these settings.

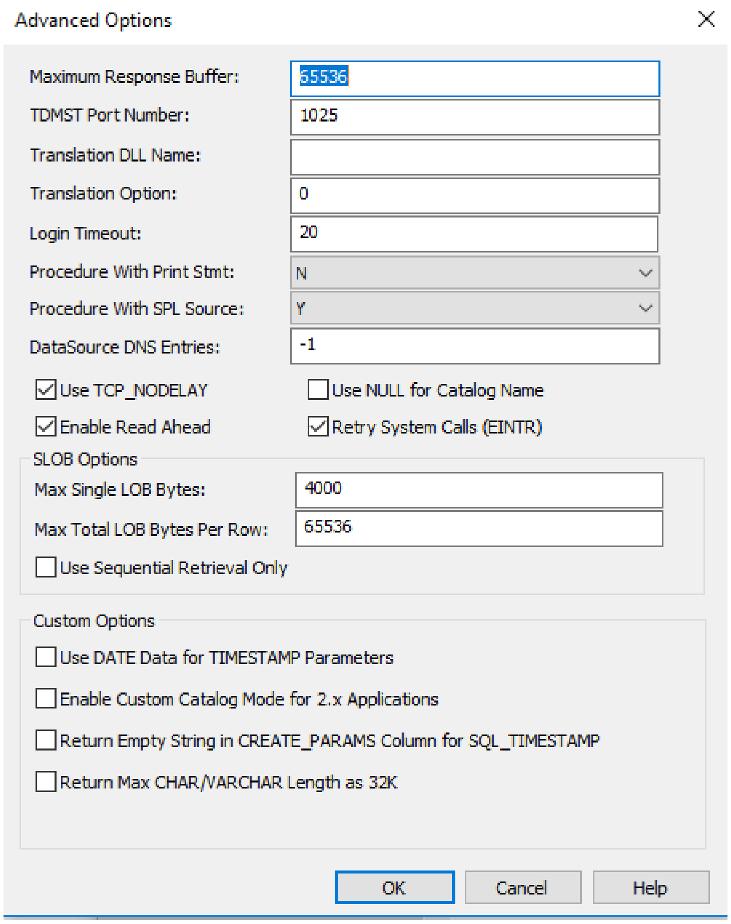
Teradata ODBC Driver Advanced Options

Click the **Advanced** button in the **Warning** area of the **Teradata ODBC Driver Options** dialog box to see further options available. The **Advanced Options** dialog box appears. The table below lists the advanced options.

Advanced Options Caution

Note:

It is *strongly recommended* NOT to change the settings in the **Advanced Options** dialog box.



Maximum Response Buffer: 65536
TDMST Port Number: 1025
Translation DLL Name:
Translation Option: 0
Login Timeout: 20
Procedure With Print Stmt: N
Procedure With SPL Source: Y
DataSource DNS Entries: -1
☑ Use TCP_NODELAY
☐ Use NULL for Catalog Name
☑ Enable Read Ahead
☑ Retry System Calls (EINTR)

SLOB Options
Max Single LOB Bytes: 4000
Max Total LOB Bytes Per Row: 65536
☐ Use Sequential Retrieval Only

Custom Options
☐ Use DATE Data for TIMESTAMP Parameters
☐ Enable Custom Catalog Mode for 2.x Applications
☐ Return Empty String in CREATE_PARAMS Column for SQL_TIMESTAMP
☐ Return Max CHAR/VARCHAR Length as 32K

Field, Check Box, or Button	Description
Maximum Response Buffer Size	Default = 65536 (64K) Enter the value used to try to limit the Teradata response buffer size for SQL requests. This value can be adjusted dynamically if Teradata cannot send a result within the limited packet size defined. See the "MaxRespSize=<integer 16775168>" option in Teradata DSN Options.

Field, Check Box, or Button	Description
TDMST Port Number	Default = 1025 Lists the port number to use to access Teradata Database. Do not change this value unless instructed to do so by Technical Support. See the “TDMSTPortNumber=<integer>” option in Teradata DSN Options.
Translation DLL Name	Specifies the translation DLL path. Translation DLL is used to convert between session character set and application character set. See Session Character Sets and Translation DLLs.
Translation Option	Specifies the translation DLL option. The option is used by translation DLL. See Session Character Sets and Translation DLLs.
Login Timeout	Default = 20 Defines a value corresponding to the number of seconds to wait when establishing a virtual circuit with Teradata for login. Enter an integer value. See Teradata DSN Options.
ProcedureWithPrintStmt	Default = N Activates the print option when creating stored procedures. See the “PrintOption=[N | P]” option in Teradata DSN Options.
ProcedureWithSPLSource	Default = Y Specifies the SPL option when creating stored procedures. See the “SplOption=[Y | N]” option in Teradata DSN Options.
Data Source DNS Entries	The **Data Source DNS Entries** DSN option notifies the ODBC Driver for Teradata how many entries are defined in DNS for the database name. The initial value of this option controls how the ODBC Driver for Teradata resolves database names to IP addresses. If this value is not set, the default value is undefined (empty). If multiple database

Field, Check Box, or Button	Description
	names are provided in ODBC DSN, the **Data Source DNS Entries** option is applicable to all names. Note: If a database is identified by IP address instead of a name in the ODBC DSN or connection-string, the **Data Source DNS Entries** option is ignored. The database is identified in the **Name(s) or IP address(es)** field described in ODBC Driver Setup Parameters. **Data Source DNS Entries**=undefined (default setting) is recommended for best results. This setting enables the ODBC Driver for Teradata to lookup DNS dynamically and find all available COPs for a given database name. Using this approach, ODBC Driver for Teradata will automatically detect new nodes added to the Teradata database (and DNS) in the future, without ODBC modification. For more information, see Resolving a Data Source Name. **Data Source DNS Entries**= 0 indicates that DNS does not contain ***cop*** entries for the database name. The database name will only be resolved by itself. No attempt will be made to resolve using a cop suffix. This behavior can be desirable in an environment utilizing DNS to load balance. When DNS is used for load balancing, administrators can configure DNS to provide a different IP address or multiple IP addresses in different order each time the database name is resolved using DNS. **Data Source DNS Entries**= value. Entering a non-zero value indicates that DNS contains ***cop*** entries for the database name and the last cop entry is value. The first connection attempt will chose a random number between 1 and value. Each subsequent connection will then increment to the next number (round-robin). This approach will not encounter costly DNS resolution failures (how costly depends on how the DNS is configured). However, if additional entries are added to DNS at a later time, they will not be discovered by the ODBC Driver for Teradata unless the supplied value is increased.
Use TCP_NODELAY	Default = Selected Valid for the Teradata DSN in ODBC Driver for Teradata.

Field, Check Box, or Button	Description
	Transmission Control Protocol (TCP) provides an option called TCP_NODELAY to control the transmission of data. See the “TCPNoDelay=[Yes | No]” option in Teradata DSN Options.
Use NULL for Catalog Name	Default = Cleared When this option is selected – NULL values are assumed for the Catalog Name parameters in any of the Catalog API functions, even if the application passes a value. When this option is cleared and a value is passed for the Catalog Name parameter instead of NULL – ODBC Driver for Teradata returns an error because catalogs are not supported by Teradata Database.
Enable Read Ahead	Default = Selected When this option is selected - the ODBC Driver reads ahead by requesting the next response message from the database when the current response message being processed is not the last. The database can have one request active for each session at any point in time. An active request is either an SQL request which is executing or a request for the next part of the result from an earlier SQL request. When this option is cleared - the ODBC Driver only requests the next response message from the database when the current response message has been processed by the driver.
Retry system calls (EINTR)	Default = Selected When this option is cleared – the ODBC Driver returns an SQL_ERROR to the ODBC Application. The ODBC Application is responsible for recovery from the interrupted socket system calls.When this option is selected – the ODBC Driver is responsible for retrying the socket system calls when they have been interrupted by some event such as a SIGALRM.
SLOB Options group box	
Max Single LOB Bytes	Unsigned int 32

Field, Check Box, or Button	Description
	Default value is 4000: The maximum size in byte for LOB data to be returned as SLOBs for each row. 0 means the feature is disabled.
Max Total LOB Bytes Per Row	Unsigned int 32 Default value is 65536: The maximum size in byte for LOB data to be returned as SLOBs for each row. 0 means the feature is disabled.
Use Sequential Retrieval Only	Boolean Default values is false: This is a performance-related setting. It indicates that only sequential retrieval or random access will be used. If sequential retrieval only is enabled, the driver will not cache the data for SLOBs. It retrieves data faster by skipping the caching step. It also means if sequential only is set and the client performs random access, performance will be downgraded because the data is retrieved using a deferred LOB. If random access is required, each SLOB data will be cached for later use/reuse. This slows the retrieval process, but permits the client to return to the data at a later time.
Custom Options group box	
Use DATE data for TIMESTAMP parameters	Default = Cleared When this option is selected– ODBC Driver for Teradata is directed to send DATE data for parameters bound as SQL_C_TIMESTAMP and SQL_TIMESTAMP. This option should not be enabled for applications that are not using Microsoft Access Jet databases. This option should only be selected for this circumstance, as this results in truncation of SQL_C_TIMESTAMP data to contain only the DATE portion.
When this option is selected – the ODBC DriverEnable	Default = Cleared

Field, Check Box, or Button	Description
Custom Catalog Mode for 2.x Applications	Provides backwards compatibility for ODBC 2.x applications that have taken advantage of a defect in the ODBC Driver where the functionality of the Catalog APIs are noncompliant with the *ODBC Programmer's Reference* specification. The behavior when a NULL value is passed to the SQLTables API for the SchemaName argument results in a search for tables belonging to the userid, DBC, and default database schema names, rather than a % search pattern as noted in the *ODBC Programmer's Reference* specification.
Return Empty string in CREATE_PARAMS column for SQL_TIMESTAMP	Default = Cleared Returns an empty string for the CREATE_PARAMS column of SQLGetTypeInfo for SQL_TIMESTAMP data type, and disallows MC-ACCESS from using any TIMESTAMP precision value in Create Table text.
Return max. CHAR/ VARCHAR length as 32K	Default = Cleared Returns a value of 32000 (in general, could be 64000 also) for COLUMN_SIZE column of SQLGetTypeInfo for SQL_CHAR and SQL_VARCHAR data types. This allows MS-ACCESS to handle column size value returned by ODBC Driver for Teradata without any numeric overflow.

Upon completion, return to each previous screen in the sequence by clicking the **OK** button.

After entering the required information into the **Teradata Database ODBC Driver 16.20 DSN Setup** dialog box, click **OK**.

The data source that you just configured displays in the user Data Sources list of the **ODBC Data Source Administrator** dialog box.

To continue adding data sources, repeat the process.

Reconfiguring a Data Source

The following topics describe how to modify an existing data source configuration for ODBC Driver for Teradata.

Modifying a Data Source

1. Select the data source name from the **User DSN** or the **System DSN** tab of the **ODBC Data Source Administrator** dialog box and click the **Configure** button.
2. Make editing changes.
3. Click **OK** when the configuration is complete.

Deleting a Data Source

1. Select the data source name from the **User DSN** tab or the **System DSN** tab of the **ODBC Data Source Administrator** dialog box and click the **Remove** button.
2. Confirm the deletion by clicking the **Yes** button when the **ODBC Administrator** dialog box appears.
3. Click **OK** from the **ODBC Data Source Administrator** dialog box when the configuration is complete.

Resolving a Data Source Name

ODBC Driver for Teradata can connect to Teradata Database using an IP-address (IPv4 or IPv6), an alias name, or a Fully Qualified Domain Name (FQDN).

Cop Discovery

The ODBC Driver for Teradata determines if a DNS contains cop entries for the database name. The first attempt to resolve the database name has a `cop1` suffix appended. Cop discovery is disabled if a `cop1` entry is not defined in DNS. All resolutions will be made for this name without adding the cop suffix, and no further attempts to resolve cop1 will be attempted.

If an entry for cop1 is found in DNS, then it is assumed that DNS contains further cop entries for the database name. However, the number of entries is not known. Subsequent resolutions will increment to the next number and attempt to find an entry in DNS. The first entry that is not found in DNS marks the last entry for the given name. At that point the list of DNS entries is complete. The first connection attempt chooses a random number between the first and last cops found. Each subsequent connection will increment to the next number (round-robin).

If multiple database names are provided in ODBC DSN, and the first name has no cops defined, all names will be resolved without adding a cop suffix. No attempt will be made to find the number of cops defined.

In order for the ODBC Driver for Teradata to discover the last entry in the sequence, a DNS resolution must fail. DNS resolutions which result in a failure might be time consuming, depending on how the DNS is configured. If a name lookup failure in the DNS is time consuming, type the number of DNS-defined cops into the **Datasource DNS Entries** field. If additional entries are added to DNS at a later time, they will be discovered by the ODBC Driver for Teradata (if Data Source DNS Entries is not set or undefined).

For details about how to modify cop handling, see **Data Source DNS Entries** in Teradata ODBC Driver Advanced Options.

Session Character Sets and Translation DLLs

The following table indicates SQLDriverToDataSource conversions. The subsequent table describes SQLDataSourceToDriver conversions. UTF8/UTF16 session character sets do not require translation DLLs.

Function of SQLDriverToDataSource
Convert Windows application code page to UTF16

The following table describes the functions of the SQLDataSourceToDriver.

Function of SQLDataSourceToDriver
Convert UTF16 to Windows application code page

User-defined session character sets can be used either with or without a translation DLL. If no translation DLL is specified, the driver's conversion is based on the current application code page. With Teradata-provided session character sets, no translation DLLs are needed as long as the session character set matches to the current application code page. For example, a Chinese session character set matches the Chinese Windows PC.

It is recommended translation DLLs be copied to the same folder and location where the driver was installed.

Note:

> KANJIEBCDIC5038_0I is supported on Windows systems when set up as a Japanese machine (CP932). Other configurations are *not* supported. This character set is used for communications to and from the database. The application must use the Windows CP932 session character set when communicating with the ODBC driver. This behavior is identical to KANJIEUC_0U.

Working with DSNs

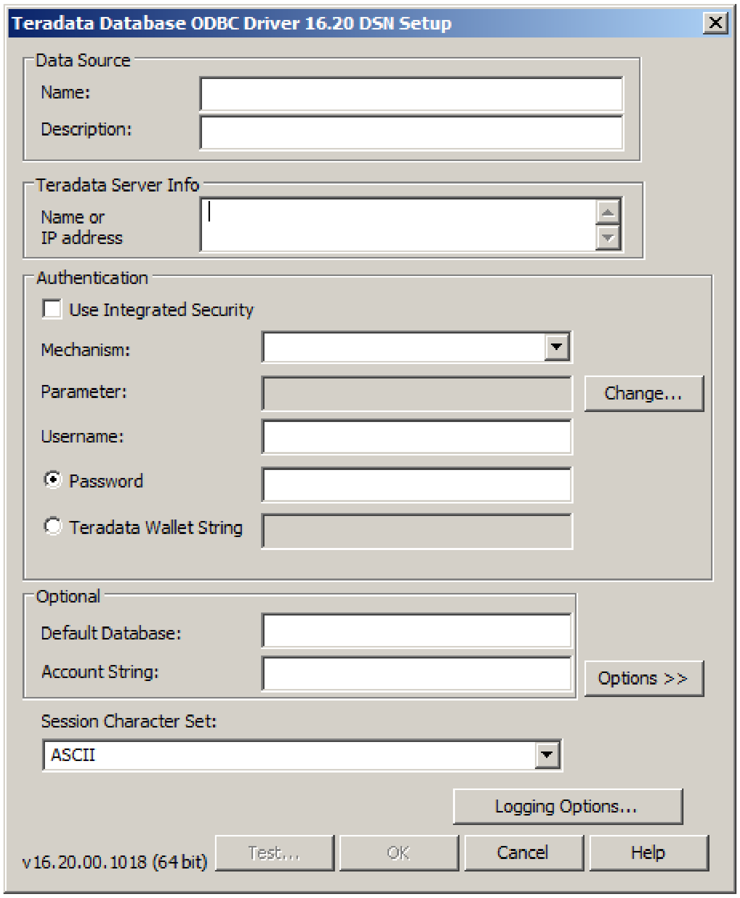
The following subsections describe how to trace, migrate, and restore DSNs.

DSN Tracing

ODBC Driver for Teradata can provide detailed trace information for DSNs associated with Teradata Database. This is referred to as *DSN* tracing, which differs from the higher level *Driver Manager* tracing, which is enabled and disabled using the ODBC Data Source Administrator.

The DSN trace file is a text file with internal information from ODBC Driver for Teradata about API calls, message transfers, and other driver actions. It is primarily intended for use by Teradata Technical Support.

Driver level tracing can be performed easily using ODBC Data Source Administrator by choosing your DSN entry and selecting the **Logging Options** button.



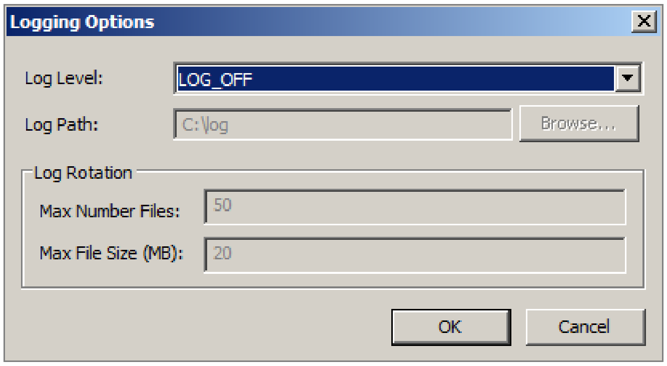
● Password

○ Teradata Wallet String

Session Character Set: ASCII

v16.20.00.1018 (64 bit)

The **Logging Options** dialog box appears. Select the desired level of logging and the destination for the logs. Refer to the section below for more information on logging levels and steps for how to enable driver logging on Windows.



Log Level: LOG_OFF

Log Path: C:\log

Log Rotation

Max Number Files: 50

Max File Size (MB): 20

Note:

Using DSN tracing will significantly degrade performance.

DSN Migration

Note:

DSN migration is no longer done automatically during installation.

Enabling Driver Logging on Windows

1. Open ODBC Data Source Administrator where you created the DSN to access logging options.
2. Select the DSN and click **Configure > Logging Options.**
3. From the **Log Level** drop down list, select the logging level corresponding to the amount of information you want included in the files.

Logging Level	Description
OFF (Default)	Disables all logging.
FATAL	Logs severe error events that lead the driver to abort.
ERROR	Logs error events that might allow the driver to continue running.
WARNING	Logs events that might result in an error if no action is taken.
INFO	Logs general information describing the progress of the driver.
DEBUG	Logs detailed information that is useful for debugging the driver.
TRACE	Logs all driver activity.

4. In the **Log Path** field, specify the full path to the target folder where you want to save log files.
5. In the **Max Number Files** field, type the maximum number of log files to keep.
   After the maximum number of log files is reached, each time an additional file is created the driver deletes the oldest log file.
6. In the **Max File Size** field, type the maximum size of each log file in megabytes (MB).
   After maximum file size is reached, the driver creates a new file and continues logging.
7. Restart your ODBC application to verify the settings.
   The new Teradata ODBC Driver produces two log files at the location you specify using the LogPath key, where *`[DriverName]`* is the name of the driver:
   - A *`[DriverName]`*`_driver.log` file that logs driver activity not specific to a connection.
   - A *`[DriverName]`*`_connection_`*`[Number]`*`.log` for each connection made to the database, where *`[Number]`* is a number that identifies each log file. This file logs driver activity specific to the connection.

Disabling Driver Logging on Windows

1. Open the ODBC Data Source Administrator where you created the DSN.
2. Select the DSN and click **Configure > Logging Options.**
3. From the **Log Level** drop-down list, select **LOG_OFF**.
4. Click **OK**.
5. Restart your ODBC application to verify the new settings.

Configuration for Apple OS X

Overview

This section provides instructions for configuring a data source and ODBC Driver for Teradata on systems running Apple OS X.

For information about the UNIX system versions of ODBC Driver for Teradata, see Configuration for UNIX/Linux Systems.

For information about the Windows version of ODBC Driver for Teradata, see Configuration for Windows.

ODBC Driver Is Universal Binary

Teradata ODBC Driver library is a Universal binary containing 32-bit and 64-bit versions. Depending on application bitness, either 32-bit or 64-bit version of driver will be loaded appropriately.

ODBC Driver Directories

Pathname	Description
`/Library/Application Support/teradata/client/<TTU version>/lib`	ODBC Driver libraries
`/Library/Application Support/teradata/client/ODBC/include`	symbolic link to `'/Library/Application Support/teradata/client/<TTU version>/odbc/include'`
`/Library/Application Support/teradata/client/ODBC/lib`	symbolic link to `'/Library/Application Support/teradata/client/<TTU version>/odbc/lib'`
`/Library/Application Support/teradata/client/<TTU version>/bin`	ODBC Connection test tool
`/Library/Application Support/teradata/client/<TTU version>/include`	Teradata ODBC Driver custom definitions
`/Library/Application Support/teradata/client/<TTU version>/odbc/msg`	Teradata ODBC Driver error message catalog file

Pathname	Description
`/Library/Application Support/teradata/client/<TTU version>/odbc/samples`	Sample ODBC application source code
`/Library/Application Support/teradata/client/ODBC/odbc.ini`	Sample `odbc.ini` file

ODBC Driver Manager

Teradata ODBC driver is supported with iODBC driver manager (version 3.52.8) that is installed by default on Apple OS X. The Teradata Tools and Utilities install suite will not install ODBC Driver Manager along with ODBC Driver for Teradata.

odbc.ini File

ODBC DSNs are saved in `odbc.ini` file. The default locations to save the `odbc.ini` file are:

User DSNs: `"~/Library/ODBC/"`

System DSNs: `"/Library/ODBC/"`

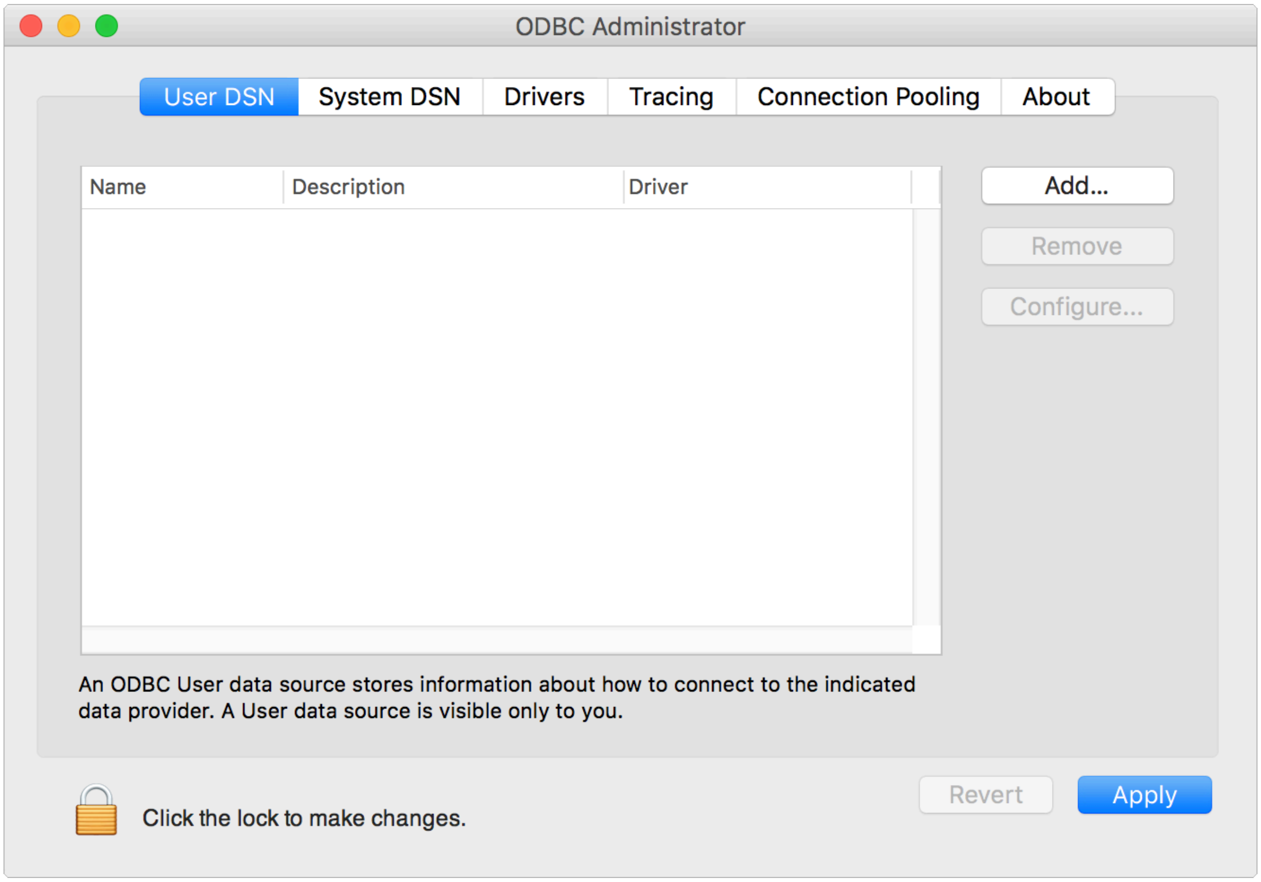
Optionally, the ODBCINI environment variable can point to an `odbc.ini` file present in non-default locations.

ODBC DSNs can be configured using the GUI Administrator tool or manually.

Configuring a DSN Using ODBC Administrator Tool

The ODBC Administrator Tool from Apple is available for download at http://support.apple.com/kb/DL895. This is the ODBC administrator tool recommended by Teradata. Once you have installed it, follow the following procedure to configure a DSN.

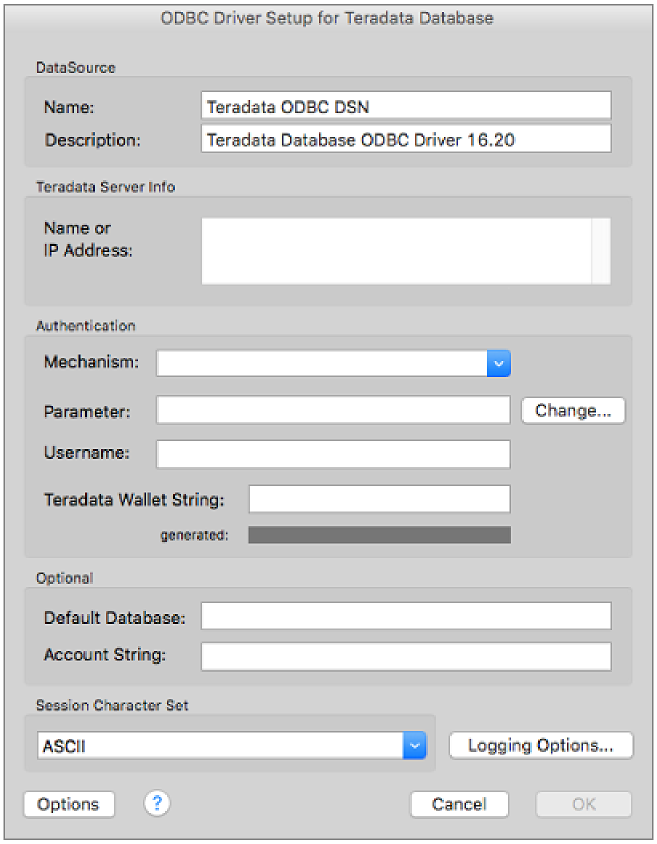
1. In the Finder under **Go** select **Utilities > ODBC Administrator** .



An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is visible only to you.

Click the lock to make changes.

2. The **User DSN** tab shows a list of DSNs saved in `~/Library/ODBC/odbc.ini` file. To modify or remove an existing DSN, select it, and press the respective button. To add a new user DSN, click **Add**.
3. In the **Choose a Driver** dialog, select **Teradata Database ODBC Driver 16.20** and click **OK**.



ODBC Driver Setup for Teradata Database

DataSource

Name: Teradata ODBC DSN

Description: Teradata Database ODBC Driver 16.20

Session Character Set: ASCII

4. In the **ODBC Driver Setup for Teradata Database** dialog, fill in the required information as described in the following table and click **OK** to save the DSN.

Note:

The **OK** button does not become available until the **Name or IP Address** field is filled in.

Field, Check Box, or Button	Description
Data Source group box	
Name	Enter the Data Source Name (DSN) that the application is to refer to when executing SQLConnect or SQLDriverConnect. The entered DSN is the name that will appear in the **Data Sources** dialog box during a manual connection. See the "data-source-name=<driver>" option in ODBC Data Sources Section.
Description	[Optional] Enter descriptive text about the data source in this box.

Field, Check Box, or Button	Description
	This is a comment field that is not retrievable by an SQL or ODBC command. See the “Description=<data-source-desc>” option in Data Source Specification Section.
Teradata Server Info	
Name or IP address	Perform one of the following: • Enter the Teradata Database name (alias or FQDN) without COP suffix. • Enter the name or IP address of the LAN-connected node in your system, one per line. After providing the name, ODBC Driver for Teradata dynamically detects all associated COP entries. You must define COP names or the name without COP suffix in either a Domain Name Services (DNS) or the local hosts file.
Authentication group box	
Mechanism	Default = determined by a configuration option set in an XML file by TeraGSS program, `tdgssconfigure`. Specify the desired security checking mechanism. Valid values are: • TD2 - selects Teradata 2 as the authentication mechanism. Username and password are required. • TDNEGO – selects one of the Authentication Mechanisms automatically based on the policy, without user involvement. • LDAP - selects Lightweight Directory Access Protocol (LDAP) as the Authentication Mechanism. The application provides the username and password. • KRB5 – selects Kerberos (KRB5) on Windows and Apple OS X clients working with Windows servers. The application provides the username and password. • JWT – selects JSON Web Token (JWT) as the Authentication Mechanism. An authentication mechanism based on the JWT needs to be provided in the form of "`token=`*`<JWT token>`*" where *`<JWT token>`* is the actual JWT token. See complete descriptions at Authentication Mechanisms.

Field, Check Box, or Button	**Description**
Parameter	Users cannot enter parameters in this field. Instead, click the **Change** button located to the right of the field, and a new dialog box opens, allowing the parameter to be entered. Once a parameter is entered, the text will be masked with * characters. Indicate a string of characters to be regarded as a parameter of the authentication mechanism. The string is opaque to ODBC Driver for Teradata and is passed to the Teradata authentication software called to set the mechanism. Enclose characters [] {} () , ; ? * = ! @ in braces. You can use a Teradata Wallet reference string instead of a plain text parameter by specifying the **`$tdwallet()`** token. For example: `$tdwallet(WalletRefString)` For more information, see Teradata Wallet.
Username	Default = Cleared Specify a username for the default Teradata Database. Shows the default username that was specified during the data source(s) configuration of the driver. The default value can be overridden here. If required, the user is prompted for additional information. The username and password are Teradata-specific, not to be confused with other user ids and passwords. The username is interpreted in the context of the authentication mechanism. If the authentication mechanism allows fully qualified usernames, then the username can contain a domain or realm; for example, {*judy@linedata*}. Values containing a character such as @ must be enclosed in braces. SSO is indicated by the absence of a Username and Password. See the “Username=<name>” option in Data Source Specification Section.

Field, Check Box, or Button	Description
Teradata Wallet String	Enter and save a Teradata Wallet reference string as password for the user. Do not need to enclose the Teradata Wallet reference string in `$tdwallet()` token. For more information, see Teradata Wallet.
Optional group box	
Default Database	Default = Cleared Specify a default database. See the “DefaultDatabase=<database-name>” option in Data Source Specification Section.
Account String	Default = Cleared Enter a user in Teradata Database while configuring the data source rather than having to provide account information during configuration of ODBC Driver for Teradata. This information can be used to help isolate users, to find out what application the users are running, or to restrict users from logging on. See the “AccountStr=<account>” option in Teradata DSN Options.
Session Character Set	Default = ASCII Specify the character set for the session. It is strongly recommended to use the default ASCII session only for 7-bit ASCII characters. UTF8 is the recommended default session character set for all languages including US English. To use a different character set than is chosen by default, specify or select it here. The options available from the drop-down list are as follows: • ASCII • UTF8 • UTF16 • LATIN1252_OA • LATIN9_OA • LATIN1_OA • Shift-JIS (Windows, DOS compatible, KANJISJIS_OS) • EUC (Unix compatible, KANJIEC_OU) • IBM Mainframe (KANJIEBCDIC5035_OI) • KANJI932_1SO

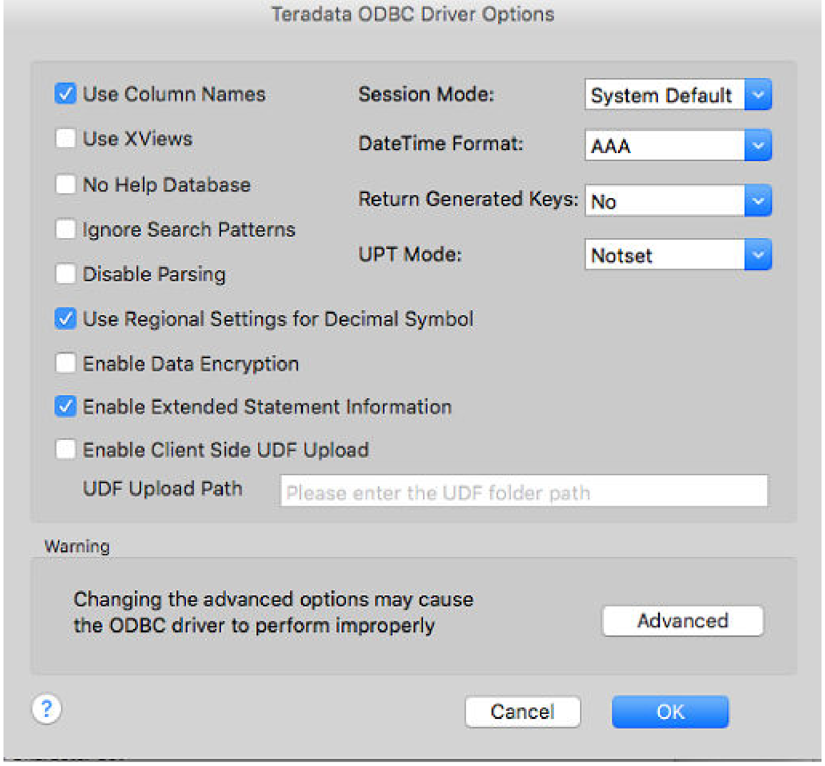
Field, Check Box, or Button	**Description**
	• BIG5 (TCHBIG5_1R0) • GB (SCHGB2312_1T0) • SCHINESE936_6R0 • TCHINESE950_8R0 • NetworkKorean (HANGULKSC5601_2R4) • HANGUL949_7R0 • ARABIC1256_6A0 • CYRILLIC1251_2A0 • HEBREW1255_5A0 • LATIN1250_1A0 • LATIN1254_7A0 • LATIN1258_8A0 • THAI874_4A0 Note: For user-defined session character sets that are not shown on the above list, type the name of the user-defined session character set.
Buttons	
OK	Click to enable the driver to use any changes that have been made in the dialog box. Note that this button does not become available unless the **Name** box in the **Data Source** group box and the **Teradata Server Info Name(s) or IP address(es)** group box are filled in.
Cancel	Click to cancel any changes made to the dialog box and abort the current driver and data source selection.
?	Click to obtain detailed help about this dialog box.
Options	Click to display the **Teradata ODBC Driver Options** dialog box and configure additional options and formats.

Related Information:

Teradata ODBC Driver Options
Teradata ODBC Driver Advanced Options

Teradata ODBC Driver Options

1. Use the **Teradata ODBC Driver Options** dialog to configure additional options. The options are listed in the following table.



Teradata ODBC Driver Options

☑ Use Column Names
☐ Use XViews
☐ No Help Database
☐ Ignore Search Patterns
☐ Disable Parsing

Session Mode: System Default
DateTime Format: AAA
Return Generated Keys: No
UPT Mode: Notset

☑ Use Regional Settings for Decimal Symbol
☐ Enable Data Encryption
☑ Enable Extended Statement Information
☐ Enable Client Side UDF Upload

UDF Upload Path: Please enter the UDF folder path

Warning

Changing the advanced options may cause the ODBC driver to perform improperly

Field, Check Box, or Button	Description
Use Column Names	Default = Selected Determines whether ODBC Driver for Teradata returns column names or column titles. See the “DontUseTitles=[Yes|No]” option in Teradata DSN Options.
Use X Views	Default = Cleared Determines whether the X View is used. See the “UseXViews=[Yes|No]” option in Teradata DSN Options.
No HELP DATABASE	Default = Cleared Determines whether the Help Database is used.

Field, Check Box, or Button	Description
	See the "DontUseHelpDatabase=[Yes|No]" option in Teradata DSN Options.
Ignore Search Patterns	Default = Cleared Determines whether search pattern characters _ and % are used in search patterns or are passed as regular characters. See the "IgnoreODBCSearchPattern=[Yes|No]" option in Teradata DSN Options.
Enable Reconnect	Default = Cleared Causes ODBC Driver for Teradata to determine if sessions on Teradata are to be reconnected after a system crash or reset is detected. See the "EnableReconnect=[Yes|No]" option in Teradata DSN Options.
Disable Parsing	Default = Cleared Disables or enables parsing of SQL statements by ODBC Driver for Teradata. When parsing is enabled, the driver parses SQL statements and transforms ODBC escape sequences into SQL. • When selected, ODBC Driver for Teradata does *not* parse SQL statements. • When cleared, SQL statements are parsed. See the "NoScan=[Yes|No]" option in Teradata DSN Options.
Use Regional Settings for Decimal Symbol	Default = Selected When selected, ODBC Driver for Teradata uses regional settings to determine the decimal symbol. If cleared, ODBC Driver for Teradata uses a "." character as the decimal symbol and ignores the regional settings.
Enable Data Encryption	Default = Cleared When Enable Data Encryption is checked – the option directs the ODBC Driver to use Data Encryption, causing ODBC Driver for Teradata and Teradata Database to communicate with each other in encrypted manner. When Enable Data Encryption is cleared–Data Encryption is disabled.

Field, Check Box, or Button	Description
Enable Extended Statement Information	Default = Selected Determines whether extended statement information is to be used by ODBC Driver for Teradata, provided that it is available from Teradata Database. Database versions from V2R6.2 and up support extended statement information, including metadata for parameters used in SQL requests and columns in result sets. When EnableExtendedStmtInfo is Selected – ODBC Driver for Teradata requests and uses extended statement information from the database if supported. If extended statement information is available, then the ODBC API function SQLDescribeParam is supported and SQLGetFunctions returns SQL_TRUE (supported) for SQL_API_SQLDESCRIBEPARAM. When EnableExtendedStmtInfo is cleared – ODBC Driver for Teradata does not request or use extended statement information from the database, even if supported. If extended statement information is unavailable, SQLDescribeParam is not supported and SQLGetFunctions returns SQL_FALSE (not supported) for SQL_API_SQLDESCRIBEPARAM.
Enable Client Side UDF Upload	Default=Cleared. If this is unchecked, the driver will not support UDF source file uploads. See the "EnableUDFUpload=[*Yes*|*No*] option in Teradata DSN Options.
UDF Upload Path	Fully qualified path where source files will be found. If defined, the driver looks at this location for files the database requests, unless the database gives a fully qualified path as part of the file name. No relative paths using “..” are allowed in this value. The default value displayed in the field (`Please enter the UDF folder path`) MUST be changed to the value you wish to use (either a valid path or empty). Delete the default value to leave the field empty. If **Enable Extended Statement Information** is checked: • The path you specify in **UDF Root Directory** will be prepended to all file names specified with an EXTERNAL NAME clause of a CREATE FUNCTION or REPLACE function.

Field, Check Box, or Button	**Description**
	- To use fully qualified file names in EXTERNAL NAME clauses, leave this field empty. See "UDFUploadPath=<***path***>" option in Teradata DSN Options.
Session Mode	Specifies the mode (Teradata or ANSI) for sessions on Teradata Database. The selected mode applies for the duration of the session. The default value is determined by the database based on the option used in the Teradata Database CREATE or MODIFY USER statement.
DateTime Format	Assigns the ANSI formats for DATE, TIME, and TIMESTAMP. The default setting is **AAA**. **IAA** is optional. Because the Integer data type has been deprecated for the TIME format, it is not recommended. For information, see Integer Time. See the “DateTimeFormat=[A|I]AA” option in Teradata DSN Options.
Return Generated Keys	Default = No Determines the result from requests that insert into identity columns (INSERT, INSERT … SELECT, UPSERT, MERGE-INTO). These requests can optionally return a result set containing identity column values (also known as auto-generated keys) for the inserted rows. Auto-generated key retrieval is not supported in Teradata Database versions prior to V2R6.2. The setting of Return Generated Keys has no effect when using a pre-V2R6.2 database server. Valid values are No, Identity Column, and Whole Row: - No = Auto-generated key retrieval is disabled (default) - Identity Column = Retrieve identity column only - Whole Row = Retrieve entire row(s) When Return Generated Keys is set to Identity Column or Whole Row, a request that inserts into tables containing identity columns returns two results: a row count with the number of inserted rows and a result set containing either the auto-generated keys as a single column or the complete rows inserted. The insert request becomes similar to a macro that first inserts and then selects the identity column or all columns of the rows just inserted.

Field, Check Box, or Button	Description
	When Return Generated Keys is set to No, the behavior of requests that insert into identity columns is not changed.
UPT Mode	Default = NOTSET Enables Unicode Pass Through Mode for the ODBC Application. The default value is NOTSET, which means that the UPT Mode set by the database is used; the ODBC Driver for Teradata does not send anything to the database when this option is set. UPTON: ODBC Driver for Teradata sends “SET SESSION CHARACTER SET UNICODE PASS THROUGH ON” to the database while connecting, thereby enabling UPT MODE for that session. UPTOFF: ODBC Driver for Teradata sends “SET SESSION CHARACTER SET UNICODE PASS THROUGH OFF” to the database while connecting, thereby disabling UPT MODE for that session.
Warning group box	
Advanced	Click to bring up the **Teradata ODBC Driver Advanced Options** dialog box. The dialog box contains further setting options available; however, it is *strongly recommended* NOT to change these settings.

Related Information:

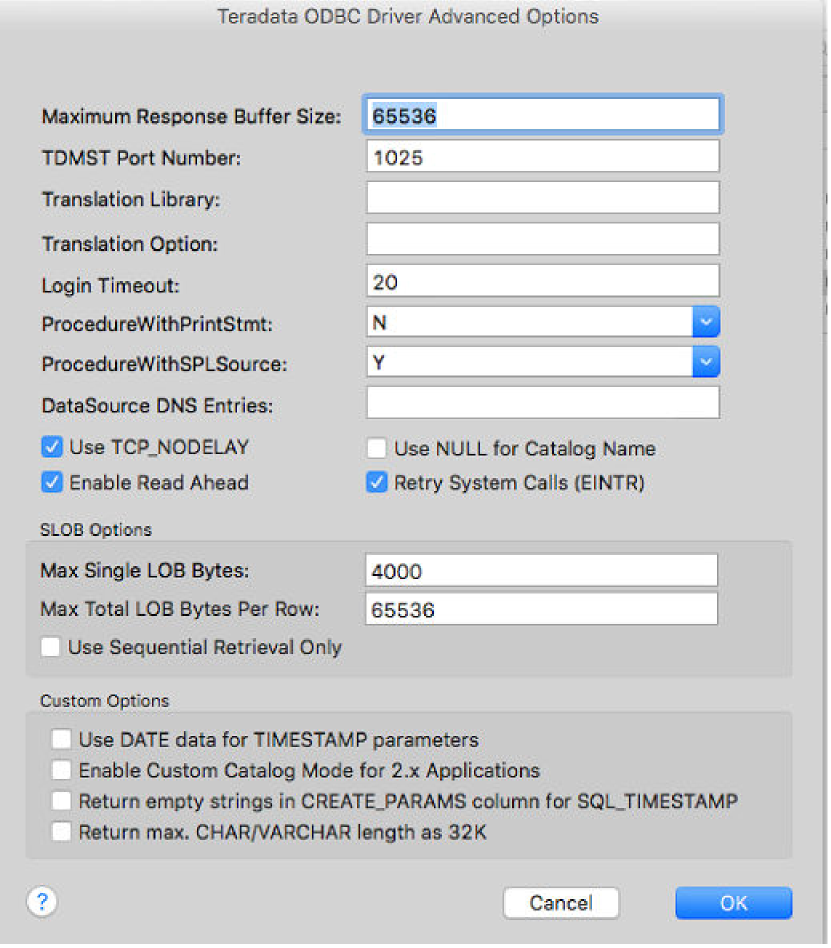
Configuring a DSN Using ODBC Administrator Tool
Teradata ODBC Driver Advanced Options

Teradata ODBC Driver Advanced Options

The **Teradata ODBC Driver Advanced Options** dialog box contains further setting options.

Note:

> It is strongly recommended NOT to change the default settings in the **Teradata ODBC Driver Advanced Options** dialog.



Teradata ODBC Driver Advanced Options

Maximum Response Buffer Size: 65536
TDMST Port Number: 1025
Translation Library:
Translation Option:
Login Timeout: 20
ProcedureWithPrintStmt: N
ProcedureWithSPLSource: Y
DataSource DNS Entries:

☑ Use TCP_NODELAY
☐ Use NULL for Catalog Name
☑ Enable Read Ahead
☑ Retry System Calls (EINTR)

SLOB Options

Max Single LOB Bytes: 4000
Max Total LOB Bytes Per Row: 65536
☐ Use Sequential Retrieval Only

Custom Options

☐ Use DATE data for TIMESTAMP parameters
☐ Enable Custom Catalog Mode for 2.x Applications
☐ Return empty strings in CREATE_PARAMS column for SQL_TIMESTAMP
☐ Return max. CHAR/VARCHAR length as 32K

Field, Check Box, or Button	Description
Maximum Response Buffer Size	Default = 65536 (64K) Enter the value used to try to limit the Teradata response buffer size for SQL requests. This value can be adjusted dynamically if Teradata cannot send a result within the limited packet size defined. See the “MaxRespSize=” option in Teradata DSN Options.
Redisplay Reconnect Wait	This feature has been deprecated for ODBC Driver for Teradata 14.10.
TDMST Port Number	Default = 1025 Lists the port number to use to access Teradata Database. Do not change this value unless instructed to do so by Technical Support.

Field, Check Box, or Button	Description
	See the “TDMSTPortNumber=<integer>” option in Teradata DSN Options.
Translation DLL Name	Specifies the translation DLL path. Translation DLL is used to convert between session character set and application character set.
Translation Option	Specifies the translation DLL option. The option is used by translation DLL.
Login Timeout	Default = 20 Defines a value corresponding to the number of seconds to wait when establishing a virtual circuit with Teradata for login. Enter an integer value. See Teradata DSN Options.
ProcedureWithPrintStmt	Default = N Activates the print option when creating stored procedures. See the “PrintOption=[N | P]” option in Teradata DSN Options.
ProcedureWithSPLSource	Default = Y Specifies the SPL option when creating stored procedures. See the “SplOption=[Y | N]” option in Teradata DSN Options.
Data Source DNS Entries	The **Data Source DNS Entries** DSN option notifies the ODBC Driver for Teradata how many entries are defined in DNS for the database name. The initial value of this option controls how the ODBC Driver for Teradata resolves database names to IP addresses. If this value is not set, the default value is undefined (empty). If multiple database names are provided in ODBC DSN, the **Data Source DNS Entries** option is applicable to all names. Note: If a database is identified by IP address instead of a name in the ODBC DSN or connection-string, the **Data Source DNS Entries** option is ignored. The database is identified in the **Name(s) or IP address(es)** field described in the table titled *ODBC Driver Setup (Apple OS X)*.

Field, Check Box, or Button	Description
	Data Source DNS Entries=undefined (default setting) is recommended for best results. This setting enables the ODBC Driver for Teradata to lookup DNS dynamically and find all available COPs for a given database name. Using this approach, ODBC Driver for Teradata will automatically detect new nodes added to the Teradata database (and DNS) in the future, without ODBC modification. **Data Source DNS Entries**= 0 indicates that DNS does not contain *cop* entries for the database name. The database name will only be resolved by itself. No attempt will be made to resolve using a cop suffix. This behavior can be desirable in an environment utilizing DNS to load balance. When DNS is used for load balancing, administrators can configure DNS to provide a different IP address or multiple IP addresses in different order each time the database name is resolved using DNS. **Data Source DNS Entries**= value. Entering a non-zero value indicates that DNS contains *cop* entries for the database name and the last cop entry is value. The first connection attempt will chose a random number between 1 and value. Each subsequent connection will then increment to the next number (round-robin). This approach will not encounter costly DNS resolution failures (how costly depends on how the DNS is configured). However, if additional entries are added to DNS at a later time, they will not be discovered by the ODBC Driver for Teradata unless the supplied value is increased.
Maximum Single LOB Bytes	Default=4000 Enter the value used to set the maximum size of any one SLOB (CLOB, BLOB) item.
Maximum Total LOB Bytes Per Row	Default=65536 Enter the value used to set the maximum size of all LOB bytes per row.
Enable DSN Tracing	The **Enable DSN Tracing** option controls whether DSN tracing is enabled or disabled. • Unchecked (default) = disabled • Checked = enabled **Path**: Specifies the absolute path of the trace file.

Field, Check Box, or Button	Description
	The default is `/tmp/ODBC.Trace.xxxxx`. If the entry is missing, a default pathname of `/tmp/ODBC.Trace.xxxxx` is used, where `xxxxx` is the pid of the creating process.
Use TCP_NODELAY	Default = Selected Valid for the Teradata DSN in ODBC Driver for Teradata. Transmission Control Protocol (TCP) provides an option called TCP_NODELAY to control the transmission of data. See the “TCPNoDelay=[Yes | No]” option in Teradata DSN Options.
Use NULL for Catalog Name	Default = Cleared When this option is selected – NULL values are assumed for the Catalog Name parameters in any of the Catalog API functions, even if the application passes a value. When this option is cleared and a value is passed for the Catalog Name parameter instead of NULL – ODBC Driver for Teradata returns an error because catalogs are not supported by Teradata Database.
Enable Read Ahead	Default = Selected When this option is selected - the ODBC Driver reads ahead by requesting the next response message from the database when the current response message being processed is not the last. The database can have one request active for each session at any point in time. An active request is either an SQL request which is executing or a request for the next part of the result from an earlier SQL request. When this option is cleared - the ODBC Driver only requests the next response message from the database when the current response message has been processed by the driver.
Retry system calls (EINTR)	Default = Selected When this option is selected – the ODBC Driver is responsible for retrying the socket system calls when they have been interrupted by some event such as a SIGALRM.

Field, Check Box, or Button	Description
	When this option is cleared – the ODBC Driver returns an SQL_ERROR to the ODBC Application. The ODBC Application is responsible for recovery from the interrupted socket system calls.
Custom options group box	
Use DATE data for TIMESTAMP parameters	Default = Cleared When this option is selected– ODBC Driver for Teradata is directed to send DATE data for parameters bound as SQL_C_TIMESTAMP and SQL_TIMESTAMP. This option should not be enabled for applications that are not using Microsoft Access Jet databases. This option should only be selected for this circumstance, as this results in truncation of SQL_C_TIMESTAMP data to contain only the DATE portion.
Enable Custom Catalog Mode for 2.x Applications	Default = Cleared Provides backwards compatibility for ODBC 2.x applications that have taken advantage of a defect in the ODBC Driver where the functionality of the Catalog APIs are noncompliant with the *ODBC Programmer's Reference* specification. The behavior when a NULL value is passed to the SQLTables API for the SchemaName argument results in a search for tables belonging to the userid, DBC, and default database schema names, rather than a % search pattern as noted in the *ODBC Programmer's Reference* specification.
Return Empty string in CREATE_PARAMS column for SQL_TIMESTAMP	Default = Cleared Returns an empty string for the CREATE_PARAMS column of SQLGetTypeInfo for SQL_TIMESTAMP data type, and disallows MC-ACCESS from using any TIMESTAMP precision value in Create Table text.
Return max. CHAR/ VARCHAR length as 32K	Default = Cleared Returns a value of 32000 (in general, could be 64000 also) for COLUMN_SIZE column of SQLGetTypeInfo for SQL_CHAR and SQL_VARCHAR data types. This allows

Field, Check Box, or Button	Description
	MS-ACCESS to handle column size value returned by ODBC Driver for Teradata without any numeric overflow.

User DSNs will be saved in the `~/Library/ODBC/odbc.ini` file.

System DSNs will be saved in the `/Library/ODBC/odbc.ini` file.

Note:

To be able to save a DSN, the user need write permission to the respective `odbc.ini` file.

Related Information:

Configuring a DSN Using ODBC Administrator Tool
Teradata ODBC Driver Options

Configuring a DSN Manually in odbc.ini

For information on configuring a DSN manually, see Configuration of odbc.ini in UNIX/Linux and Apple OS X.

Verifying Connection to the Teradata Database

Verify that the client system can connect over TCP/IP to the server. Use connection test program `tdxodbc` located in `/library/application support/Teradata/client/<TTU version>/bin`.

The sample test program can take the ODBC DSN name, database user-id, and password as input and establish a connection to the database. It can take basic SQL request input from the user, submit it to the database, and display the results. In addition to verifying connection to the Teradata database, `tdxodbc` can also determine whether or not the ODBC driver has been loaded correctly and report if there are any missing, required libraries.

Use this command to see the usage syntax of the test program:

```
/library/application support/Teradata/client/<TTU version>/bin/tdxodbc -h
```

5

Configuration of odbc.ini in UNIX/Linux and Apple OS X

ODBC.INI Structure

This section describes the `odbc.ini` file. For examples of the various sections in the `odbc.ini` file, see odbc.ini File Examples.

ODBC Driver for Teradata maps a logical Data Source Name (DSN) to the necessary software components and any additional information required for accessing data in a Teradata Database. The end user chooses the DSN that reflects the type of data or system it represents. The following table lists the sections that make up the odbc.ini file.

Section	Description
ODBC Options [ODBC] see ODBC Options Section	Specifies the ODBC Driver for Teradata installation directory and options.
ODBC Data Sources; [ODBC Data Sources] see ODBC Data Sources Section	Lists the name of each data source and describes its associated driver.
Data Source Specification [*<data source>*] see Data Source Specification Section	Provides an entry for each data source listed in the ODBC Data Sources section, with additional details about each data source.

Supported Releases

Minimum supported versions:

- iodbc-3.52 to 3.52.12
- unixODBC 2.3.2 to 2.3.4
- DataDirect Driver Manager 7.1.6 to 7.1.6.292

Specifying ODBC Driver Managers on Non-Windows Machines

Set the library path environment variable to verify your machine uses the correct ODBC driver manager to load the driver.

macOS

For macOS machines, set the `DYLD_LIBRARY_PATH` environment variable to include the paths to the ODBC driver manager libraries. For example, if the libraries are installed in `/usr/local/lib`, run the following command to set DYLD_LIBRARY_PATH for the current user session:

```
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/usr/local/lib
```

For information about setting an environment variable permanently, refer to the macOS shell documentation.

Linux

For Linux machines, set the `LD_LIBRARY_PATH` environment variable to include the paths to the ODBC driver manager libraries. For example, if the libraries are installed in `/usr/local/lib`, run the following command to set LD_LIBRARY_PATH for the current user session:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

For information about setting an environment variable permanently, refer to the Linux shell documentation.

Troubleshooting

An error message occurs when the name of the library file for the driver manager is different than the default. To resolve this issue:

1. Confirm the name of the library file used by your driver manager.
2. In a text editor, open the **teradata.teradataodbc.ini** file. This file is located in ***[InstallDir]*/lib** by default.
3. Add the following line to the end of the file, in which ***[DMLibFile]*** is the name of the library file:

   ```
   ODBCInstLib=[DMLibFile]
   ```

4. Save the **teradata.teradataodbc.ini** file.

ODBC Options Section

The ODBC Options section [ODBC] of the ini file specifies whether the ODBC driver manager options, such as tracing, are enabled or not. This format of this section is:

```
[ODBC]
keyword=value
```

On UNIX/Linux, DataDirect ODBC DM finds the location of its supporting libraries and message files using the path value of the InstallDir keyword under this section. On UNIX/ Linux, for DataDirect ODBC DriverManager, `InstallDir` is needed at a minimum under the [ODBC] section.

The following table lists applicable keywords under the [ODBC] section.

Keyword	Description
InstallDir=<*filepath*>	Note: InstallDir is applicable only on UNIX/Linux for DataDirect DriverManager. Specifies the installation directory for ODBC Driver for Teradata. An example default installation directory is `/opt/teradata/client/ODBC_64`. A common error encountered while running an application is the inability to load ODBC Driver for Teradata. The cause is usually the fact that the exact location of ODBC Driver for Teradata has not been specified in the `odbc.ini`file. The `odbc.ini` file *must* reference the exact location of ODBC Driver for Teradata.
Trace=[0] or [1]	Enables Driver Manager tracing. • 0 (default) – disabled • 1 – enabled
TraceDll=<*tracedll pathname*>	Note: TraceDLL is the keyword for DataDirect DM on UNIX/Linux. It is not applicable on Apple OS X. Specifies the path of the trace DLL, which is located in the Driver Manger library: • IBM AIX, Linux, and Solaris (default): `<InstallDir>/lib/odbctrac.so`
TraceFile=<*filepath*>	Specifies the path of the file where trace information is logged.
TraceAutoStop=[0] or [1]	Disables tracing when SQLFreehandle for ENV is called. • 0 (default) – disabled • 1 – enabled

ODBC Data Sources Section

Each entry in the ODBC Data Source section [ODBC Data Sources] of the `odbc.ini` file lists a DSN and its associated ODBC Driver for Teradata. The following table provides applicable keywords.

The section format is:

```
[ODBC Data Sources]
 data-source-name=<driver>
```

Keyword	Description
data-source-name=<*driver*>	[Required] The name of ODBC Driver for Teradata in the [ODBC Data Sources] section of `odbc.ini`. Use the following driver file extensions: • Solaris, Linux, IBM AIX- Use the`.so` extension. • Apple OS X – Use the `.dylib` extension. You can use this keyword to configure multiple data sources for the same system so different default databases or keywords can be associated with each data source.

Data Source Specification Section

Each data source listed in the ODBC Data Sources section [<*data-source-name*>] of the `odbc.ini` file includes its own Data Source Specification section. These sections have the following format:

```
[<data-source-name>]
Driver=<driver-path>
Keyword=<attribute>
```

The lines following the data source name define the required and optional attributes.

The following table contains the available options that can be entered into the Data Source Specification section.

Keyword/Synonym	Description
Driver=<*driver-path*>	[Required] The full path to the ODBC Driver for Teradata shared objects. • Solaris, Linux, IBM AIX – `<Install_Dir>/teradata/client/ODBC/lib/tdataodbc_sbu.so`

Keyword/Synonym	Description
	- Apple OS X – `/Library/Application Support/Teradata/Client/ODBC/lib/tdataodbc_sbu.dylib`
Description=<*data-source-desc*>	[Optional] Descriptive text about the data source.
DBCName=<*IP-addr-or-alias*>	[Required] The IP address or FQDN (fully qualified domain name) of the Teradata server. When a name is specified, ODBC Driver for Teradata automatically detects associated COP entries. For example, if the name contains a COPx suffix. For more information, see Cop Discovery.

Username=<*name*>
Or
UID=<*name*>
	[Optional] The default username for logging onto a Teradata server system.

Password=<*password*>	[Optional] The password required for the default Username. ***NOTICE*** The `odbc.ini` file has no password security. The password is retained in unencrypted plain text and can be viewed by any user with read-access to the file. Using this feature is not recommended, because it is deprecated. For more information, see Deprecated SQL Transformations. You can use a Teradata Wallet reference string instead of a plain text password by specifying the `$tdwallet()` token. For example: `Password=$tdwallet(WalletRefString)` For more information, see Teradata Wallet.

DefaultDatabase=<*database-name*>
Or
Database=<*database-name*>
	[Optional] The default database associated with the specified data source.
	When no value is specified for this field, it is assigned the value of the user name as entered into the Username field.

Keyword/Synonym	Description
	This field entry can be overridden when a new connection is specified. All catalog functions are associated with the default database if a table owner is not specified.
UPTMode	Default = NOTSET Enables Unicode Pass Through Mode for the ODBC Application. The default value is NOTSET, which means that the UPT Mode set by the database is used. The ODBC Driver for Teradata does not send anything to the database when this option is set. UPTON: ODBC Driver for Teradata sends “SET SESSION CHARACTER SET UNICODE PASS THROUGH ON” to the database while connecting, thereby enabling UPT MODE for that session. UPTOFF: ODBC Driver for Teradata sends “SET SESSION CHARACTER SET UNICODE PASS THROUGH OFF” to the database while connecting, thereby disabling UPT MODE for that session.

ODBC Administration

Modifying the odbc.ini File

The `odbc.ini` contains the names and descriptions of the data sources available to users. Edit this file to add, delete, or change ODBC data sources.

Adding a Data Source

When adding or configuring a data source, the **`odbc.ini`** file is edited using a text editor.

Each data source listed in the ODBC Data Sources section of the **`odbc.ini`** file must have its own Data Source Specification section. See ODBC Data Sources Section and Data Source Specification Section.

Note:

The example in this procedure displays the file extension convention for Solaris, Linux, and IBM AIX (`.so`). The file extension for Apple OS X is `.dylib`.

---

1. Open the `odbc.ini` file using any text editor.
2. Add an entry for the new data source and its corresponding driver in the [ODBC Data Sources] section.

   For example, enter the following to assign a data source called [*financial*] and an ODBC driver called `tdataodbc_sbu.so`:

   `financial=tdataodbc_sbu.so`
3. Add a descriptive entry for each data source listed in the [ODBC Data Sources] section to the Data Source Specification section.

   For example, to set up a Data Source Specification section called [*financial*], specify the location of ODBC Driver for Teradata and add a description of the driver, similar to the following:

   ```
   [financial]
   Driver=/opt/teradata/client/ODBC_64/lib/tdataodbc_sbu.so
   Description=Teradata 5550H running Teradata Database
   ```

Setting ODBCINI

The default location of the `odbc.ini` file is as follows:

Operating System	Default Location
UNIX/Linux	`$HOME/.odbc.ini(dot odbc.ini)`
Apple OS X	`$HOME/Library/ODBC/odbc.ini`

The directory (or directories) where ODBC Driver for Teradata is installed, contain a sample `odbc.ini` file. This sample contains a basic template of `odbc.ini` and sample DSN called `testdsn`. You can use this sample as a reference, or update and copy it to the default location.

Location of sample `odbc.ini`:

Operating System	Location
UNIX/Linux (32-bit applications)	`<`*install dir*`>/teradata/client/ODBC_32/odbc.ini`
UNIX/Linux (64-bit applications)	`<`*install dir*`>/teradata/client/ODBC_64/odbc.ini`

Operating System	Location
Apple OS X (32-bit and 64-bit applications)	`/Library/Application Support/Teradata/Client/ODBC/ odbc.ini`

Optionally, *`ODBCINI`* environment variable may point to the `odbc.ini` file present in non-default location.

For example, to use `odbc.ini` present in `/usr/dev/odbc.ini`:

```
export ODBCINI=/usr/dev/odbc.ini
```

Teradata DSN Options

The keyword options in the following table can be added to the Data Source Specification and Default Data Source Specification sections of the `odbc.ini` file. The options can only be configured through the `odbc.ini` file. The following table lists the options that are configurable in the `odbc.ini` file.

ODBC Driver for Teradata has established specific application uses for available DSN settings. For details, see DSN Settings for Third-Party Applications.

Keyword/Synonym	Description

AccountStr=<*`account`*>
Or
Account=<*`account`*>
	Specifies the account value to be entered during database logon. If unspecified, the database defaults to the account value specified when the user was created or modified.
	Note:
	This option can help isolate users by determining the applications users are running or restricting users from logging on.

CharacterSet=<*`charset name`*>
Or
Charset=<*`charset name`*>
	Specifies the session character set. Default is ASCII.
	Specify the character set for the session. It is strongly recommended to use the default ASCII session only for 7-bit ASCII characters. UTF8 is the recommended default session character set for all languages including US English. To use a

Keyword/Synonym	Description
	different character set than is chosen by default, specify or select it here. The available options: • ASCII • UTF8 • UTF16 • LATIN1252_0A • LATIN9_0A • LATIN1_0A • Shift-JIS (Windows, DOS compatible, KANJISJIS_0S) • EUC (Unix compatible, KANJIEC_0U) • IBM Mainframe (KANJIEBCDIC5035_0I) • KANJI932_1S0 • BIG5 (TCHBIG5_1R0) • GB (SCHGB2312_1T0) • SCHINESE936_6R0 • TCHINESE950_8R0 • NetworkKorean (HANGULKSC5601_2R4) • HANGUL949_7R0 • ARABIC1256_6A0 • CYRILLIC1251_2A0 • HEBREW1255_5A0 • LATIN1250_1A0 • LATIN1254_7A0 • LATIN1258_8A0 • THAI874_4A0 **Note:** For user-defined session character sets not shown on the above list, type the name of the user-defined session character set.
DateTimeFormat=[*A*|*I*]*AA*	Specifies the format of DATE, TIME, and TIMESTAMP data types. • A - ANSI • I - Integer The three-character specification represents:

Keyword/Synonym	Description
	• First character = DATE format • Second character = TIME format • Third character = TIMESTAMP format The recommended settings are either the **AAA** (default), or the **IAA** (optional) formats. Because the Integer data type has been deprecated for the TIME format, it is not recommended. For information, see Integer Time. The last character that represents TIMESTAMP is always ANSI.
DataSourceDNSEntries	The **DataSourceDNSEntries** DSN option notifies the ODBC Driver for Teradata how many entries are defined in DNS for the database name. The initial value of this option controls how the ODBC Driver for Teradata resolves database names to IP addresses. If this value is not set, the default value is undefined (empty). If multiple database names are provided in ODBC DSN, the **DataSourceDNSEntries** option is applicable to all names. **Note:** If a database is identified by IP address instead of a name in the ODBC DSN or connection-string, the **DataSourceDNSEntries** option is ignored. **DataSourceDNSEntries**=undefined (default setting) is recommended for best results. This setting enables the ODBC Driver for Teradata to lookup DNS dynamically and find all available COPs for a given database name. Using this approach, ODBC Driver for Teradata will automatically detect new nodes added to the Teradata database (and DNS) in the future, without ODBC modification. **DataSourceDNSEntries**= 0 indicates that DNS does not contain ***cop*** entries for the database name. The database name will

Keyword/Synonym	Description
	only be resolved by itself. No attempt will be made to resolve using a cop suffix. This behavior can be desirable in an environment utilizing DNS to load balance. When DNS is used for load balancing, administrators can configure DNS to provide a different IP address or multiple IP addresses in different order each time the database name is resolved using DNS. DataSourceDNSEntries= value. Entering a non-zero value indicates that DNS contains ***cop*** entries for the database name and the last cop entry is value. The first connection attempt will chose a random number between 1 and value. Each subsequent connection will then increment to the next number (round-robin). This approach will not encounter costly DNS resolution failures (how costly depends on how the DNS is configured). However, if additional entries are added to DNS at a later time, they will not be discovered by the ODBC Driver for Teradata unless the supplied value is increased.

DontUseHelpDatabase=[*Yes*|*No*]
Or
DontUseHelpDB=<[*Yes*|*No*]
	Specifies whether the Help Database is used.
	• No (default) - The driver uses the HELP DATABASE command.
	• Yes - SQLTables uses a SELECT statement instead of the HELP DATABASE command when no wildcard characters are used in SQLTables.
	Note:
	SQLTables uses `dbc.tables` or `dbc.tablesx`, depending on the UseXViews setting.

DontUseTitles=[*Yes*|*No*]	Specifies whether column names or column titles are returned.

Keyword/Synonym	Description
	• No – Returns column titles, if they are defined; otherwise, returns column names. • Yes (default) – Returns column names rather than column titles, as required by some applications, such as Crystal Reports. Column titles for SQLColumns are shown in the LABEL column.
EnableUDFUpload=[*Yes*|*No*]	Specifies whether the ODBC driver will support UDF source file uploads. • Yes – UDF source files are uploaded. • No (default) – UDF source files are not uploaded. For more information, see *ElicitFile* in *Teradata® Call-Level Interface Version 2 Reference for Mainframe-Attached Systems*, B035-2417, and *Teradata® Database SQL External Routine Programming*, B035-1147.
EnableExtendedStmtInfo=[*Yes*|*No*]	Specifies whether extended statement information is used when it is available from the database. • Yes (default) – Extended statement information is requested and used. If extended statement information is available, the ODBC API function SQLDescribeParam is supported and SQLGetFunctions returns SQL_TRUE (supported) for SQL_API_SQLDESCRIBEPARAM. • No – Extended statement information is not used, even if the database supports it. If extended statement information is not available, SQLDescribeParam is not supported and SQLGetFunctions returns SQL_FALSE (not supported) for SQL_API_SQLDESCRIBEPARAM.

Keyword/Synonym	Description
	Note: Teradata Database versions V2R6.2 and up support extended statement information, which includes additional metadata for parameters used in SQL requests and for columns in result sets.
EnableReadAhead=[*Yes*|*No*]	Specifies whether the ODBC Driver performs read-ahead to receive the next response message while the current message is being processed. • Yes (default) - the ODBC Driver reads ahead by requesting the next response message from the database when the current response message being processed is not the last. The database can have one request active for each session at any point in time. An active request is either an SQL request which is executing or a request for the next part of the result from an earlier SQL request. • No - the ODBC Driver only requests the next response message from the database when the current response message has been processed by the driver.
IANAAppCodePage=<ODBC application code page>	The current ODBC application code page is defined as IANAAppCodePage. See ODBC Application Code Page Values (Linux/UNIX and Apple OS X) for a list of valid ODBC application code page values and cautionary information.

IgnoreODBCSearchPattern=[*Yes*|*No*]
Or
IgnoreSearchPat=<[*Yes*|*No*]
	Specifies that characters _ and % work like regular wildcard characters for values given to table names, schema names, and so forth when passed to catalog functions, such as SQLTables.

Keyword/Synonym	Description
	This option is useful for applications, such as Microsoft Access, that do not support search patterns. • No (default, except for Microsoft Access) - Characters _ and % are processed as regular characters. **Note:** This setting causes Microsoft Access to use the Data Source section of the **`odbc.ini`**file to process search patterns. • Yes - Characters _ and % are not processed. Instead use the following commands for a normal search pattern: SQLTables SQLColumns SQLTablePrivileges SQLProcedures SQLProcedureColumns SQLGetInfo **Note:** SQL_SEARCH_PATTERN_ESCAPE returns an empty string.
integer≥0>LoginTimeout=<	Defines the number of paused seconds before a virtual circuit is established with Teradata Database for login. Default is 20. Enter an integer value greater than or equal to 0.
MaxRespSize=<*integer*≤16775168>	Limits the Teradata response buffer size for SQL requests. Default is 65536 (64K). The maximum integer value is 16775168. This value can be adjusted dynamically if Teradata cannot send a result within the limited packet size defined:

Keyword/Synonym	Description
	• If using a slow TCP/IP interface, such as PPP or SLIP, enter a smaller value. • If expecting large result sets in a LAN environment, enter a larger value.

MechanismName=<***MechanismName***>
Or
Authentication=<***MechanismName***>
	Identifies the authentication mechanism used for connections to the data source.
	Default is determined by a configuration option that is set by the TeraGSS program in an XML file called **`tdgssconfigure`**.
	Valid values are as follows:
	• Empty - The same as omitting the keyword.
	• TD2 - Selects Teradata 2 as the authentication mechanism. Username and password are required.
	• TDNEGO – selects one of the Authentication Mechanisms automatically based on the policy without user involvement.
	• LDAP - Selects LDAP as the authentication mechanism. The application provides the username and password.
	• KRB5 - Selects Kerberos as the authentication mechanism. The application provides the username and password.
	See Network Security for complete descriptions of authentication mechanisms.

MechanismKey=<***Value***>
Or
AuthenticationParameter=<***Value***>
	Value = ***string***
	A string of characters regarded as a parameter to the authentication mechanism. It is opaque for ODBC Driver for Teradata and is passed on to the Teradata authentication software called to set the mechanism.

Keyword/Synonym	Description
	NOTICE The odbc.ini file has no security. The MechanismKey is retained in unencrypted plain text and can be viewed by any user with read-access to the file. You can use a Teradata Wallet reference string instead of a plain text `MechanismKey` value by specifying the `$tdwallet()` token. For example: `MechanismKey=$tdwallet(RefString)` For more information, see Teradata Wallet.
NoScan=[*Yes*|*No*]	Enables or disables parsing of SQL statements by ODBC Driver for Teradata. When enabled, the driver transforms ODBC escape sequences to SQL. • No (default) - SQL statements are parsed by ODBC Driver for Teradata. • Yes - SQL statements are sent unmodified to Teradata Database without parsing by ODBC Driver for Teradata. **NOTICE** If the SQL statements contain ODBC-specific syntax, do not enable this option. Setting this option while using ODBC-specific syntax in the SQL statement results in Teradata Database reporting errors.
PrintOption=[*N* |*P*]	Specifies the print option for stored procedures. N (default) - Disables the print option when stored procedures are created. P - Enables the print option.

Keyword/Synonym	Description
retryOnEINTR	Controls whether ODBC Driver for Teradata retries the socket system calls on an EINTR or returns an SQL_ERROR. The affected socket system calls are as follows: • connect() • select() • recv() • send() Values are `Yes` for retries or `No` for no retries. Default is `Yes`.
ReturnGeneratedKeys=<*value*>	Determines the result from requests that insert data into identity columns (INSERT, INSERT ... SELECT, UPSERT, MERGE-INTO). These requests can optionally return a result set containing identity column values (also known as auto-generated keys) for the inserted rows. Auto-generated key retrieval is not supported in Teradata Database versions prior to V2R6.2 and the setting of ReturnGeneratedKeys has no effect when using a pre V2R6.2 database server. • C - Retrieves identity column only. Returns a row count of inserted rows and a result set that contains the auto-generated keys as a single column. • R - Retrieves entire rows. Returns a row count of inserted rows and a result set that contains the auto-generated keys of all columns of the rows just inserted. • N (default) or not set - No auto-generated key retrieval. The behavior of requests that insert into identify columns is unchanged.
SessionMode=[*Yes*|*No*]	Specifies the mode (Teradata or ANSI) for sessions on Teradata Database. The selected mode applies for the duration of the session.

Keyword/Synonym	Description
	The default value is determined by the database based on the option used in the CREATE or MODIFY USER statement. **Note:** An application cannot set SessionMode programmatically. SessionMode can be set only while connecting.
SplOption=[*Y* | *N*]	Specifies the stored procedure language (SPL) option when creating stored procedures. • N - Created without SPL text. • Y (default) - Created with SPL text.
TCPNoDelay=[*Yes*|*No*]	Specifies whether Transmission Control Protocol (TCP) immediately sends small packets or waits to gather packets into a single, larger packet. This option is valid for the Teradata Data Source Entry. • Yes (default) - TCP immediately sends small packets. This option avoids transmission delays so a larger number of small packets, including acknowledgments, can be sent over the network. • No - TCP gathers small packets into a single packet. This option can reduce network traffic, but it can also delay packet transmission. Refer to the TCP documentation for complete information.
TranslationDLL=<*path*>	Specifies the name of the translation DLL. It is recommended that you assign a fully qualified pathname for the translation DLL. User-defined session character sets can be used without a value for this option. Conversion is then based on the current application code page.

Keyword/Synonym	Description
	For details, see User-Defined Session Character Set Support and Translation DLLs.
TranslationOption=<*integer*>	A 32-bit value with a specific meaning for a given translation DLL. For example, it could specify a certain character set translation. If unspecified, a zero value is passed as an option to SQLDriverToDataSource and SQLDataSourceToDriver of the Translation DLL.
TDMSTPortNumber=<*integer*>	Specifies the number of the port that accesses Teradata Database. Default is 1025. **Note:** Do not change this value unless instructed to do so by Technical Support.
UDFUploadPath=<*path*>	Specifies the fully qualified path where UDF source files will be found. If defined, the driver looks at this location for files the database requests, unless the database gives a fully qualified path as part of the file name (field must be empty). For more information, see *ElicitFile* in *Teradata® Call-Level Interface Version 2 Reference for Mainframe-Attached Systems*, B035-2417, and *Teradata® Database SQL External Routine Programming*, B035-1147.
USE2XAPPCUSTOMCATALOGMODE=[*Yes*|*No*] Or 2XAPPCUSTOMCATALOGMODE=[*Yes*|*No*]	Provides backwards compatibility for ODBC 2.x applications that use a noncompliant search patterns. Earlier versions of ODBC Driver for Teradata allowed users to create search patterns other than the % search pattern stated in the *ODBC Programmer's*

Keyword/Synonym	Description
	Reference specification. On noncompliant systems, if a NULL value is passed to the SQLTables API for the SchemaName argument, the result is a search for tables by userid, DBC, and default database schema names, rather than the % search pattern. • No (default) – Use the % search pattern. • Yes – Allow searches by userid, DBC, and default database schema names.

UseDataEncryption=[***Yes***|***No***]
Or
DataEncryption=[***Yes***|***No***]
	No (default) – Encrypt only logon information.
	• Enables the Teradata gateway and ODBC Driver for Teradata to communicate in an encrypted manner.
	• Yes – Enable data encryption.

UseXViews=[***Yes***|***No***]	Specifies whether X tables are used. X tables only contain information that users have permission to access. These tables are optional for Teradata, so check to ensure they exist before using the option. • Yes – ◦ SQLTables() and SQLProcedures() use dbc.tablesVX and dbc.databasesVX. ◦ SQLColumns() and SQLProcedureColumns() use dbc.columnsVX instead of dbc.columnsV. ◦ SqlStatistics() uses dbc.tablesizeVX. • No (default) – ◦ SQLTables() and SQLProcedures() use dbc.tablesV and dbc.databasesV. ◦ SQLColumns() and SQLProcedureColumns() use dbc.columnsV.

Keyword/Synonym	Description
	SqlStatistics() uses dbc.tablesizeV.
The Linux/UNIX system version of ODBC Driver for Teradata recognizes the following options when the LANG environment variable contains a value recognized by ODBC Driver for Teradata for Japanese, Chinese, or Korean locales. These options are not available on Apple OS X.	
ClientKanjiFormat={SJIS|EUC|Big5|Network Korean| GB}	Note: Using this feature is not recommended, because it is deprecated. Specifies which character set to use for the user's choice of character set format: • SJIS • EUC • Big5 • Network Korean • GB Default is set by the user. The data returned is translated from the session character set to the ClientKanjiFormat for SJIS and EUC only. For Big5, NetworkKorean, and GB, this value should match the session character set.

DSN Tracing Attributes

Enable logging in the driver as described in the section below.

For more information on tracing, refer to New Teradata ODBC Driver Compatibility Reference.

Important:

Using any of the tracing options will significantly degrade performance.

Configuring Logging Options on a Non-Windows Machine

To help troubleshoot issues, you can enable logging in the driver. Logging is configured through driver-wide settings in the `teradata.teradataodbc.ini` file, which apply to all connections that use the driver.

Note:

> Using any of the logging options will significantly degrade performance. Enable logging only long enough to capture an issue.

---

1. Open the `teradata.teradataodbc.ini` configuration file in a text editor.
2. Set the **LogLevel** setting to one of the following values:

LogLevel Value	Description
0	Disables all logging.
1	Logs severe error events that lead the driver to abort.
2	Logs error events that might allow the driver to continue running.
3	Logs events that might result in an error if no action is taken.
4	Logs general information describing the progress of the driver.
5	Logs detailed information useful for debugging the driver.
6	Logs all driver activity.

3. Set the **LogPath** key to the full path of the target folder where you want to save the files.
4. Set the **LogFileCount** key to the maximum number of log files to keep.
   After the maximum number of log files is reached, the driver deletes the oldest log files each time an additional file is created.
5. Set the **LogFileSize** key to the maximum size of each log file in megabytes.
   After the maximum file size is reached, the driver creates a new file and continues logging.
6. Save the `teradata.teradataodbc.ini` configuration file.
7. Restart your ODBC application to verify the new settings.
   The new Teradata ODBC Driver produces two log files at the location you specify using the **LogPath** key, where *`[DriverName]`* is the name of the driver:
   - A *`[DriverName]`*`_driver.log` file that logs driver activity that is not specific to a connection.

- A `[DriverName]_connection_[Number].log` for each connection made to the database, where `[Number]` is a number that identifies each log file. This file logs driver activity that is specific to the connection.

Disabling Logging on a Non-Windows Machine

1. Open the `teradata.teradataodbc.ini` configuration file in a text editor.
2. Set the **LogLevel** key to 0.
3. Save the `teradata.teradataodbc.ini` configuration file.
4. Restart your ODBC application to verify the new settings.

6

ODBC Application Development

Overview

This section provides information needed to use ODBC Driver for Teradata with custom and off-the-shelf ODBC applications, relative to the published ODBC standard. The following information is covered.

Section Heading	Contents
Software Development Kits	SDKs required for developing ODBC applications
ODBC Conformance	The level(s) to which ODBC Driver for Teradata implements the ODBC specification
ODBC SQL Grammar	ODBC SQL grammar and core compliance
ODBC Connection Functions and Dialog	The ODBC Driver for Teradata API defines three connection functions for establishing a connection to a database
ANSI SQL 1992 Syntax	How ODBC Driver for Teradata supports the entry level ANSI SQL 1992 syntax
Keywords for SQLDriverConnect() and SQLBrowseConnect()	Attributes that can be used with SQLDriverConnect() and SQLBrowseConnect()
ODBC Pattern Escape Character	The ODBC pattern escape character
Large Objects	Support for Large Objects, including BLOB and CLOB
User-Defined Functions	Support for User-Defined Functions
User-Defined Types and User-Defined Methods	Support for User-Defined Types and User-Defined Methods
Parameter Arrays	Using parameter arrays to reduce network traffic
Large Decimal and BIGINT Support	Support for Large Decimal and BIGINT
64-bit Support	Useful references when working in 64-bit

Software Development Kits

Windows Application Development

Microsoft Visual Studio 2012 is the recommended compiler to use when developing custom Windows applications with 32-bit and 64-bit Windows operating systems.

UNIX OS Application Development

For UNIX OS application development, an SDK is provided as part of the UNIX system versions of ODBC Driver for Teradata. During the installation of the UNIX system versions of ODBC Driver for Teradata, header and library files for the SDK are installed.

The following table lists compilers for UNIX platforms that ODBC Driver for Teradata supports. These compilers are recommended for use when developing custom applications.

Operating System	Compiler
IBM AIX	IBM XL C/C++ Enterprise Edition for AIX, v9.0
Oracle Solaris on SPARC systems	Forte Developer 6 update 2 C++ 5.3
Oracle Solaris on AMD Opteron systems	Oracle Solaris Studio 11 C++ 5.8
Red Hat Linux (32-bit)	g++ (GCC) 3.4.6 20060404 (Red Hat 3.4.6-8)
Red Hat Linux (64-bit)	g++ (GCC) 3.4.6 20060404 (Red Hat 3.4.6-3)

Apple OS X Application Development

ODBC Application SDK comes with Xcode. Xcode, a suite of software development tools, can be downloaded from Apple.

UNIX OS Compilation Options

The ODBC include files must be used when developing ODBC applications on the UNIX systems. These files are located in:

```
<InstallDir>/include
```

The inclusion of the ODBC Driver Manager requires some changes to compilation options for your applications.

As an example, look at:

`<InstallDir>/samples/C/Makefile`

or

`<InstallDir>/samples/C++/Makefile`

and corresponding platform.include files for compilation options for each UNIX platform.

If compiling C++ applications on any UNIX platform that uses ODBC Driver for Teradata, use the following compile time options:

```
-DODBCVER=0x0352
```

If compiling for C applications on an IBM AIX platform, use the following compile time options:

```
-DODBCVER=0x0352
-qlanglvl=EXTended
```

Apple OS Compilation Options

If compiling ODBC applications on OS X platform that uses ODBC Driver for Teradata, use the following compile time options:

```
-mmacosx-version-min=10.7 <- The minimum OS X version on which application can run
-DODBCVER=0x0350 <- ODBC Compliance level
```

ODBC Conformance

ODBC defines conformance levels for ODBC API and the ODBC SQL grammar. These conformance levels establish standard sets of functionality for using the ODBC drivers for Teradata Database. The drivers are compliant with ODBC 3.8 SDK core level 1.

ODBC API

This section lists the ODBC API functions that are supported in this release.

Core Level Functions

All ODBC 3.8 core level functions listed in the following table are supported. Following each function are any limitations, restrictions, or enhancements that exist in ODBC Driver for Teradata.

Function Name	Purpose
SQLAllocHandle	Allocates an environment, connection, statement, or descriptor handle

Function Name	Purpose
SQLCloseCursor	Closes a cursor that had been opened on a statement and discards the pending results
SQLColAttribute	Describes attributes of a column in the result set. Retrieving bookmark metadata on column 0 is *not* supported. By default, SQLDescribeCol and SQLColAttribute return the column name instead of the Teradata column title. If an application wants the driver to return the title instead of the actual column name, then the option UseColumnNames in the ODBC Driver for Teradata options dialog must be unselected for the DSN used, or ***`DontUseTitles`*** option in the **`odbc.ini`** for the UNIX OS must be No. Returning the column title instead of the actual column name might cause problems for certain applications, such as Crystal Reports, because the applications expect to see the column name and not the column title returned.
SQLConnect	Connects to a specific driver by data source name, user ID, and password
SQLCopyDesc	Copies descriptor information from one descriptor handle to another
SQLEndTran	Requests a commit or rollback operation for all active operations on all statements associated with a connection depending on ***`Completion type`*** parameter. SQLEndTran can also request that a commit or rollback operation be performed for all connections associated with an environment.
SQLExtendedFetch	Fetches the specified rowset of data from the result set and returns data for all bound columns. The parameters for this API specify that rowsets can be specified at an absolute or relative position, or by bookmark. ODBC Driver for Teradata only supports an orientation of SQL_FETCH_NEXT, and bookmarks are not supported.
SQLFetchScroll	Fetches the specified rowset of data from the result set and returns data for all bound columns. Though the parameters for this API are Position and Orientation, the 03.00.00 or higher versions of the driver support only SQL_FETCH_NEXT, and only read-only and forward-only cursors are supported.

Function Name	Purpose
	SQLFetchScroll can be called only while a result set exists–that is, after a call that creates a result set and before the cursor over that result set is closed. If any columns are bound, SQLFetchScroll returns the data in those columns. If the application has specified a pointer to a row status array or a buffer in which to return the number of rows fetched, SQLFetchScroll returns this information as well.
SQLFreeHandle	Releases an environment, connection, statement, or descriptor handle. This replaces SQLFreeEnv, SQLFreeConnect, SQLFreeStmt.
SQLGetConnectAttr	Returns the current setting of a connection attribute ODBC Driver for Teradata provides driver-specific connection attributes to retrieve Session character set, logical Host ID, and Session Number information to the application. ODBC Driver for Teradata defined attributes provided to support the functionality include: SQL_ATTR_TDATA_HOST_ID (13001)–the host-id is returned as an unsigned integer SQL_ATTR_TDATA_SESSION_NUMBER (13002)–the session number is returned as an unsigned integer SQL_ATTR_TDATA_SESSION_CHARSET (13003)–the session character set is retrieved as a character string SQL_ATTR_AGKR (13004)–an integer value that determines the result requests that insert into identity columns (INSERT, INSERT ... SELECT, UPSERT, MERGE-INTO). SQLGetConnectOption with 2.x application or SQLGetConnectAttr with 3.x application tests whether a particular connection is active or not. The option accepts the SQL_ATTR_CONNECTION_DEAD attribute also. This is an ODBC 3.51 attribute.
SQLGetDescField	Returns the current setting or value of a single field of a descriptor record
SQLGetDescRec	Returns the current settings or values of multiple fields of a descriptor record that describes the name, data type, and storage

Function Name	Purpose
SQLGetDiagField	Returns the current value of a field of a diagnostic record associated with a handle
SQLGetDiagRec	Returns the current values of a diagnostic record associated with a handle, including the SQLSTATE, the native error code, and the diagnostic message text SQLGetDiagRec replaced SQLError starting in ODBC 3.x. SQLError() sometimes returns a SQLSTATE of S1000 (general error) rather than a more-specific SQLSTATE. NativeError contains the precise Teradata error code, and the ErrorMessage contains the Teradata error message. If NativeError contains -1, the error was detected in ODBC Driver for Teradata.
SQLGetEnvAttr	Returns the current setting of an environment attribute
SQLGetStmtAttr	Returns the current setting of a statement attribute. A list of supported attributes is provided in DSN Tracing Attributes.
SQLSetConnectAttr	Sets a connection attribute to a specified value.
SQLSetDescField	Sets the value of a single field of a descriptor record
SQLSetDescRec	Sets the values of multiple fields of a descriptor record that describes the name, data type, and storage
SQLSetEnvAttr	Sets an environment attribute to a specified value
SQLSetStmtAttr	Sets a statement attribute to a specified value

SQLSetConnectOption() and SQLSetConnectAttr()

The following table contains the list of attributes for the SQLSetConnectOption or the SQLSetConnectAttr functions.

Parameter	Description
SQL_ACCESS_MODE	Supported
SQL_AUTOCOMMIT	Supported
SQL_LOGIN_TIMEOUT	Supported
SQL_TRANSLATE_DLL	Supported

Parameter	Description
SQL_TRANSLATE_OPTION	Supported
SQL_TXN_ISOLATION	Only supports SQL_TXN_SERIALIZABLE and SQL_TXN_READ_UNCOMMITTED
SQL_QUIET_MODE	Supported
SQL_CURRENT_QUALIFIER	Not supported
SQL_PACKET_SIZE	Not supported
SQL_ODBC_CURSORS	Driver Manager handles
SQL_OPT_TRACE	Driver Manager handles
SQL_OPT_TRACEFILE	Driver Manager handles

Statement Options Set in SQLSetStmtAttr()

The following table lists the Statement Options in SQLSetStmtAttr().

Statement	Support
SQL_QUERY_TIMEOUT	Supported
SQL_MAX_ROWS	Supported SQL_MAX_ROWS confines the results returned to a user-specified limit. This is implemented on a per SELECT statement basis and as such, might not reduce the network traffic for multi-statement requests due to the Teradata mechanism used for spool control. In multi-statement requests, network traffic cannot be avoided since the current result set must be scrolled through to get to the next. In single-statement requests or if within the last statement of a multi-statement request, canceling the request reduces network traffic. Reduce network traffic by fetching only the needed result rows of data. If all of the results from a single-statement request or from the last

Statement	Support
	statement of a multi-statement request are not needed, calling SQLFreeSTMT(SQL_CLOSE) discards all pending results.
SQL_NOSCAN	Supported The implementation of SQL_NOSCAN varies slightly from the ODBC definition. By default, requests are scanned (parsed) to convert ODBC SQL syntax to Teradata SQL syntax. If an application calls SQLSetStmtOption(hstmt, SQL_NOSCAN, SQL_NOSCAN_ON), ODBC Driver for Teradata does not parse SQL requests. The ODBC SDK mentions that only ODBC escape clauses are not scanned when SQL_NOSCAN_ON is set. For custom ODBC applications that generate only Teradata-specific SQL, set SQL_NOSCAN_ON to avoid parsing SQL requests and to improve performance.
SQL_MAX_LENGTH	Not supported
SQL_ATTR_ASYNC_ENABLE	Supported
SQL_BIND_TYPE	Only supports SQL_BIND_BY_COLUMN
SQL_CURSOR_TYPE	Only supports SQL_CURSOR_FORWARD_ONLY
SQL_CONCURRENCY	Only supports SQL CONCUR_READ_ONLY
SQL_KEYSET_SIZE	Only supports SQL_KEYSET_SIZE=0
SQL_ROWSET_SIZE	Only supports SQL_ROWSET_SIZE=1
SQL_SIMULATE_CURSOR	Only supports SQL_SC_NON_UNIQUE
SQL_RETRIEVE_DATA	Only supports SQL_RD_ON
SQL_USE_BOOKMARKS	Not supported
SQL_ATTR_ASYNC_ENABLE	Supported
SQL_ATTR_METADATA_ID	Supported

Statement	Support
SQL_ATTR_CONNECTION_POOLING	Supports SQL_CP_PER_DRIVER, SQL_CP_ONE_PER_HENV, and SQL_CP_OFF

Extension Level Functions

The following table lists the Extension Level Functions and their purpose.

Note:

> Functions from Versions 1.x and 2.x, which are changed in implementation due to new features or retained for backward compatibility:

Function Name	Purpose
SQLBindCol	Binds application data buffers to columns in the result set.
SQLBindParameter	Binds data buffers to parameter markers in an SQL statement. SQLBindParameter accepts a null pointer for the parameter data (rgbValue) when the parameter length (pcbValue) is equal to SQL_NULL_DATA. The maximum number of parameters supported by SQLBindParameter is 256. Stored procedures can contain up to 1024 parameters. Although Teradata Database stores and supports them, ODBC Driver for Teradata cannot execute stored procedures containing more than 256 parameters because of the SQLBindParameter restriction. Default parameter (pcbValue= SQL_DEFAULT_PARAM) for procedures (macros) that rely on resolving the bind at SQL_DATA_AT_EXEC is not implemented. Set the ColumnSize argument to SQLBindParameter to one of the following: • The size of the largest object expected to be transferred (This is recommended for improved performance.) • The size of the object being transferred. Note: For Unicode character columns, the largest ColumnSize is 32000 characters.

Function Name	Purpose
	Note: If the StrLen_or_IndPtr is NULL and ColumnSize is zero when binding to SQL character or binary, the driver does not report HY104 from SQLBindParameter, but truncates the data bound to the parameter with a warning during execution.
SQLBrowseConnect	Finds attributes and attribute values required to connect to a data source.
SQLCancel	Cancels the processing on a statement.
SQLColumnPrivileges	Returns a list of columns and associated privileges for one or more tables. This is a Level 2 feature that has been incorporated into the Core level driver because it replaces SQLColumnPrivilege in version 2.x. Since ANSI defines column-level privileges, ODBC Driver for Teradata returns the column privileges associated with a table. When all columns in a table have a particular privilege, ODBC Driver for Teradata returns the value ALL as the column name.
SQLColumns	Returns a list of column names in one or more tables SQLColumns returns column information for tables and views with up to 22 columns. SQLColumns requires SELECT privileges on the table(s) in question.
SQLDataSources	Returns information about the available data sources.
SQLDescribeCol	Returns the column name, type, column size, decimal digits, and nullability for a column in the result set. Retrieving bookmark meta data on column 0 is *not* supported. By default, SQLDescribeCol and SQLColAttribute return the column name instead of the Teradata column title. If an application wants ODBC Driver for Teradata to return the column title instead of the actual column name, then the option **Use Column Names** in the **Teradata ODBC Driver Options** dialog box must not be selected for the DSN used, or set DontUseTitles = No on the UNIX OS. Returning the column title instead of the actual column name can cause problems for certain applications, such as Crystal Reports, because they expect to get the column name and not the column title.

Function Name	Purpose
	SQLDescribeCol and SQLColAttribute return an SQL Type of SQL_TIME for columns in the result set that are defined as INTEGER FORMAT 99:99:99 or FLOAT FORMAT 99:99:99. These columns can be fetched into SQL_C_TIME data or any form where time data can be converted. SQLColAttribute returns that a column is a currency type if the DBC FORMAT for that column contains dollar signs, such as FORMAT '$$$,$$9.99'. This can be useful to help an application know how to format the result data.
SQLDescribeParam	Is supported by ODBC Driver for Teradata when the EnableExtendedStmtInfo feature is enabled. SQLDescribeParam returns metadata for a column or expression corresponding to a parameter marker associated with a prepared SQL statement. This metadata also is available in the fields of the Implementation Parameter Description (IPD). Refer to the ODBC Specification for a detailed description of SQLDescribeParam.
SQLDisconnect	Closes the connection associated with a connection handle.
SQLDriverConnect	Connects to a specific driver by connection string or requests that the Driver Manager and ODBC Driver for Teradata display connection dialog boxes.
SQLDrivers	Returns the list of installed drivers and their attributes.
SQLExecDirect	Executes an unprepared statement.
SQLExecute	Executes a prepared statement.
SQLFetch	Fetches the next rowset of data from the result set and returns data for all bound columns.
SQLForeignKeys	Returns a list of column names that make up foreign keys, if they exist for a table. This is a Level 2 implementation.
SQLGetCursorname	Returns the cursor name associated with a statement.
SQLGetData	Returns a part or all of one column of one row of a result set.
SQLGetFunctions	Returns information about the ODBC Driver for Teradata API functions supported.

Function Name	Purpose
SQLGetInfo	Returns general information about ODBC Driver for Teradata and the data source associated with a connection. Note: When connected to Teradata Database release 14.0 or later, running a query with SQLGetInfo(SQL_KEYWORDS) returns a list of keywords from the Teradata Database. This might cause it to run longer than it did in previous releases.
SQLGetTypeInfo	Returns information about the Data Types that the data source supports. Additionally, returns a value to describe the driver type.
SQLMoreResults	Determines whether there are more result sets available. If so, SQLMoreResults initializes processing for the next result set. This is a Level 1 Function.
SQLNativeSql	Returns the text of an SQL statement that ODBC Driver for Teradata translates.
SQLNumParams	Returns the number of parameters in an SQL statement.
SQLNumResultCols	Returns the number of columns in a result set.
SQLParamData	Supplies parameter data used by SQLPutData at statement execution time.
SQLPrepare	Prepares an SQL statement for execution.
SQLPrimaryKeys	Returns the column names that make up the primary key for a table.
SQLProcedureColumns	Returns the list of input and output parameters, the driver type, and the columns that make up the result set for the procedures.
SQLProcedures	Returns the list of procedure names stored in a data source.
SQLPutData	Sends a part or all of a data value for a parameter. SQLPutData() cannot handle putting parameter data in parts.
SQLRowCount	Returns the number of rows affected by an UPDATE, INSERT, or DELETE statement. After an SQLExecute or SQLExecDirect, SQLRowCount can be called to get the DBC Activity count for any type of SQL statement, even for SELECT statements.

Function Name	Purpose
	For portability, applications should not rely on this behavior, because other ODBC drivers might return -1 rather than the row count after a SELECT statement.
SQLSetCursorName	Associates a cursor name with an active statement
SQLSpecialColumns	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated. SQLSpecialColumns() works correctly for views only if the view is on a single table and there is exactly one unique index on the table. SQLSpecialColumns() works correctly for tables.
SQLStatistics	Returns statistics about a single table and the list of indexes associated with the table. SQLStatistics() retrieves information about the indexes on a table, and supplies cardinality statistics on the indexes, but the number of pages is estimated for indexes. The number of pages is returned correctly for the base table (a page is assumed to be 4096 bytes), but is set to 0 for views. SQLStatistics() supplies precise cardinality statistics on the table itself only if SQL_ENSURE is set; otherwise, the cardinality on the base table is set to the largest cardinality from any index on the table (usually accurate if there are any unique indexes). The only way to accurately get the cardinality of a table is for the driver to perform a SELECT COUNT(*) FROM table. Consequently, SQL_ENSURE can be quite slow. When the function SQLStatistics() is called with the parameter value SQL_ENSURE, it is only accurate for small tables. For larger tables, SQL_ENSURE could effectively make the table unusable for extended periods, so the SQL_QUICK method is used instead, which gives the approximate values for cardinality and pages that are sufficient for most applications. In Windows, SQLStatistics() returns index information on a specified table so you can read or update using a view on the table. User-defined index names, using ODBC grammar, which were specified to create the index are returned by

Function Name	Purpose
	SQLStatistics() as an index name if the index name is still defined in the DSN entry in the Registry (IndexName0-9). Using ODBC-style named indexes is not recommended, because it is deprecated. For details, see ODBC-Style Named Indexes (deprecated in 15.00). The Teradata index names are returned instead if: • The user did not specify an index name using ODBC grammar (Teradata SQL grammar was used instead). • The IndexName entry has been reused in the DSN entry. • The PC performing SQLStatistics() is not the same PC that was used when the index name was created. For example, an index name ***`MyIndex`*** was used to create an index on table ***`MyTable`*** using ODBC grammar syntax. Internally, Teradata created the index ***`MyTable001`*** for the table, but SQLStatistics() returns **`MyIndex`** as the index name instead of ***`MyTable001`*** since the DSN entry contains this index name for ***`MyTable`***. On all other PCs, ***`MyTable001`*** is returned as the index name.
SQLTablePrivileges	Returns a list of tables and the privileges associated with each table. This is a Level 2 feature. ODBC Driver for Teradata returns the information as a result set on the specified ***`hstmt`***. In the result set description are: PRIVILEGE: Identifies the table privilege. Can be one of the following or a data source-specific privilege. SELECT: The grantee is permitted to retrieve data for one or more columns of the table. INSERT: The grantee is permitted to insert new rows containing data for one or more columns into to the table. UPDATE: The grantee is permitted to update the data in one or more columns of the table. DELETE: The grantee is permitted to delete rows of data from the table.

Function Name	Purpose
	REFERENCES: The grantee is permitted to refer to one or more columns of the table within a constraint (for example, a unique, referential, or table check constraint). The scope of action permitted the grantee by a given table privilege is data source-dependent. For example, the UPDATE privilege might permit the grantee to update all columns in a table on one data source and only those columns for which the grantor has the UPDATE privilege on another data source.
SQLTables	Returns the list of table names stored in a data source SQLTables accepts types VIEW, TABLE, and SYSTEM TABLE to limit its display. Also, DBC tables and views are returned as SYSTEM TABLE types for easier processing by ODBC applications, unless you log in as DBC. If logged in as user DBC, DBC tables are returned as type TABLE. 'GLOBAL TEMPORARY' or 'TEMPORARY' can be specified as a table type. ODBC Driver for Teradata is not able to provide information on volatile tables because information on these is not kept in the Teradata data dictionary.

SQLTables

The 16.20 database has a new table called a *Primary Time Index* (PTI) table. For details about PTI tables, refer to *Detailed Feature External Specification for Time Series.* The TD ODBC Driver does not add any additional functionalities to handle this new table type, but the ODBC API function **SQLTables** now returns one additional column described in the following table:

Column Name	Column Number	Data Type	Comment
TD_TABLE_FLAVOR	6	varchar	Contains “`PRIMARY TIME INDEX`” for PTI tables. Null if table is not a PTI table.

SQLStatistics

The 16.20 database has a new table called a *Primary Time Index* (PTI) table. For more information, refer to *Detailed Feature External Specification for Time Series.* The TD ODBC

Driver does not add any additional functionalities to handle this new table type, but the ODBC API function SQLStatistics now returns two additional columns described in the following table:

Column Name	Column Number	Data Type	Comment
TD_TIMEZERO	14	varchar	The time zero point associated with the table. For example, **“`2015-10-22`”**. Null if the table is not a PTI table.
TD_TIMEBUCKET	15	varchar	The length of a time bucket. For example, **“`HOURS(1)`”**. Null if the table is not a PTI table.

SQLTablePrivileges

SQLTablePrivileges listed in the following table return all object-level privileges applicable to the specified tables or other database objects (macro, stored procedure, view, and so forth).

Privilege Name	Purpose
ALTER EXTERNAL PROCEDURE	Alter external procedure
ALTER FUNCTION	Alter function
ANY PRIVILEGE	All default privileges
ALTER PROCEDURE	Alter procedure
CREATE EXTERNAL PROCEDURE	Create an external procedure
CREATE FUNCTION	Create function
CREATE OWNER PROCEDURE	Create owner procedure
CREATE ROLE	Create new user role
CRETRIG	Create trigger
CHECKPT	Checkpoint
CRESPROC	Create procedure
DROP FUNCTION	Drop function

Privilege Name	Purpose
DROPTBL	Drop table
DROPTRIG	Drop trigger
DROPSPROC	Drop Procedure
DROPDB	Drop database/user
DROPMAC	Drop macro
DROPVW	Drop view
EXECUTE FUNCTION	Execute function
EXSPROC	Execute procedure
OVERRIDE INSERT POLICY FOR CONSTRAINT	Override insert policy for constraint
OVERRIDE UPDATE POLICY FOR CONSTRAINT	Override update policy for constraint
OVERRIDE DELETE POLICY FOR CONSTRAINT	Override delete policy for constraint
OVERRIDE SELECT POLICY FOR CONSTRAINT	Override select policy for constraint
OVERRIDE DUMP OBJECT	Override dump for object
OVERRIDE RESTORE OBJECT	Override restore for object
REPOVRRIDE	Replication Override
SHOW	Access column data
STATISTICS	Make table statistics
unknown	New privilege to Teradata Database
and SELECT, INSERT, UPDATE, DELETE, GRANT, DUMP, EXECUTE, RESTORE, REFERENCES, and INDEX.	

Unsupported Functions

The functions listed in the following table are not supported at this time.

Function Name	Description
SQLSetPos	Not supported
SQLSetScrollOptions	Not supported

Attributes

The following table lists the environment attributes that are supported in ODBC Driver for Teradata.

Environment Attribute	Values Supported	Values Not Supported
SQL_ATTR_CONNECTION_POOLING	SQL_CP_ONE_PER_DRIVER SQL_CP_ONE_PER_HENV	–
SQL_ATTR_ODBC_VERSION	SQL_OV_ODBC2 SQL_OV_ODBC3	–

The following table lists the connection attributes that are supported in ODBC Driver for Teradata.

Connection Attribute	Values Supported	Values Not Supported
SQL_ATTR_ACCESS_MODE	SQL_MODE_READ_ONLY SQL_MODE_READ_WRITE	–
SQL_ATTR_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF SQL_ASYNC_ENABLE_ON	–
SQL_ATTR_AUTOCOMMIT	SQL_AUTOCOMMIT_OFF SQL_AUTOCOMMIT_ON	In Teradata, a DDL stateme can be entered either as a si statement or as the last statement in a transaction. Catalog and information functions, such as `SQLGetInf` `SQLGetTypeInfo`, might issue requests when called. To pre an error returning from the database, DDL requests in manual-commit mode must

Connection Attribute	Values Supported	Values Not Supported
		closed with `SQLEndTran();` be calling these functions.
SQL_ATTR_AUTO_IPD	SQL_TRUE SQL_FALSE	This is a read-only attribute based on the database supp of extended statement info.
SQL_ATTR_LOGIN_TIMEOUT	An integer value	–
SQL_ATTR_METADATA_ID	SQL_FALSE SQL_TRUE	–
SQL_ATTR_QUIET_MODE	A 32-bit window handle	–
SQL_ATTR_TRANSLATE_LIB	A character string value	–
SQL_ATTR_TRANSLATE_OPTION	A 32-bit integer value	–
SQL_ATTR_TXN_ISOLATION	SQL_TXN_READ_UNCOMMITTED SQL_TXN_SERIALIZABLE	SQL_TXN_READ_COMMITE SQL_TXN_REPEATABLE_R

In addition to the connection attributes given in the previous table, applications can use the following connection attributes supported by the Driver Manager:

- SQL_ATTR_ODBC_CURSORS
- SQL_ATTR_TRACE
- SQL_ATTR_TRACEFILE

The following table lists the statement attributes that are supported in ODBC Driver for Teradata.

Statement Attribute	Values Supported	Values Not Supported
SQL_ATTR_APP_PARAM_DESC	A Descriptor Handle	–
SQL_ATTR_APP_ROW_DESC	A Descriptor Handle	–
SQL_ATTR_ASYNC_ENABLE	SQL_ASYNC_ENABLE_OFF SQL_ASYNC_ENABLE_ON	–
SQL_ATTR_CONCURRENCY	SQL_CONCUR_READ_ONLY	SQL_CONCUR_LOCK SQL_CONCUR_ROWVER SQL_CONCUR_VALUES

Statement Attribute	Values Supported	Values Not Supported
SQL_ATTR_CURSOR_TYPE	SQL_CURSOR_FORWARD_ONLY	SQL_CURSOR_STATIC SQL_CURSOR_KEYSET_DRIV SQL_CURSOR_DYNAMIC
SQL_ATTR_ENABLE_AUTO_IPD	SQL_TRUE SQL_FALSE	–
SQL_ATTR_IMP_PARAM_DESC	A Descriptor Handle	–
SQL_ATTR_IMP_ROW_DESC	A Descriptor Handle	–
SQL_ATTR_KEYSET_SIZE	Zero	Other than zero
SQL_ATTR_MAX_LENGTH	Zero	Other than zero
SQL_ATTR_MAX_ROWS	An integer value	–
SQL_ATTR_METADATA_ID	SQL_FALSE SQL_TRUE	–
SQL_ATTR_NOSCAN	SQL_NOSCAN_OFF SQL_NOSCAN_ON	–
SQL_ATTR_PARAM_BIND_OFFSET_PTR	A pointer to an integer value	–
SQL_ATTR_PARAM_BIND_TYPE	SQL_PARAM_BIND_BY_COLUMN or an integer value specifying the row size (for row-wise binding)	–
SQL_ATTR_PARAM_OPERATION_PTR	An array, in which the status values are stored or NULL	–
SQL_ATTR_PARAM_STATUS_PTR	An array, in which the status values are stored or NULL	–
SQL_ATTR_PARAMS_PROCESSED_PTR	A pointer to an integer value or NULL	–
SQL_ATTR_PARAMSET_SIZE	An integer value	–
SQL_ATTR_QUERY_TIMEOUT	An integer value	–
SQL_ATTR_RETRIEVE_DATA	SQL_RD_ON	SQL_RD_OFF
SQL_ATTR_ROW_ARRAY_SIZE	An integer value	–
SQL_ATTR_ROW_BIND_OFFSET_PTR	A pointer to an integer value	–

Statement Attribute	Values Supported	Values Not Supported
SQL_ATTR_ROW_BIND_TYPE	SQL_BIND_BY_COLUMN or an integer value specifying the row size (for row-wise binding)	–
SQL_ATTR_ROW_NUMBER	An integer value	–
SQL_ATTR_ROW_STATUS_PTR	An array, in which the status values are stored or NULL	–
SQL_ATTR_ROWS_FETCHED_PTR	A pointer to an integer value or NULL	–
SQL_ATTR_SIMULATE_CURSOR	SQL_SC_NON_UNIQUE	SQL_SC_TRY_UNIQUE SQL_SC_UNIQUE
SQL_ATTR_TRUSTED_SQL	SQLULEN value of either SQL_TRUE or SQL_FALSE	–
SQL_ATTR_USE_BOOKMARKS	SQL_UB_OFF	SQL_UB_VARIABLE SQL_UB_FIXED

Supported Descriptor Fields

The descriptor fields consist of header fields and record fields. The list of supported descriptor fields is provided in the following tables. It is important to understand that descriptors are new to ODBC 3.x and that there are no restrictions on the values for the supported descriptor fields.

The following table lists Header Fields.

Header Field
SQL_DESC_ALLOC_TYPE
SQL_DESC_ARRAY_SIZE
SQL_DESC_ARRAY_STATUS_PTR
SQL_DESC_BIND_OFFSET_PTR
SQL_DESC_BIND_TYPE
SQL_DESC_COUNT
SQL_DESC_ROWS_PROCESSED_PTR

The following table lists Record Fields.

Record Field
SQL_DESC_AUTO_UNIQUE_VALUE
SQL_DESC_BASE_COLUMN_NAME
SQL_DESC_BASE_TABLE_NAME
SQL_DESC_CASE_SENSITIVE
SQL_DESC_CONCISE_TYPE
SQL_DESC_DATA_PTR
SQL_DESC_DATETIME_INTERVAL_CODE
SQL_DESC_DATETIME_INTERVAL_PRECISION
SQL_DESC_DISPLAY_SIZE
SQL_DESC_FIXED_PREC_SCALE
SQL_DESC_INDICATOR_PTR
SQL_DESC_LENGTH
SQL_DESC_LITERAL_PREFIX
SQL_DESC_LITERAL_SUFFIX
SQL_DESC_LOCAL_TYPE_NAME
SQL_DESC_NAME
SQL_DESC_NULLABLE
SQL_DESC_OCTET_LENGTH
SQL_DESC_OCTET_LENGTH_PTR
SQL_DESC_PARAMETER_TYPE
SQL_DESC_PRECISION
SQL_DESC_ROWVER
SQL_DESC_SCALE
SQL_DESC_SEARCHABLE

Record Field
SQL_DESC_TYPE
SQL_DESC_TYPE_NAME
SQL_DESC_UNNAMED
SQL_DESC_UNSIGNED
SQL_DESC_UPDATABLE

ODBC SQL Grammar

Core Compliance

The following table lists the following core-compliant grammar:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Set Functions

DDL	DML	Set Functions
CREATE TABLE	SELECT	AVG
DROP TABLE	INSERT	MIN
ALTER TABLE	UPDATE	MAX
CREATE INDEX	DELETE	COUNT
DROP INDEX	CALL	SUM
CREATE VIEW	UPSERT	
DROP VIEW		
CREATE PROCEDURE		
DROP PROCEDURE		
REPLACE PROCEDURE		
GRANT		
REVOKE		

Subqueries

The following table lists the supported SQL grammar extensions:

Extensions	
String	Functions
Numeric	Data Type Conversions
Time/Date	Procedures (Stored and Macros)
System Scalar	

ODBC Escape Clauses

The following table lists both standard and extended escape forms of ODBC escape clauses:

Escape Clauses	
date literals	macro calls
time literals	date parameters
timestamp literals	

CREATE and DROP INDEX

CREATE INDEX is supported in ODBC syntax as well as in Teradata syntax. DROP INDEX is supported in Teradata syntax as well as ODBC syntax for locally created index names. For example, it is possible to drop an index using the index name in which it was created, provided you are working on the same client system where it was created. However, you cannot drop an index using the Teradata internal named index convention (for example, *`TableName004`*).

Using this feature is not recommended, because it is deprecated. For details, see“ODBC-Style Named Indexes (deprecated in 15.00)”.

ODBC Named Indexes

Note:

Using this feature is not recommended, because it is deprecated.

ODBC index-name grammar is supported on a per workstation basis for the following:

```
CREATE INDEX
DROP INDEX
DROP TABLE
SQLStatistics()
```

For example,

```
CREATE INDEX myindex ON mytable (column1, column2)
```

ODBC grammar ASC and DESC column-identifier clauses for CREATE INDEX are ignored since Teradata does not support this type of indexing. It is possible to DROP an index using the index-name which was specified when it was created, provided you are working on the same PC and the IndexName entry in the Registry file has not been reused.

The specified index-name need not exist on the PC; it is not relevant to Teradata because an index is created for the specified table using the standard index grammar.

For example,

```
CREATE INDEX (columnname(s)) ON tablename
```

If the named index has been deleted from the Registry, or if a user is on a different PC, the index needs to be DROPped using the Teradata grammar by specifying the tablename and columns that make up the index. For example,

```
DROP INDEX (columnname(s)) ON tablename
```

You cannot DROP a Teradata named index (for example, ***`TableName004`***).

Scalar Functions

The Teradata Database supports a number of ODBC scalar functions, with additional scalar functions being added incrementally in new database releases. Refer to the *Teradata Database Functions, Operators, Expressions and Predicates User Guide* (B035-1445) for the list of the currently supported scalar functions by the Teradata Database.

When a client queries a scalar function using a function escape sequence (using the ‘`{fn scalar-function}`’ syntax, for example, ‘`SELECT {fn MOD(x, y) }`’), the ODBC Driver will determine whether the connected database supports the scalar function. If so, the ODBC Driver will send the query directly to the database (e.g., “`SELECT MOD(x, y)`”); if not, the driver will transform the user query to a query that the connected database supports (e.g., “`SELECT((x) MOD (y))`”).

A client may send a scalar function query without the function escape sequence, for example, “`SELECT MOD(x, y)`”. In this case, the ODBC Driver will send the client query to the database as is without performing any kind of checks or transformation. This query will succeed if the connected database supports the scalar function, it will fail otherwise. (For the MOD function, it is supported by the 16.20 Feature Update 1 Database, but not the 15.10 Database, for example.)

The following table details a few examples exhibiting the above described behavior.

Statement Text	SQL Generated by the ODBC Driver (i.e., SQLNativeSql)	Teradata Database 16.20 Feature Update 1	Teradata Database 15.10
`SELECT {fn BIT_LENGTH('Teradata')}`	`SELECT(Octet_Length('Teradata')*8)`	Succeeds	Succeeds
`SELECT BIT_LENGTH('Teradata')`	`SELECT BIT_LENGTH('Teradata')`	Succeeds	Fails
`SELECT {fn DAYOFYEAR('2018-02-24') }`	`SELECT (((('2018-02-24')(DATE))- (((('2018-02-24')(DATE))/ 10000*10000+0101(DATE)))) + 1(TITLE 'DayOfYear()'))`	Succeeds	Succeeds
`SELECT DAYOFYEAR('2018-02-24')`	`SELECT DAYOFYEAR('2018-02-24')`	Fails	Fails

The following functions were updated in the Teradata Database to support the ODBC Driver-accepted syntax.

Function Call	Change	Teradata Database Version Implemented
CURRENT_DATE()	Can be called with or without parenthesis.	16.20 Feature Update 1
LENGTH(*s*)	Can also support numeric expressions.	16.20 Feature Update 1
LTRIM(*s*)	Can also operate on numeric expressions.	16.20 Feature Update 1
RTRIM(*s*)	Can also operate on numeric expressions.	16.20 Feature Update 1

Function Call	**Change**	**Teradata Database Version Implemented**
SUBSTRING*`(s,n1[,n2])`*	New syntax in Teradata mode	16.20 Feature Update 1

The following functions are now supported in Teradata Database version 16.20 Feature Update 1.

Function Call	**Purpose**
BIT_LENGTH(*s*)	Returns the length in bits of the specified string.
CONCAT(*s1*,*s2*[,...*sn*])	Concatenates two or more parameter values into a single string.
CURDATE()	Returns the current date.
CURTIME()	Returns the current local time.
DAYOFMONTH(*e*)	Returns the number of days from the beginning of the month to the specified date.
DAYOFWEEK(*e*)	Returns the day of the week on which the specific date falls.
HOUR(*e*)	Returns the hour field of the specified time expression.
LOCATE(*s1*,*s2*[,*p*])	Returns the position in ***s2*** where the substring ***s1*** starts.
MINUTE(*e*)	Returns the minute field of the specified time expression.
MOD(*x1*, *x2*)	Returns the remainder from a division.
MONTH(*e*)	Returns the number of months from the beginning of the year to the specified date.
NOW()	Returns the current date and time.
SECOND(*e*)	Returns the second field of the specified time expression.
UCASE(*s*)	Returns an equal string, which all lowercase characters convert to uppercase.
WEEK(*e*)	Returns the number of weeks from the beginning of the year to the specified date.
YEAR(*e*)	Returns the year of the specified date.

ANSI SQL 1992 Syntax

This section describes how the entry level ANSI SQL 1992 syntax is supported by ODBC Driver for Teradata, including DATE and TIME literals and the functions listed.

Syntax	Notes
EXTRACT()	None
TRIM()	Remove only spaces and Os The ANSI syntax for TRIM is very different from ODBC syntax for Teradata syntax. Migration: The TRIM function works differently in Teradata and ANSI modes. In Teradata mode, TRIM only removes trailing blanks. In ANSI, TRIM removes leading as well as trailing blanks.
SUBSTRING()	The ANSI syntax for SUBSTRING is very different from ODBC syntax for Teradata syntax
CAST()	None

ANSI SQL with No Equivalent Teradata Simulation

ANSI SQL abilities that have no equivalent in Teradata Database are not simulated. Use of these features will cause errors. Examples are as follows:

- UPDATE and DELETE where CURRENT OF cursor (cursor library can be used to provide this functionality)
- Outer join extensions to SQL
- Cursor scrolling is limited to the forward direction only

Interval Values

Both TIMESTAMPADD( ) and TIMESTAMPDIFF( ) support the interval values listed in the following table.

Interval Values	
SQL_TSI_FRAC_SECOND	SQL_TSI_DAY
SQL_TSI_SECOND	SQL_TSI_MONTH

Interval Values	
SQL_TSI_MINUTE	SQL_TSI_YEAR
SQL_TSI_HOUR	

Data Types for ANSI Compliance

The NUMERIC, REAL, and DOUBLE PRECISION data types are actually synonyms for existing Teradata data types: NUMERIC is mapped to DECIMAL, REAL and DOUBLE PRECISION are mapped to FLOAT. As a result, Teradata does not return column type values of NUMERIC, REAL, or DOUBLE PRECISION. Instead, it returns DECIMAL, FLOAT, and FLOAT respectively. SQLGetTypeInfo (SQL_ALL_TYPES) does not return these new ANSI types.

Note:

Refer to Teradata Database documentation for a complete description of ANSI features and migration issues.

ODBC Driver for Teradata supports the standard Teradata data types, in addition to the ODBC data types listed in the following table:

ANSI Data Types	
TINYINT	INTERVAL_YEAR
NUMERIC	INTERVAL_MONTH
REAL	INTERVAL_DAY
DOUBLE PRECISION	INTERVAL_HOUR
BIT	INTERVAL_MINUTE
BINARY	INTERVAL_SECOND
VARBINARY	INTERVAL_HOUR_TO_MINUTE
TIME	INTERVAL_DAY_TO_SECOND
TIMESTAMP	INTERVAL_HOUR_TO_MINUTE
DATE	INTERVAL_HOUR_TO_SECOND
INTERVAL	INTERVAL_MINUTE_TO_SECOND

The tables that follow list data type changes occurring between ODBC 2.x and ODBC 3.x. These changes affect the data type names only. Their functionality remains as before.

The following table lists SQL Data Type name changes.

ODBC 2.5	ODBC 3.51
SQL_DATE	SQL_TYPE_DATE
SQL_TIME	SQL_TYPE_TIME
SQL_TIMESTAMP	SQL_TYPE_TIMESTAMP

The following table lists SQL C-Type identifier changes.

ODBC 2.5	ODBC 3.51
SQL_C_DATE	SQL_C_TYPE_DATE
SQL_C_TIME	SQL_C_TYPE_TIME
SQL_C_TIMESTAMP	SQL_C_TYPE_TIMESTAMP

The DATE data types automatically get a default format of ***`YYYY-MM-DD`*** to match the ODBC and ANSI specs. The following table lists columns renamed for ODBC 3.x.

Note:

Columns names won't change. The following columns have been renamed for ODBC 3.x. The column name changes do not affect backward compatibility, because applications bind by column number.

ODBC 2.0 column	ODBC 3.*x* column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS
RADIX	NUM_PREC_RADIX

The following table lists the columns added to the result set returned by SQLColumns in ODBC driver:

Columns Added	
CHAR_OCTET_LENGTH	ORDINAL_POSITION
COLUMN_DEF	SQL_DATA_TYPE
IS_NULLABLE	SQL_DATETIME_SUB

ODBC Connection Functions and Dialog

The ODBC Driver for Teradata API defines three connection functions for establishing a connection to a database.

- SQLConnect()
- SQLDriverConnect()
- SQLBrowseConnect()

SQLConnect

SQLConnect assumes that a data source name, username, and password are sufficient to connect to a data source and that all other connection information can be stored on the system. This information is stored in the `DSN Settings` file in the registry on Windows in the user's `odbc.ini` file.

Syntax:

```
SQLRETURN SQLConnect(
SQLHDBC ConnectionHandle,
SQLCHAR * ServerName,
SQLSMALLINT NameLength1,
SQLCHAR * UserName,
SQLSMALLINT NameLength2,
SQLCHAR * Authentication,
SQLSMALLINT NameLength3);
```

SQLConnect establishes a connection to a driver and a data source. The connection handle references storage of all information about the connection to the data source, including status, transaction state, and error information.

SQLConnect is the simplest connection function. It requires a ServerName (data source name) and accepts an optional UserName and Authentication (typically a password). The UserName and Authentication parameters are passed unmodified to the driver.

SQLConnect works well for applications that hard-code a data source name and do not require a UserName or Authentication.

It also works well for applications that want to control their own look and feel or that have no user interface. Such applications can build a list of data sources using SQLDataSources, prompt the user for data source, UserName, and Authentication, and then call SQLConnect.

SQLConnect establishes an authenticated connection using the authentication mechanism selected. SQLConnect returns SQL_ERROR and a new error message if no authentication mechanism can be selected.

If TeraSSO allows fully qualified usernames then the username might contain a domain or realm, for example: ***george@linedata***.

Teradata Database Connect

If the username and password boxes are filled, the ODBC driver converts them to a simplified logon format which is passed to TeraGSS as mech data.

Teradata Database Connect Set Up

Field, Check Box, or Button	Description
DBC Name or Address	Specify the COP or AMP for your first connection. This is the list of names and IP addresses that was configured in the **Teradata Server Info** group box of the **OBDC Driver Setup for Teradata Database** dialog box.
Authentication group box	
Parameter	Type in the parameters required for the desired security checking mechanism. With simplified login support, it is not necessary to fill in the **Parameter** field for most logins. Sometimes it might be necessary to supply additional information such as ***Profile=MyProfile*** to log on to the database. In such cases, the additional information is contained in the **Parameter** field. The Parameter value can be specified when the DSN is created. Like a password, the value will be masked. To change the value of an existing parameter or to enter a new value, click on the **Change** button next to the **Parameter** field. A new **Parameter** dialog box opens, which allows the specification of a new Parameter value. If a Parameter value already exists, it will be displayed in clear text in the dialog box and can be edited or replaced. When changes are complete, click **OK**.
Username	Default = Cleared

Field, Check Box, or Button	Description
	Shows the default username that was specified during the data source(s) configuration of the driver. The default value can be overridden here. If required, the user is prompted for additional information. If the **Username** field is filled in, the ODBC Driver converts it to a simplified logon format which is passed to the Teradata security library (TeraGSS).
Password	Sends the password (if any) that was configured for this data source (unless a different password is supplied). If required, the user is prompted for additional information. If the **Password** field is filled in, the ODBC Driver converts it to a simplified logon format which is passed to the Teradata security library (TeraGSS).
Optional group box	
Default Database	Indicate the default data space that has been allocated to the user (when supplied). If provided, the user does not have to supply the fully qualified table name with each request. If not supplied, the Teradata system automatically assigns the user to the default data space, which might not have the desired permissions and tables.
Account String	Identifies the user (if the Teradata system has account information being gathered).
Buttons	
Change	To change the value of an existing parameter or to enter a new value, click on the **Change** button next to the **Parameter** field. A new **Parameter** dialog box opens, which allows the specification of a new Parameter value. If a Parameter value already exists, it will be displayed in clear text in the **Parameter** dialog box and can be edited or replaced. When changes are made, click **OK**.
OK	Click to enable the driver to use any changes that have been made in the dialog box. Changes are temporary and are not stored, with the exception of the username, which is stored as the last user that connected to the data source.
Cancel	Click to cancel any changes made to the dialog box and abort the current driver and data source selection.

Field, Check Box, or Button	Description
Help?	Click to obtain detailed help about this dialog box.

SQLDriverConnect

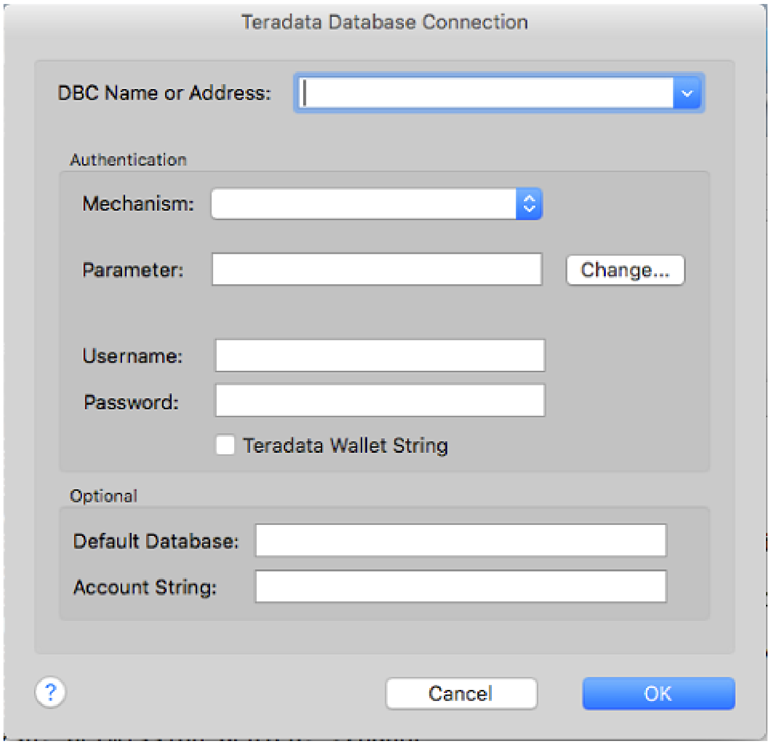
SQLDriverConnect allows the ODBC driver to define an arbitrary amount of connection information in the form of keyword-value pairs in the connection string. For example, a custom program that always uses the XYZ Corp data source might prompt the user for names and passwords and build the following set of keyword-value pairs, or connection string, to pass to SQLDriverConnect:

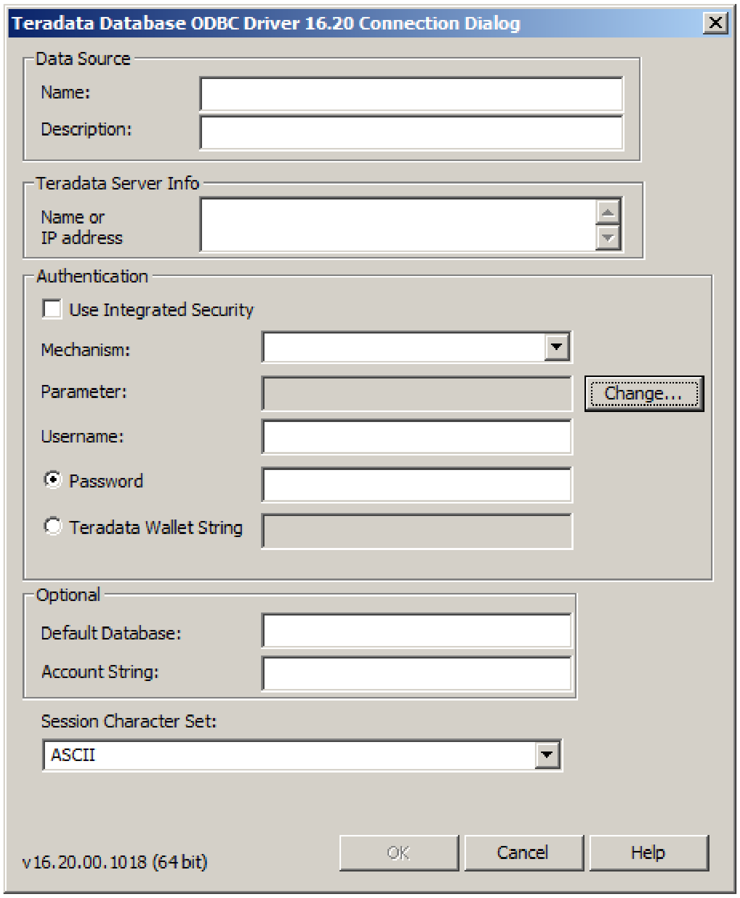
```
DSN=XYZ Corp;UID=Gomez;PWD=Sesame;
```

The SQLDriverConnect function is allowed to prompt the user for required logon information unless called with the SQL_DRIVER_NOPROMPT option.

Sometimes it might be necessary to supply additional information such as *`Profile=MyProfile`* to log on to the database. In this case, the additional information is contained in the **Parameter** field. The Parameter value was specified when the DSN was created. Like a password, the value will be masked.

The following example illustrates this.



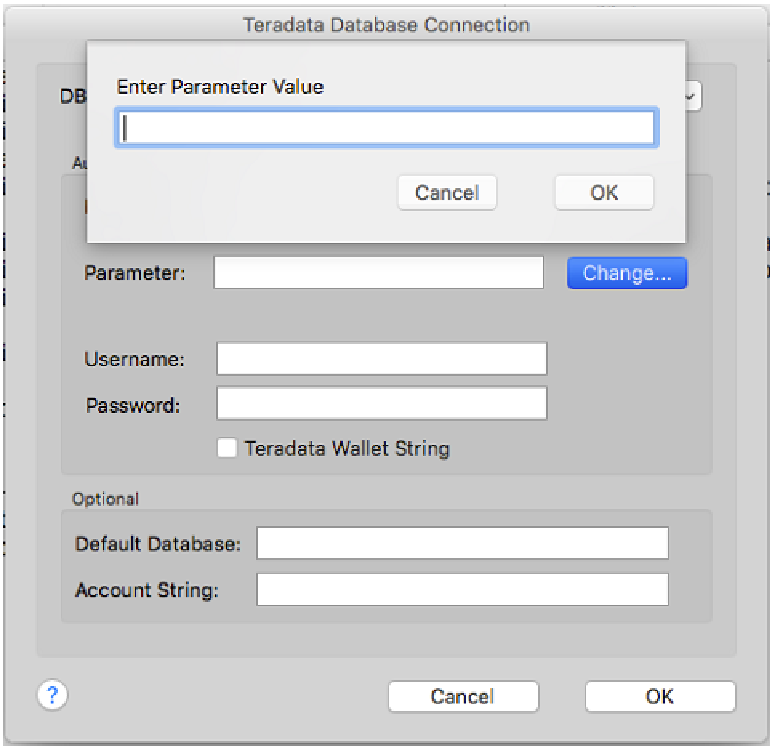


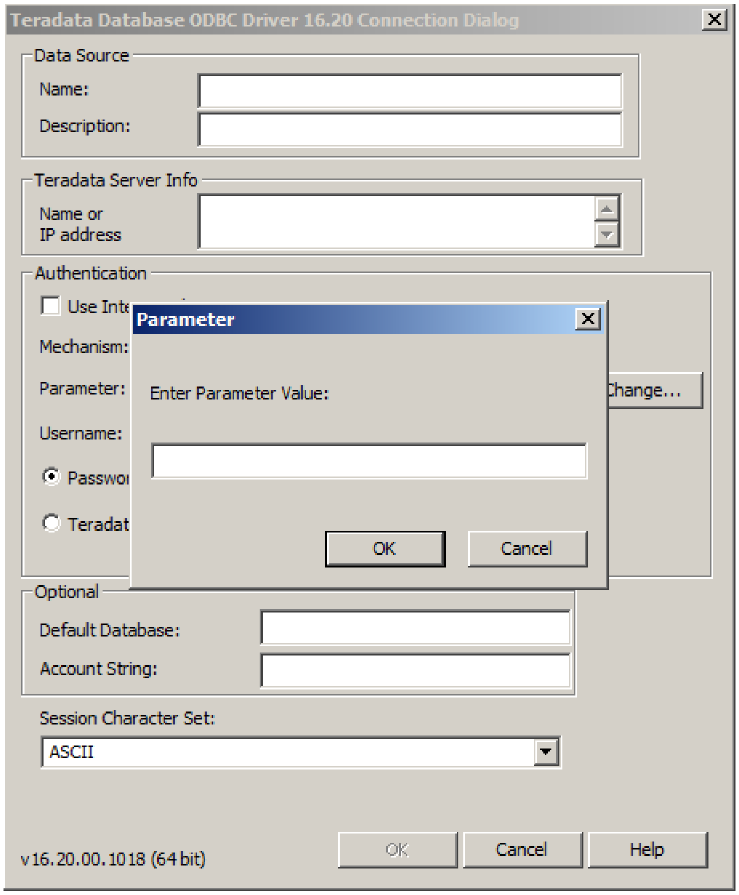
● Password

Session Character Set: ASCII

To change the value of an existing parameter or to enter a new value, the user clicks on the **Change** button, which is located on the Teradata Database **Connect** dialog box next to the **Parameter** field. A new **Parameter** dialog box displays, allowing the user to specify a new Parameter value. If a value already exists, it will be displayed in clear text and the user can edit or replace it. When changes are made, click the **OK** button.

The following example illustrates this.


Teradata Database Connection
Enter Parameter Value
Cancel
OK
DB
Parameter:
Change...
Username:
Password:
☐ Teradata Wallet String
Optional
Default Database:
Account String:
?
Cancel
OK


Teradata Database ODBC Driver 16.20 Connection Dialog
Data Source
Name:
Description:
Teradata Server Info
Name or
IP address
Authentication
☐ Use Inte
Mechanism:
Parameter:
Username:
● Passwo
○ Teradat
Change...
Parameter
Enter Parameter Value:
OK
Cancel
Optional
Default Database:
Account String:
Session Character Set:
ASCII
v16.20.00.1018 (64 bit)
OK
Cancel
Help

Syntax

```
SQLRETURN SQLDriverConnect(
   SQLHDBC      ConnectionHandle,
   SQLHWND      WindowHandle,
   SQLCHAR *    InConnectionString,
   SQLSMALLINT  StringLength1,
   SQLCHAR *    OutConnectionString,
   SQLSMALLINT  BufferLength,
   SQLSMALLINT *        StringLength2Ptr,
   SQLUSMALLINT         DriverCompletion);
```

SQLDriverConnect supports data sources that require more connection information than the three arguments in SQLConnect, dialog boxes to prompt the user for all connection information, and data sources that are not defined in the system information.

After a connection is established, SQLDriverConnect returns the completed connection string. The application can use this string for subsequent connection requests.

See the connection string keywords and Connection dialog.

SQLDriverConnect returns SQL_ERROR and an error message if no authentication mechanism can be selected.

Simplified Logon Without a Connection Dialog Box

The preceding dialog boxes might not be displayed if the ODBC Driver has sufficient information to attempt logging on to the database. The behavior is controlled by arguments supplied to SQLDriverConnect.

Configuring a DSN

The **DSN Setup** dialog box is similar to the **Teradata Database Connect** dialog box, and works in a similar fashion.

Simplified Logons with the Default Mechanism

Simplified logons will not work when the default mechanism is anything other than TD2 and the **Mechanism** text box is blank.

SQLBrowseConnect

SQLBrowseConnect is similar to SQLDriverConnect, but allows the caller to provide the connection information in iterations. SQLBrowseConnect does not allow the driver to prompt for additional logon information. Instead, the driver returns a string indicating what information is required and what information might be optionally specified.

If, for example, the user specifies:

`DRIVER=Teradata`

then the ODBC driver might return the following string:

```
"DBCNAME:Teradata System=?;UID:Teradata User Id=?;PWD:Teradata
Password=?;*USEINTEGRATEDSECURITY:Enable SSO =
?;*DATABASE:DefaultDatabase=?;*ACCOUNT:Account=?;"
```

where the * indicates an optional field, and the “?” indicates a missing value.

Syntax

```
SQLRETURN SQLBrowseConnect(
   SQLHDBC                     ConnectionHandle,
   SQLCHAR *         InConnectionString,
   SQLSMALLINT           StringLength1,
   SQLCHAR *         OutConnectionString,
   SQLSMALLINT     BufferLength,
   SQLSMALLINT *         StringLength2Ptr);
```

SQLBrowseConnect supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source.

Each call to SQLBrowseConnect returns successive levels of attributes and attribute values and the OutConnectionString contains the attributes needed for the next level. When all levels have been enumerated, a connection to the data source is completed and a complete connection string is returned by SQLBrowseConnect.

It uses the connection string keywords related to security and logon.

SQLBrowseConnect establishes an authenticated connection using the authentication mechanism selected.

SQLBrowseConnect returns the following in the OutConnectionString for the connection string attribute keywords:

`*AUTHENTICATION:Authentication Mechanism={<MechanismList>}`

where the braces are literal (they are returned by the driver) and the <MechanismList> is a comma-separated list of colon-separated pairs of mechanism names and mechanism labels. The asterisk indicates that the AUTHENTICATION attribute is optional.

The mechanism name labels are user-friendly names intended by ODBC to be shown as labels in a dialog box. If such names are available from TeraGSS then they will be used; otherwise, the mechanism names will be used. Default mechanisms will not be indicated.

The mechanisms enumerated in the <MechanismList> are supported on the client if there is not enough information in the connection string to establish a connection to the gateway. Otherwise, the list will contain the authentication mechanisms supported by both the client and gateway.

An example:

```
*AUTHENTICATION:Authentication Mechanism={TD2:Teradata 2, KRB5:Kerberos}
*AUTHENTICATIONSTRING:Authentication String=?
```

where the "?" indicates that a value is required.

SQLBrowseConnect returns SQL_ERROR and an error message if no authentication mechanism can be selected.

Keywords for SQLDriverConnect() and SQLBrowseConnect()

The keywords listed in the following table can be specified in the connect string input of SQLDriverConnect() or in the connect string input of SQLBrowseConnect() for ODBC Driver for Teradata.

Keywords are not case sensitive. Optional keywords are designated by an *.

Note:

An attribute-key value enclosed in { } cannot have the } character.

Keyword	Explanation
*ACCOUNTSTRING=<***Account Name***> Or *ACCOUNT=<***Account Name***>	See the "AccountString=<***account***>" option in the the table titled *Options Configurable in the odbc.ini File* in Teradata DSN Options.
*CHARACTERSET=<***Character set***> Or *CHARSET=<***Character set***>	Default = ASCII See the "CharacterSet=<***charset name***>" option in Teradata DSN Options.

DATASOURCEDNSENTRIES=<value>
Or
DSDNSENTRIES=<value>
	Default = unassigned
	See **Data Source DNS Entries** in Teradata ODBC Driver Advanced Options.

Keyword	Explanation
*DATETIMEFORMAT=[*A*|*I*]*AA*	See the "DateTimeFormat=[*A*|*I*]*AA* option in Teradata DSN Options. The recommended settings are either the **AAA** (default), or the **IAA** (optional) formats. Because the Integer data type has been deprecated for the TIME format, it is not recommended. For information, see Integer Time. The last character that represents TIMESTAMP is always ANSI.
*DEFAULTDATABASE=<***database name***> Or *DATABASE=<***database name***>	See the "Database=<***database name***>" option in Data Source Specification Section.
*DBCNAME=123.45.67.89	See the "DBCName=<IP-addr-or-alias>" option in Data Source Specification Section. This setting is made in the **Teradata Server Info** section in the **ODBC Driver Setup** dialog box. For details, see ODBC Driver Setup Parameters. Only one name or IP address is valid. When a name is specified, ODBC Driver for Teradata automatically detects associated COP entries. For example, if the name contains a COPx suffix. For more information, see Cop Discovery.
*DISPKANJICONVERRS=[*Y* | *N*]	This DSN option is ignored. Invalid characters result in substitute characters being used.
*DOMAIN=<***Domain Name***>	(Windows only) When the Domain Name is provided for Third Party Sign-On, you should provide the domain name along with the username and password. If a domain name is not provided, then the local domain is assumed.

Keyword	Explanation
*DONTUSEHELPDATABASE=[*Y* | *N*] Or *DONTUSEHELPDB=[*Y* | *N*]	Default = No See the "DontUseHelpDatabase=[*Yes* | *No*]" option in Teradata DSN Options.
*DRIVER=<ODBC Driver for Teradata>	See the "Driver=<driver-path>" option in Data Source Specification Section.
DSN=<*DataSourceName*>	See the "data-source-name=<driver>" option in ODBC Data Sources Section.
*ENABLEEXTENDEDSTMTINFO=[Y | N]	Default = Y See the "EnableExtendedStmtInfo=[*Yes* | *No*]" option in Teradata DSN Options.
*ENABLEREADAHEAD=[Y | N]	Default = Y See the "EnableReadAhead=[*Yes* | *No*]" option Teradata DSN Options.
*ENABLERECONNECT=[*Y* | *N*]	Default = No See the "EnableReconnect=[*Yes* | *No*]" option in Teradata DSN Options.
*IGNORESEARCHPATTERN=[*Y* | *N*] Or *IGNORESEARCHPAT=[*Y* | *N*]	Default = No See the "IgnoreODBCSearchPattern=[*Yes* | *No*]" option in Teradata DSN Options.
*LOGINTIMEOUT=<*Time out Value*>	Default = 20 See Teradata DSN Options.
*MAXRESPSIZE=<*Resp Buffer size*>	Default = 65536 (64K) See Teradata DSN Options.

MECHANISMKEY=<*Value*>
Or
AUTHENTICATIONPARAMETER=<*Value*>
	Value=*string*
	A string of characters opaque to the driver, and regarded as a parameter to the authentication mechanism. Passed to the Teradata authentication software called to set the authentication mechanism.

Keyword	Explanation
	For instructions on configuring this parameter, see the MechanismKey=<***Value***> description in Teradata DSN Options. For more information, see TDGSS Support for UTF16. You can use a Teradata Wallet reference string as the value for this keyword (or part of the value) by specifying the**`$tdwallet()`** token. For example: `AUTHENTICATIONPARAMETER=` **`$tdwallet(WalletRefString)`**; For more information, see Teradata Wallet.

MECHANISMNAME=<***MechanismName***>
Or
AUTHENTICATION=<***MechanismName***>
	Identifies the authentication mechanism used for connections to the data source.
	The default is determined by a configuration option set in an XML file by the TeraGSS program, tdgssconfigure.
	For instructions on configuring the authentication mechanism, see the MechanismName=<***MechanismName***> description in Teradata DSN Options.
	For more information, see Authentication Mechanisms.

*MSACCESSINTEROPMODE=[*Y* | *N*] Or *USEDATEDATAFORTIMESTAMPPARAMS=[*Y* | *N*]	Default = No (Windows Only)
*NOSCAN=[*Y* | *N*]	Default = No

Keyword	Explanation
	See the "NoScan=[*Yes* | *No*]" option in Teradata DSN Options.
*NTERRLOG=[*Y* | *N*]	Default = No (Windows only) When **Yes** is selected, this option enables the user to log errors returned by ODBC Driver for Teradata in the Event Log. When **No** is selected, no errors are logged in the Event Log.
*PASSWORD=<*value*> Or *PWD=<*value*>	See the "Password=<password>" option in Data Source Specification Section. You can use a Teradata Wallet reference string as the value for this keyword (or part of the value) by specifying the `$tdwallet()` token. For example: `PWD=$tdwallet(WalletRefString)`; For more information, see Teradata Wallet.
*PRINTOPTION= [*P* | *N*]	Default = No See the "PrintOption=[*O* | *P*]" option in Teradata DSN Options.
*PWD2=<*value*>	A secondary password used with the default username when logging onto a Teradata server. (See the *Password* option in Data Source Specification Section.) If PWD2 is enabled, and the primary password has expired, the driver will automatically use this secondary password. If PWD2 is not enabled, and RunInQuietMode=**No**, then the driver displays a prompt for a new password. For more information, see the

Keyword	Explanation
	RunInQuietMode=[Yes / No] option in Teradata DSN Options. When SQLDriverConnect or SQLBrowseConnect is called and PWD has expired, connect keywords recognized for ConnectStringInput are extended by PWD2=new_password_value. When an attempt is made to establish a Teradata session, a logon request is issued using PWD. If a restricted logon is returned because the password has expired, a MODIFY is issued using PWD2. The ConnectStringOutput is then updated to reflect the password used to establish the session. Applications are responsible for checking the password used for the login and using it in their own login processes. • An expired password error will return when calling SQLConnect, because PWD2 does not support the ConnectStringInput string. • PWD2 might not work in applications that normally cache connection information. • PWD2 does not work for passwords stored in the Registry.

Keyword	Explanation
	Note: For PWD2 to work, an application must be aware of how password expiration is handled by the Teradata session connection process and must use that knowledge when sending connection requests. Off-the-shelf ODBC applications or 3rd party tools cannot use PWD2 because their connection code does not know how to use the secondary password. PWD2 only works in applications that have been customized to use it.
*RECONNECTWAIT=<*wait time in min*>	Default = 10 See the "ReconnectWait=<integer>" option in Teradata DSN Options.
RETRYONEINTR=[*Y* | *N*]	Default = Y This option allows the user to control whether ODBC Driver for Teradata should retry the socket system calls on an EINTR or return an SQL_ERROR. The socket system calls affected are: • connect() • select() • recv() • send() See the "RETRYONEINTR=[*Y* | *N*]" option in Teradata DSN Options.
*RETURNGENERATEDKEYS=<value>	Default = N See the RETURNGENERATEDKEYS=<value> description in Teradata DSN Options.
*SESSIONMODE=[*ANSI*|*Teradata*]	The default value is determined by the database based on the option used in the Teradata Database CREATE or MODIFY USER statement.

Keyword	Explanation
	See the “SessionMode=[*TeradataANSI*]” option in Teradata DSN Options. For Windows configuration, see the **Session Mode** menu description in Teradata ODBC Driver Options.
*SPLOPTION=[*Y* | *N*]	Default = No See the “SplOption=[*Y* | *N*]” option in Teradata DSN Options.
*TABLEQUALIFIER=[*Y* | *N*]	Default = No When this option is Yes, NULL values are passed for the Table Qualifier parameters in the Catalog API functions. When this option is No, NULL values are *not* passed for the Table Qualifier parameters in the Catalog API functions.
*TCPNODELAY=[*Y* | *N*]	Default = Yes See the “TCPNoDelay=[*Yes* | *No*]” option in Teradata DSN Options.
*TDMSTPORTNUMBER=<*port number*>	Default = 1025 See the “TDMSTPortNumber=<integer>” option in Teradata DSN Options.
*USECOLUMNNAMES=[*Y* | *N*] Or *DONTUSETITLES=[*Y* | *N*]	Default = Yes See the “DontUseTitles=[*Yes* | *No*]” option in Teradata DSN Options.

USEDATAENCRYPTION=[*Y* |*N*]
Or
DATAENCRYPTION=[*Y* |*N*]
	Default = No
	When DATAENCRYPTION is Yes, ODBC Driver for Teradata encrypts data, and thus the Teradata gateway and ODBC Driver for Teradata communicate with each other in encrypted manner.

Keyword	Explanation
	When DATAENCRYPTION is No, ODBC Driver for Teradata does not encrypt data, except for logon information.

USEREGIONALSETTINGS=[*Y* | *N*]
Or
REGIONALDECIMALSYMBOL=[*Y* | *N*]
	Default = Yes
	(Windows and Apple OS X) See the "Use Regional Settings for Decimal Symbol" option in Teradata ODBC Driver Options.

*USERNAME=<*value*> Or *UID=<*value*>	See the "Username=<name>" option in Data Source Specification Section.
*USEXVIEWS=[*Y* | *N*]	Default = No See the "UseXViews=[Yes|No]" option in Teradata DSN Options.

SQLGetInfo – Get User Name

Syntax

```
SQLRETURN SQLGetInfo(
  SQLHDBC       ConnectionHandle,
  SQLUSMALLINT  InfoType,
  SQLPOINTER    InfoValuePtr,
  SQLSMALLINT   BufferLength,
  SQLSMALLINT *         StringLengthPtr);
```

SQLGetInfo returns general information about the driver and data source associated with a connection. The information types are described in the SQLGetInfo section of the ODBC API Reference chapter in the *ODBC Programmer's Reference*.

There is an InfoType, which returns the username of the current database user:

SQLGetInfo Info Type	Value Returned
SQL_USER_NAME	A character string with the name used in a particular database, which can be different from the login name.

Previously, the username provided to ODBC (either explicitly or implicitly through SSO) was identical to the Teradata Database username, but with the LDAP authentication mechanism it is possible to map the mechanism username to a different Teradata Database username.

ODBC Pattern Escape Character

The ODBC pattern escape character is supported for the following ODBC functions (up to the limitations of Teradata):

- SQLColumnPrivileges
- SQLColumns
- SQLProcedureColumns
- SQLProcedures
- SQLTablePrivileges
- SQLTables

The Teradata LIKE clause does not contain an escape character in the same sense as ODBC; therefore, pattern escape characters for one string must either be in use or not be in use. The following table provides an example of the ODBC Pattern Escape Character:

Valid Syntax	Illegal Syntax
“table_all”	“tab_e_all”
“this%table”	“this%table_”
“table_all\%”	

If a string containing an escaped pattern character also contains an un-escaped pattern character, the string will be converted to all un-escaped characters. For example:

The string “table_emp%” will be converted to “table_emp%”.

Large Objects

This section describes a set of ODBC data types and their mapping to the Teradata Large Object Data Types.

LOB is a common term for relational data types that are able to contain large objects such as image, video, and audio data. These are objects with a size that significantly exceeds the size of traditional relational data types.

Teradata supports two types of Large Objects:

- BLOB
- CLOB

ODBC LOB Data Types

LONG data types is the ODBC terminology for Large Objects. ODBC defines two types of LONG data types:

- SQL_LONGVARBINARY
- SQL_LONGVARCHAR

As with other ODBC data types, these are abstract types that must be mapped to real data types as they are supported by the database.

Teradata LOB Data Types

The Teradata support contains two large object column types:

- BLOB – A variable-length binary string
- CLOB – A variable length character string

A BLOB is stored in the client system format and no translation occurs between the client and the server. A CLOB has a server side character set of Latin, Unicode® or Kanji. CLOBs are stored in the Teradata internal character format and translation occurs on the server.

Before the Teradata Native LOB Support was introduced, it was not possible to store really large objects in Teradata Database. In some instances, VARBYTE and VARCHAR data types, which are variable length with a maximum length of 64000 bytes, were sometimes used to work around this limitation.

Teradata SQL and LOB Types

Table columns containing LOB data can be created using a Create Table SQL statement in the following manner.

```
CREATE TABLE FOO (
     C01 INTEGER,
     C02 BLOB,
     C03 CLOB);
```

The CONVERT function of ODBC Driver for Teradata supports BLOB and CLOB data types.

CONVERT will map

```
"{fn CONVERT(value_exp, SQL_LONGVARBINARY)}" to
"(value_exp (BLOB))"
```

and

```
"{fn CONVERT(value_exp, SQL_LONGVARCHAR)}" to "(value_exp(CLOB))".
```

LOB Limitations

Presently BLOB and CLOB Large Object types are limited to a size of about 2 GB (2,097,088,000 bytes or characters). If the character set of a CLOB column is defined with Unicode for its character set, it has a maximum size limit of approximately 1 GB.

Application Programming Considerations

Inserting LOB Data

To insert LOB data, ODBC programs have to take into account that the size of LOB data may be very large and it will rarely be feasible to insert the data in one single large chunk. Rather it is advisable to insert LOB data piecemeal.

The ODBC DATA_AT_EXEC feature should be used to implement this piecemeal insertion into the database. When using this feature, data are inserted separately from the request to insert these data.

The following ODBC identifiers are available to specify the use of DATA_AT_EXEC:

- SQL_LEN_DATA_AT_EXEC(LEN)
- SQL_DATA_AT_EXEC

SQL_DATA_AT_EXEC is a pseudo length which indicates that the length is unknown at the time when the request to insert data is made and that data will be supplied later, possibly in several chunks.

SQL_LEN_DATA_AT_EXEC is a macro which has the same effect as SQL_DATA_AT_EXEC and in addition allows the total length of the data to be specified. For Teradata LOBs this length is optional.

Restrictions

Both SQLPutData and Teradata Database have a restriction regarding inserting LOBs. It is not possible to insert a NULL LOB value using SQLPutData.

Retrieving LOB Data

As described previously in Inserting LOB Data, LOB data is relatively large by nature, therefore, retrieving LOB data, like inserting LOB data, will likely also be done in parts, rather than reading the whole LOB in one operation.

To retrieve LOB data, use a series of SQLGetData calls.

Migration Issues

Applications that migrate to an ODBC driver with Native LOB Support may now behave differently than before the migration. This might be related to the use of LOBs.

There could be two reasons for the difference in behavior:

- The application relies on the ODBC feature that allows an application to query for all available types and uses that information to generate SQL
- The application uses the ODBC types SQL_LONGVARBINARY and SQL_LONGVARCHAR

Applications that query for all types will now receive information about two data types (BLOB and CLOB). This may in some cases cause problems. See SQLGetTypeInfo.

Applications that are using SQL_LONGVARBINARY and SQL_LONGVARCHAR may behave differently since these types are now mapped differently.

If an application falls into one of the mentioned categories it may be appropriate to consider disabling the Native LOB functionality by setting UseNativeLOBSupport=No.

Note:

> The option to set Native LOB Support is deprecated. For details, see Deprecated SQL Transformations

---

LOB Considerations for the ODBC API

This section lists ODBC functions that are specifically affected by the addition of LOB Support.

SQLPutData

It is not possible to insert a NULL value into a LOB column.

According to the ODBC Specification, there should be two ways to insert LOB NULL values when using DATA_AT_EXEC:

- The application calls SQLBindParameter with the data length set to SQL_NULL_DATA
- The application calls SQLPutData with the data length set to SQL_NULL_DATA

However, only the first of these alternatives is available due to a Teradata Database restriction.

SQLGetData

The following ODBC options relate to the use of SQLGetData, and are supported by the ODBC driver:

- SQL_GD_ANY_COLUMN
- SQL_GD_ANY_ORDER
- SQL_GD_BOUND

The options listed in the following table specify how columns can be used in a SQLGetData call.

Option	Description
SQL_GD_ANY_COLUMN	SQLGetData can be called for any unbound column, including those before the last bound column. Note: The columns must be called in order of ascending column number unless SQL_GD_ANY_ORDER is also returned.
SQL_GD_ANY_ORDER	SQLGetData can be called for unbound columns in any order. Note: SQLGetData can be called only for columns after the last bound column unless SQL_GD_ANY_COLUMN is also returned.
SQL_GD_BOUND	SQLGetData can be called for bound columns as well as unbound columns. Note: A driver cannot return this value unless it also returns SQL_GD_ANY_COLUMN.

ODBC supports these options for LOBs in order to provide the most flexible programming model and backward compatibility.

SQLGetTypeInfo

SQLGetTypeinfo returns information about a specified ODBC data type. It may also return information about SQL_ALL_TYPES, in which case it returns information about all the ODBC data types.

The LOB related information depends on whether or not LOB Support is available. LOB Support is available when the database supports the LOB feature.

Note:

> The option to set Native LOB Support is deprecated. For details, see Deprecated SQL Transformations.

If LOB Support is available, when SQLGetTypeInfo is called with SQL_ALL_TYPES, SQL_ALL_TYPES will include information about the SQL_LONGVARBINARY or SQL_LONGVARCHAR data types.

If LOB Support is *not* available, then SQL_ALL_TYPES will not include information about the SQL_LONGVARBINARY or SQL_LONGVARCHAR data types.

When SQLGetTypeInfo is called with either SQL_LONGVARBINARY or SQL_LONGVARCHAR, it will return information about these data types. See the table that follows.

LOB Support Available	SQL_ALL_TYPES includes LONG data types	SQL_LONGVARBINARY maps to	SQL_LONGVARCHAR maps to
Yes	Yes	BLOB	CLOB
Yes	Yes	VARBYTE(3200)	LONG VARCHAR

SQLColAttribute

SQLColAttribute returns information about columns in a result set. The fields listed in this table contain important LOB-specific information.

Field Identifier	Value

SQL_DESC_DISPLAY_SIZE	CLOB = 2097088000
	BLOB = 4194176000

SQL_DESC_NULLABLE	“No” Note: It is set to No because of the restriction on the use of LOB NULL values in SQLPutData.
SQL_DESC_SEARCHABLE	SQL_FALSE

Field Identifier	Value
	Note: BLOB and CLOB columns cannot be used in ORDER BY and in a WHERE clause.
SQL_DESC_TYPE	SQL_LONGVARBINARY for BLOB columns SQL_LONGVARCHAR for CLOB columns

SQLColumns

SQLColumns generates information about columns in a specific table. The fields listed in this table contain important LOB-specific information.

Column Name	Value
DATA_TYPE	SQL_LONGVARBINARY for BLOB columns SQL_LONGVARCHAR for CLOB columns
TYPE_NAME	BLOB for SQL_LONGVARBINARY CLOB for SQL_LONGVARCHAR
IS_NULLABLE	No **Note:** Set to No because of the restriction on the use of LOB NULL values in SQLPutData.

SQLGetInfo

SQLGetInfo provides information about general database characteristics. The information types listed in this table contain important LOB-specific information.

Information Type	Value
SQL_MAX_ROW_SIZE_INCLUDES_LONG	No
SQL_CONVERT_LONGVARBINARY	0x12fc38
SQL_CONVERT_LONGVARCHAR	0x12fc38

Information Type	Value
SQL_NEED_LONG_DATA_LEN	No

LOB Limitations

Presently LOB types are limited to a size of about 2 GB (2097088000 bytes or characters). Even though the limit is 2 GB, there are times when the LOB size gets twice as large.

The length of a CLOB based on 16-bit character set may get as large as 4 GB. The maximum display size of a BLOB is also 4 GB. To hold a counter of that magnitude (4 GB), an unsigned 32-bit value is required. Use 64-bit integers to accommodate future extensions.

If the character set of a CLOB column is defined with Unicode for its character set, it has a maximum size limit of approximately 1 GB.

Sample ODBC Programs Accessing LOB Data

LOB Retrieval Modes

Some Teradata Database instances contain Large Object (LOB) data types, such as BLOB (Binary Large Object) and CLOB (Character Large Object). The new Teradata ODBC Driver supports two ways of retrieving LOBs: *Deferred Mode* and *Smart LOB* mode. Optimize driver performance by configuring the appropriate retrieval mode:

- In *Deferred Mode*, the driver sends an additional query to retrieve each LOB. By default, the driver uses Deferred Mode.
- In *SLOB Mode*, the driver retrieves LOBs without sending any additional queries, but it may need to cache some LOBs in memory.

To optimize driver performance, use Deferred Mode when retrieving large LOBs you do not want to cache into memory; use SLOB Mode when you need to retrieve many small LOBs and want to avoid sending a large number of queries. For example, SLOB Mode can improve driver performance when retrieving geospatial data.

Note:

If SLOB Mode is not configured properly, it can decrease driver performance.

SLOB Mode Usage Guidelines

SLOB Mode is applicable only when certain size restrictions are met:

- The LOB to be retrieved must be smaller than the size specified by the Max Single LOB Bytes setting, `MaxSingleLOBBytes`. The driver falls back to using Deferred Mode when retrieving LOBs that exceed this size. The default value for this setting is 4000.
- If the total amount of LOB data being retrieved from a row exceeds the size specified by the Max Total LOB Bytes Per Row setting, `MaxTotalLOBBytesPerRow`, the driver uses Deferred Mode to retrieve the remaining LOBs from that row after using SLOB Mode to retrieve LOBs up to this size limit. The default value for this setting is 65536.

Before enabling SLOB Mode, be aware of the following conditions:

- Do not enable the Use Sequential Retrieval Only setting, `UseSequentialRetrievalOnly`, if there is any possibility you might retrieve LOBs from columns in a non-sequential order. For instance, do not enable this option and then execute a query that retrieves LOBs from the third column in a table, then from the first column, and then from the fifth column. If you enable this option and then retrieve LOBs non-sequentially, the driver discards the LOBs that are returned through SLOB Mode and must then retrieve them all again using Deferred Mode.
- When the Use Sequential Retrieval Only setting, `UseSequentialRetrievalOnly`, is disabled, the driver caches the other LOBs it reads while looking for the one to be retrieved. Caching large amounts of data in memory can decrease performance. To prevent this problem, set the size limits so the driver does not apply SLOB mode to large LOBs. LOB values that do not meet the requirements for SLOB Mode are retrieved using Deferred Mode instead, and, therefore, do not get cached.

Controlling the Scope of SLOB Mode Settings

You can configure the settings for SLOB Mode on the connection level or on the statement level. Because the optimal settings vary depending on the size of the specific LOBs you are retrieving, it may be useful to adjust the settings for each statement as you work with your data.

To configure settings for SLOB Mode on the connection level, specify the relevant driver options in a DSN or connection string. These settings apply to all queries and operations that are executed within the connection. You can override connection-level settings by using statement attributes. To configure settings for SLOB Mode on the statement level, set the following statement attributes:

- SQL_ATTR_MAX_SINGLE_LOB_BYTES: Use this attribute to specify the maximum size of the LOBs (in bytes) the driver can retrieve using SLOB Mode. LOBs that exceed this size are retrieved using Deferred Mode instead. This attribute corresponds to the Max Single LOB Bytes driver setting, `MaxSingleLOBBytes`.

- SQL_ATTR_MAX_LOB_BYTES_PER_ROW: Use this attribute to specify the maximum size of LOB data per row (in bytes) the driver can retrieve using SLOB Mode. If the total amount of LOB data contained in a row exceeds this size, the driver retrieves the LOBs from that row using Deferred Mode instead. This attribute corresponds to the Max Total LOB Bytes Per Row driver setting, `MaxTotalLOBBytesPerRow`.
- SQL_ATTR_USE_SEQUENTIAL_RETRIEVAL_ONLY: Use this attribute to indicate whether you are retrieving LOB data from columns in sequential order. This attribute corresponds to the Use Sequential Retrieval Only driver setting, `UseSequentialRetrievalOnly`.

Max Single LOB Bytes

Key Name	Default Value	Required
MaxSingleLOBBytes	4000	No

Max Total LOB Bytes Per Row

Key Name	Default Value	Required
MaxTotalLOBBytesPerRow	65536	No

This is the maximum size of LOB data per row (in bytes) the driver can retrieve using Smart LOB (SLOB) Mode. If the total amount of LOB data contained in a row exceeds this size, the driver retrieves the LOBs from that row using Deferred Mode instead.

If this option is set to 0, SLOB Mode is disabled, and the driver retrieves all LOB data using Deferred Mode.

Note:

> As an alternative to using this option, specify this setting on the statement level rather than the connection level by using the SQL_ATTR_MAX_LOB_BYTES_PER_ROW statement attribute.

Use Sequential Retrieval Only

Key Name	Default Value	Required
UseSequentialRetrievalOnly	Clear (0)	No

This option indicates to the driver whether you are retrieving LOB data from columns in sequential order or non-sequential order. When working in Smart LOB (SLOB) Mode, the driver reads and caches LOB data differently depending on this setting.

- Enabled (1): When working in SLOB Mode, the driver does not cache the other LOBs it reads while looking for the one to be retrieved. Because the driver can retrieve LOBs in a single pass if they are queried sequentially, the driver does not need to cache them.
- Disabled (0): When working in SLOB Mode, the driver caches the other LOBs that it reads while looking for the one to be retrieved. This caching allows the driver to successfully retrieve SLOBs in any order.

Note:

Do not enable this option if there is any possibility you might retrieve LOBs from columns in a non-sequential order. For instance, do not enable this option and then execute a query that retrieves LOBs from the third column in a table, then from the first column, and then from the fifth column. If you enable this option and then retrieve LOBs non-sequentially, the driver discards the LOBs that are returned through SLOB Mode and must then retrieve them all again using Deferred Mode.

As an alternative to using this option, you can specify this setting on the statement level rather than the connection level by using the SQL_ATTR_USE_SEQUENTIAL_RETRIEVAL_ONLY statement attribute.

Differences in Driver Implementation

The new driver has defaults to determine which mode to transfer LOB data with. These default sizes may be different than past versions as we try to provide values that fit most cases.

If the LOB is too large, the driver will use Deferred Mode as this is faster if you do not want to cache into memory. SLOB is better in certain cases, such as when working with geospatial data.

There are differences between old driver and the new driver's SLOB implementation:

- The new driver can cache, or attempt to cache, up to 2GB of all SLOBs in a row.
- The old driver only caches two Response Buffers, up to 32MB.

The new driver has three configuration parameters:

- 1- Enable SLOB Random Access
- 2- Max size of one SLOB
- 3- Max size of all SLOBs in a Row

When #1 is set to True, the new driver will cache up to #3, potentially up to 2GB.

Creating a Table with BLOB Data

The following CREATE TABLE statement is used to create the table used in the following code samples.

```
create table blobtable(id integer, image blob);
```

Inserting LOB Data

This section contains sample source code that inserts LOB data.

The first step is to prepare a parameterized insert statement. The first parameter is a numeric key used to identify this row. The second parameter is the BLOB data.

Note:

The second parameter is specified as a SQL_LONGVARBINARY data type.

The length of the LOB data is specified in the SQL_LEN_DATA_AT_EXEC macro call.

```
SQLINTEGER       id;
SQLCHAR     image[IMAGEPART_LEN ];
SQLINTEGER  cbid = 0;
SQLUINTEGER cbimage = 0;
SQLRETURN   retcode;

retcode = SQLPrepare(hstmt,
                    (SQLCHAR *)"insert into blobtable (id, image) values (?, ?)",
                     SQL_NTS);
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
         // Bind the parameters.
    retcode = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT,
                               SQL_C_SLONG, SQL_INTEGER, 10, 0, &id, 0, &cbid);
    retcode = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
                               SQL_LONGVARBINARY, LOBSIZE, 0, (SQLPOINTER) 2, 0,
                               &cbimage);
    //Set the 1. parameter (id)
         id = 1001;
    //Set the 2. parameter (image)
    cbimage = SQL_LEN_DATA_AT_EXEC(0);
```

Note:

Setting the length to 0 as in SQL_LEN_DATA_AT_EXEC(0) means that the length is unknown.

The next step is to insert the data. The LOB data may be very large and the data are therefore inserted in chunks.

```
      // Execute the request
      retcode = SQLExecute(hstmt);

      // For the data-at-execution parameter (2. parameter)
      // call SQLParamData and SQLPutData repeatedly
      // to insert the BLOB data in parts.
      while (retcode == SQL_NEED_DATA) {
          retcode = SQLParamData(hstmt, NULL);
          while (GetUserData(image, IMAGEPART_LEN )) {
              SQLPutData(hstmt, image, IMAGEPART_LEN );
          }
      }
}
```

Retrieving LOB Data

LOB data should be retrieved using SQLGetData to be able to retrieve and process LOB data in parts. The following code fragment illustrates that. Note the loop to handle 2 BLOB column.

```
SQLINTEGER   id;
SQLCHAR      image[IMAGEPART_LEN ];
SQLINTEGER   cbid = 0;
SQLUINTEGER  cbimage = 0;
SQLRETURN    retcode;

retcode = SQLExecDirect(hstmt, (SQLCHAR*)"select id, image from blobtable",
SQL_NTS);

while (retcode == SQL_SUCCESS ) {
    retcode = SQLFetch(hstmt);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO){

        /* Get data for column 1 */
        SQLGetData(hstmt, 1, SQL_C_ULONG, &id, 0, &cbid);

        /* Get data for column 2 */
        while (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO )
        {
          retcode= SQLGetData(hstmt, 2, SQL_C_CHAR, image, IMAGEPART_LEN,&cbimage);
            switch (retcode)
            {
            case SQL_SUCCESS_WITH_INFO:
            case SQL_SUCCESS:
                processimage(id, image);
                break;
            case SQL_NO_DATA:
                break;
            case SQL_ERROR:
            default:
                processerror(retcode);
                break;
            }
        }
    }
```

User-Defined Functions

ODBC support of Teradata Database user-defined functions is transparent to the ODBC user, and should have no effect on existing user applications.

The full specification of the syntax, format and rules for both creating and invoking User Defined Functions (UDF) is beyond the scope of this document. See *SQL Reference: UDF, UDM, and External Stored Procedure Programming* for details on how to create and invoke user-defined functions.

Return Codes

The following tables lists the ODBC API return codes for calls that execute SQL statements performing builds of user-defined functions:

Return Code	Description
SQL_SUCCESS_WITH_INFO	The build of the UDF was performed on the DBS and the UDF was created successfully. The output should be examined for possible warnings in the build of the UDF.
SQL_ERROR	The build of the UDF failed on the DBS and the UDF was not created.

In both cases the output from the build is available to the client as a sequence of diagnostic status records that may be retrieved using SQLGetDiagRecord() and SQLGetDiagField().

The first diagnostic record contains the status of the UDF creation from the DBS.

The subsequent diagnostic records contain all output from the build as message texts and repeat the SQLSTATE and native error code from the first record.

Parameter Markers

Parameters to UDFs in Teradata Database are input parameters. ODBC allows UDF parameters to be specified by parameter markers and bound by SQLBindParameter with InputOutputType SQL_PARAM_INPUT.

Table Functions

A table function is a form of UDF that can only be specified in the FROM clause of a SELECT statement. It is treated as a derived table subquery.

A table function returns one table row at a time for each invocation. The function is written in C or C++ and can only be invoked in the FROM clause of a SELECT statement. The function cannot be invoked from any other place.

Table functions run in parallel on all AMPs, however, the table function developer can determine which AMPs will participate, and which AMPs will not participate in the function.

Table functions are created with a CREATE FUNCTION statement. The dictionary entries are created for the new function type. There is no difference in how that process works. The UDF code is then compiled and linked, and the library distributed as required to all nodes.

Because the function produces a table, a row at a time, it requires the column definitions and their data types.

Like a UDF, if the table function does I/O, the function needs to include the new external security clause that is used to associate a client user with the execution of the table function.

It either associates the function with the INVOKER of the function, or with the DEFINER of the function. This is controlled by the developer of the function.

A table function returns a table a row at a time in a loop to the caller of the function. The function is capable of reading an external file or simply producing the rows of a table based on the input arguments passed to it. It does this by having the user specify the function in place of the FROM clause in an SQL SELECT statement.

The table function essentially creates a derived table from an external source (a native OS file or message queue). It can also produce the rows solely from the input arguments. For example an input argument could be a reference to a CLOB that contains XML text. From that CLOB it could parse the XML text and output a whole set of SQL rows.

Restrictions

The ability to create a UDF function with the UDF source located on the client is restricted. If you try to create a UDF from source on the client, a message will be returned indicating UDFs created from source on the client are not supported.

User-Defined Types and User-Defined Methods

User-Defined Types (UDT) and User-Defined Methods (UDM) are supported by ODBC Driver for Teradata. The full specification of the syntax, format and rules for using UDTs and UDMs is beyond the scope of this document. See *SQL Reference: UDF, UDM, and External Stored Procedure Programming* for details on how to create and use UDTs and UDMs.

Importing and Exporting UDT Values

A UDT can exist only on the Teradata Database server. Each UDT has an associated ***`from-sql routine`*** and ***`to-sql routine`***. The ***`from-sql routine`*** generates a predefined type value from a UDT. It is automatically invoked when a UDT is exported from the Teradata Database server to a client system. The ***`to-sql routine`*** constructs a UDT value from a predefined type value. It is automatically invoked when importing values from a client system into a UDT on the Teradata Database server. The ***`from-sql routine`*** and ***`to-sql routine`*** create a mapping between a UDT and a predefined type. The predefined type is called the external type of a UDT. A client application only deals with the external type; it does not deal directly with the UDT value.

For example, if a UDT named FULLNAME exists and the external type associated with FULLNAME is VARCHAR(46), then during an export of FULLNAME values, the Teradata Database server converts the values from FULLNAME values to VARCHAR(46) values by invoking the ***`from-sql routine`*** associated with the FULLNAME UDT. As a result, the client should expect to receive the data in the same format as it receives VARCHAR(46) values.

Similarly, when values are provided by the client for import into a FULLNAME UDT, the client should provide values like it would provide values for a VARCHAR(46) field, and the Teradata Database server then converts the values from VARCHAR(46) to FULLNAME values by invoking the ***`to-sql routine`*** associated with the FULLNAME UDT.

Importing UDT Values

ODBC Driver for Teradata can be used to import values into a UDT column. Use ODBC Driver for Teradata to import values into tables containing UDT columns in the same manner as is done for other tables. The application handles the UDT columns in the same manner as is done for other tables. The application should handle the UDT columns like it would handle a column containing the external type associated with that UDT. This external type is always a predefined Teradata Database type. When inserting into a UDT column, Teradata Database automatically converts from the external type to UDT internal format.

Using User-Defined Types with ODBC

ODBC applications always transfer values of a UDT as values of the external type associated with the UDT and the external type is always a predefined Teradata Database type. For example, when the select-list of a SELECT statement contains a UDT expression, the Teradata Database server automatically converts the UDT data to its external type before returning the data to the ODBC application. When inserting into a UDT column, the Teradata Database server automatically converts the external type data to UDT internal format. Therefore, the use of UDTs in requests and result sets is transparent to the ODBC application.

Creating a UDT using ODBC Driver for Teradata is done in a similar manner to creating other database objects. The ODBC client calls the ODBC SQLExecDirect() functions (or SQLPrepare() plus SQLExecute() to issue the appropriate “CREATE TYPE…” SQL statement followed by “CREATE METHOD…”, “CREATE TRANSFORM…”, and “CREATE ORDERING…” statements as needed to fully create the type.

UDTs are visible in the ODBC catalog functions and in Results.

User-Defined Methods

The use of UDMs in ODBC is similar to the use of UDFs in ODBC.

Refer to User-Defined Functions.

Return Codes

SQL statements that CREATE/REPLACE/ALTER/DROP user-defined types or methods and SQL statements that CREATE/REPLACE/ALTER UDT casts, orders, or transforms may result in UDFs being built on the database server. The ODBC API return codes for calls that execute these SQL statements are the same as for calls that build user-defined functions.

See User-Defined Functions.

Catalog Functions

Some catalog functions in the new driver behave differently compared to the old driver. The functions, and the differences between the new driver and the old driver are described in the following sections.

Catalog Function	Description
SQLTables	Returns the value “TYPE” in the TABLE_TYPE column for user-defined types. The type name is returned in the TABLE_NAME column.
SQLColumns	Returns the value SQL_UNKNOWN_TYPE (zero) in the DATA_TYPE for a UDT column. The UDT name is returned in the TYPE_NAME column.
SQLProcedures	Returns names of user-defined methods in addition to names of macros, procedures, and user-defined functions. The value of the PROCEDURE_TYPE column for a user-defined method is SQL_PT_PROCEDURE.
SQLProcedureColumns	Returns parameter information for user-defined methods. Also, parameter types might be UDTs and these are returned as for the SQLColumns catalog function (SQL_UNKNOWN_TYPE in DATA_TYPE column and UDT name in TYPE_NAME column). The output for TD_ANYTYPE parameters results in the value SQL_UNKNOWN_TYPE in the DATA_TYPE column and the string “TD_ANYTYPE” (without quotes) in the TYPE_NAME column.

All Catalog Functions

The old driver returns some column names differently depending on whether the driver is working in an ODBC 2.x or 3.x environment. The new driver always returns ODBC 3.x column names, even when working in an ODBC 2.x environment.

SQLBindParameter

Binding Date, Time, and Timestamp Literals

When binding any of these types of literals as a parameter, the old driver accepts literals that contain extra spaces. The new driver only accepts literals that are specified in the exact format specified in the ODBC specification. If you try to bind a literal that does not use the required format, the new driver returns the following error:

```
[Simba][Support] (40550) Invalid character value for cast specification.
```

For more information about the required format, see *Date, Time, and Timestamp Literals* in the *ODBC Programmers’ Reference* located at:

https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/date-time-and-timestamp-literals.

As an example, both drivers accept the value `{d '1995-01-15'}`, but only the old driver accepts the value `{ d '1995-01-15'}.` Note the missing space in the first value before 'd', but the inserted space in the second value.

Returning Error Information

The new driver responds to certain errors differently than the old driver as described below.

- The new driver does not support SQL_ARRAY_STATUS_PTR and SQL_DIAG_ROW_NUMBER for parameter sets, so the driver does not set these properties when an error occurs in a query that contains a parameter set.
- If no errors occurred, but some parameter sets were ignored, then the old driver sets SQL_ATTR_PARAMS_PROCESS_PTR to SQL_ATTR PARAMSET_SIZE minus the number of ignored sets. The new driver sets SQL_ATTR_PARAMS_PROCESSED_PTR to the exact value of SQL_ATTR_PARAMSET_SIZE.

  This behavior of the new driver is consistent with the ODBC specification.

  For more information, see the *Error Information* in *SQLBindParameter Function* in the *ODBC Programmers' Reference:* https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/sqlbindparameter-function.

Using SQL_DEFAULT_PARAM

When SQL_DEFAULT_PARAM is specified as an indicator via the StrLen_or_IndPtr argument, the old driver ignores it and instead uses the value stored in the buffer. Depending upon the environment in which SQL_DEFAULT_PARAM is being used, the new driver returns one of the following:

- If it is specified in a stored procedure call, the new driver uses the value NULL to complete the stored procedure call and returns SQL_SUCCESS_WITH_INFO.
- If it is specified in something other than a stored procedure call, the new driver returns SQL_ERROR and does not execute the statement.

The behavior of the new driver is consistent with the ODBC specification, which states that SQL_DEFAULT_PARAM is valid only when used with a stored procedure call. Teradata Database does not support default parameters for stored procedures, so the new driver uses NULL as the value for SQL_DEFAULT_PARAM.

For more information, see *SQLBindParameter Function* in the *ODBC Programmers' Reference:* https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/sqlbindparameter-function.

Query Parameter Binding

When binding query parameters, the new driver supports different use cases than the old driver.

Old Driver	**New Driver**
When binding JSON or WJSON data to a parameter, the old driver returns an error.	The new driver supports binding JSON and WJSON data to parameters.
The old driver does not support binding for SQL_DECIMAL and SQL_NUMERIC data that has a negative scale.	The new driver supports binding for these types of values, which is consistent with the ODBC specification: https://docs.microsoft.com/en-us/sql/odbc/reference/appendixes/decimal-digits.
The old driver does not support binding for SQL_DECIMAL and SQL_NUMERIC data that has a precision that is less than 1.	The new driver supports binding for SQL_DECIMAL and SQL_NUMERIC data that has a precision of 0, in order to support binding for NULL values.
When calling SQLBindParameter, the old driver verifies the column sizes of the data and then modifies the column sizes of the input if needed.	The new driver does not verify column sizes or modify the input from SQLBindParameter. When binding LOB data types, the new driver uses the values returned by `GetMaxLobBytes()` or `MaxJSONBytes()` as the maximum column size of the LOB data.

Output Parameter Binding

When binding output parameters, a data type conversion is sometimes required. In this case, the old driver converts the output data to its corresponding SQL type regardless of the data types specified in SQLBindParameter. In contrast, the new driver converts the output data to the specified SQL type, or returns a conversion error if the types are not compatible. The new driver's behavior is consistent with the ODBC specification.

For example, given the following procedure:

```
create procedure CharOutputStoredprocedure(OUT param1 CHAR)
cs1: BEGIN
SET param1 = 'A';
END cs1;
```

Assume that you bind the output parameters as follows:

```
SQLBindParameter(
        stmt,
        1,
        SQL_PARAM_OUTPUT,
        SQL_C_CHAR,
        SQL_CHAR,
        1024,
        0,
        &out,
        1024,
        &cbRetParam)
```

The given SQL type is the same as the SQL type of the original parameter, so the data does not need to be converted. The old driver and the new driver both successfully bind the data and return the value A.

However, if you bind the output parameter as follows, then the data must be converted from CHAR to SQL_INTEGER:

```
SQLBindParameter(
        stmt,
        1,
        SQL_PARAM_OUTPUT,
        SQL_C_CHAR,
        SQL_INTEGER,
        1024,
        0,
        &out,
        1024,
        &cbRetParam)
```

The conversion fails because A is not a valid SQL_INTEGER value. The old driver handles this situation by converting and binding the output data to SQL_CHAR instead. The new driver tries to convert the data to SQL_INTEGER and then returns a conversion error with SQL state 22018.

SQLBindParameter and Data Types with Fractional Seconds

When calling SQLBindParameter with a data type that contains fractional seconds, you must set the DecimalDigit to a value up to 6, the maximum the database supports. Previous to 16.20, this could be any number between 0-6 with the same result as if you were to set it to 6, but this is incorrect and no longer supported.

As an example, TIMESTAMP(0) is no longer valid if you are passing in a fractional second because according to the ODBC specification, you have specified zero decimal digits and will receive an error if you try to pass in a fractional second.

You can specify a maximum of up to 6 decimal digits, as this is the limit of the Teradata Database.

It is acceptable to send less than your specified number of decimal digits.

You can optionally pad out to 9 decimal digits with zeros without issue.

SQLCancel

The new driver responds to timing or processing state of the statement differently than the old driver.

The ODBC specification defines the following behavior for SQLCancel in situations where no processing has been done for the statement:

- In ODBC 3.5, SQLCancel has no effect on the statement. To close a cursor, applications need to call SQLCloseCursor instead of SQLCancel.
- In ODBC 2.x, SQLCancel has the same effect as SQLFreeStmt with the SQL_CLOSE option. This behavior is defined only for the sake of completeness; applications should call SQLFreeStmt or SQLCloseCursor instead to close cursors.

For more information, see *SQLCancel Function* in the *ODBC API Reference*: https://msdn.microsoft.com/en-us/library/ms714112%28v=vs.85%29.aspx.

Old Driver	New Driver
When executing statements asynchronously, if the execution is completed before SQLCancel is called, the old driver returns HY008.	The new driver returns the result of the statement execution, (SQL_SUCCESS or SQL_ERROR).
If SQLCancel is called before any processing has been done for the statement, the old driver closes the statement regardless of whether the driver is working in ODBC 2.x mode or ODBC 3.x mode. Closing the statement reflects behavior that is consistent with the ODBC 2.x specification, but not the ODBC 3.x specification.	The new driver does not close the statement when it is working in ODBC 3.x mode, and this behavior is consistent with the ODBC 3.x specification.

SQLForeignKeys

In the old driver, the columns UPDATE_RULE and DELETE_RULE are returned as empty strings. In the new driver, these columns are instead returned as NULL.

SQLGetConnectAttr

The following table lists the results of the new and old drivers.

Function	Old Driver Returns	New Driver Returns
SQL_ATTR_ASYNC_ENABLE	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 0, *pcbErrorMsg = 50, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Invalid Attribute"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = SQL_ASYNC_ENABLE_OFF = 0, *StringLengthPtr = 4`
SQL_ATTR_DISCONNECT_BEHAVIOR	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 0, *pcbErrorMsg = 50, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver]`	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 10210, *pcbErrorMsg = 75, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute identifier invalid or not supported: 114"`

Function	Old Driver Returns	New Driver Returns
	`Invalid Attribute"`	
SQL_ATTR_ENLIST_IN_DTC	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 0, *pcbErrorMsg = 50, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Invalid Attribute"`	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 10210, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute identifier invalid or not supported: 1207"`
SQL_ATTR_PACKET_SIZE	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 4096, *StringLengthPtr = 4`
SQL_ATTR_TRANSLATE_LIB	`Return: SQL_SUCCESS=0`	`Return: SQL_SUCCESS=0 Out:`

Function	Old Driver Returns	New Driver Returns

	`Out: *ValuePtr = "쨀쬺쬺쬺쬺쬺쬺쬺쬺쬺쬺쬺쬺쬺쬺쬺쬺...",`
	`*StringLengthPtr = 0`
		`*ValuePtr = "",`
		`*StringLengthPtr = 0`

SQLGetDiagField

At this time, the new driver does not support setting the following:

- SQL_DIAG_CURSOR_ROW_COUNT
- SQL_DIAG_ROW_COUNT

SQLGetInfo

The following table lists the results of the new and old drivers.

Function	Old Driver Returns	New Driver Returns

SQL_SQL_CONFORMANCE	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>,`
	`*StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00",`
	`*pfNativeError = 0,`
	`*pcbErrorMsg = 76,`
	`*ColumnNumber = -1,`
	`*RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = SQL_SC_SQL92_ENTRY = 1,`
		`*StringLengthPtr = 4`

SQL_CATALOG_NAME_SEPARATOR	`Return: SQL_SUCCESS=0`
	`Out: *InfoValuePtr`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = ".",`
		`*StringLengthPtr = 2`

Function	Old Driver Returns	New Driver Returns
	`= <unmodified>, *StringLengthPtr = 0`	

SQL_CREATE_TABLE	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`

SQL_DROP_TABLE	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`

SQL_DROP_VIEW	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr =`	`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr =`

Function	Old Driver Returns	New Driver Returns

	`<unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`0x00000000, *StringLengthPtr = 4`

SQL_GETDATA_EXTENSIONS	`Return: SQL_SUCCESS=0`
	`Out: *InfoValuePtr = 0x0000000B = SQL_GD_ANY_COLUMN | SQL_GD_ANY_ORDER | SQL_GD_BOUND, *StringLengthPtr = 4`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x0000000F = SQL_GD_ANY_COLUMN | SQL_GD_ANY_ORDER | SQL_GD_BLOCK | SQL_GD_BOUND, *StringLengthPtr = 4`

SQL_INDEX_KEYWORDS	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`

SQL_LOCK_TYPES	`Return: SQL_SUCCESS=0`	`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr =`

Function	Old Driver Returns	New Driver Returns
	`Out: *InfoValuePtr = 0x00000001 = SQL_LCK_NO_CHANGE, *StringLengthPtr = 4`	`0x00000002 = SQL_LCK_EXCLUSIVE, *StringLengthPtr = 4`

SQL_MAX_ASYNC_CONCURRENT_STATEMENTS	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0, *StringLengthPtr = 4`

SQL_NUMERIC_FUNCTIONS	`Return: SQL_SUCCESS=0`
	`Out:*InfoValuePtr = 0x00014D01 = SQL_FN_NUM_ABS | SQL_FN_NUM_EXP | SQL_FN_NUM_LOG | SQL_FN_NUM_MOD | SQL_FN_NUM_SQRT | SQL_FN_NUM_PI, StringLengthPtr = 4`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00194D01 = SQL_FN_NUM_ABS | SQL_FN_NUM_EXP | SQL_FN_NUM_LOG | SQL_FN_NUM_MOD | SQL_FN_NUM_SQRT | SQL_FN_NUM_PI | SQL_FN_NUM_LOG10 | SQL_FN_NUM_POWER, *StringLengthPtr = 4`

SQL_ODBC_SAG_CLI_CONFORMANCE	`Return: SQL_SUCCESS=0`
	`Out: *InfoValuePtr = SQL_OSCC_COMPLIANT = 1, *StringLengthPtr = 2`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = SQL_OSCC_NOT_COMPLIANT = 0, *StringLengthPtr = 2`

Function	Old Driver Returns	New Driver Returns
SQL_POS_OPERATIONS	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000001 = SQL_POS_POSITION, *StringLengthPtr = 4	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4
SQL_QUALIFIER_NAME_SEPARATOR	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = 0	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = ".", *StringLengthPtr = 2
SQL_SCROLL_CONCURRENCY	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000001 = SQL_SCCO_READ_ONLY, *StringLengthPtr = 4	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000003 = SQL_SCCO_READ_ONLY | SQL_SCCO_LOCK, *StringLengthPtr = 4
SQL_SQL92_GRANT	Return: SQL_ERROR=-1 Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000010 = SQL_SG_WITH_GRANT_OPTION, *StringLengthPtr = 4
SQL_SQL92_PREDICATES	Return: SQL_ERROR=-1Return: SQL_ERROR=-1 Out: *InfoValuePtr	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00003F07 = SQL_SP_EXISTS | SQL_SP_ISNOTNULL |

Function	Old Driver Returns	New Driver Returns
	= <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"	SQL_SP_ISNULL | SQL_SP_UNIQUE | SQL_SP_LIKE | SQL_SP_IN | SQL_SP_BETWEEN | SQL_SP_COMPARISON | SQL_SP_QUANTIFIED_COMPARISON, *StringLengthPtr = 4
SQL_SQL92_RELATIONAL_JOIN_OPERATORS	Return: SQL_ERROR=-1 Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x0000015A = SQL_SRJO_CROSS_JOIN | SQL_SRJO_FULL_OUTER_JOIN | SQL_SRJO_INNER_JOIN | SQL_SRJO_LEFT_OUTER_JOIN | SQL_SRJO_RIGHT_OUTER_JOIN, *StringLengthPtr = 4
SQL_SQL92_STRING_FUNCTIONS	Return: SQL_ERROR=-1 Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0,	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000006 = SQL_SSF_LOWER | SQL_SSF_UPPER, *StringLengthPtr = 4

Function	Old Driver Returns	New Driver Returns
	`*pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`	
SQL_STATIC_SENSITIVITY	`Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`	`Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000003 = SQL_SS_ADDITIONS | SQL_SS_DELETIONS, *StringLengthPtr = 4`
SQL_XOPEN_CLI_YEAR	`Return: SQL_ERROR=-1 Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`	`Return: SQL_SUCCESS=0 Out: *InfoValuePtr = "1995", *StringLengthPtr = 8`

SQLGetStmtAttr

The following table lists the results of the new and old drivers.

Function	Old Driver Returns	New Driver Returns
SQL_ATTR_CURSOR_SCROLLABLE	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = 4`
SQL_ATTR_CURSOR_SENSITIVITY	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = 4`
SQL_ATTR_KEYSET_SIZE	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = <unmodified>`	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HY092",`

Function	Old Driver Returns	New Driver Returns
		`*pfNativeError = 10210, *pcbErrorMsg = 73, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute identifier invalid or not supported: 8"`
SQL_ATTR_RETRIEVE_DATA	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = SQL_RD_ON = 1, *StringLengthPtr = 4`
SQL_ATTR_ROW_NUMBER **NOTICE** The old driver always returns SQL_ROW_NUMBER_UNKNOWN; the new driver returns the actual number of the current row in the entire result set.	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = <unmodified>`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 1, *StringLengthPtr = 4`
SQL_ATTR_SIMULATE_CURSOR	`Return: SQL_SUCCESS=0 Out:`	`Return: SQL_ERROR=-1 Out:`

Function	Old Driver Returns	New Driver Returns
	`*ValuePtr = SQL_SC_NON_UNIQUE = 0, *StringLengthPtr = <unmodified>`	`*ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HY092", *pfNativeError = 10210, *pcbErrorMsg = 74, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute identifier invalid or not supported: 10"`

SQLGetTypeInfo

The new driver returns an additional custom column "USER_DATA_TYPE" at index 20.

New driver columns:

```
1, TYPE_NAME, 9, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
2, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
3, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
4, LITERAL_PREFIX, 14, SQL_VARCHAR=12, 32, 0, SQL_NULLABLE=1
5, LITERAL_SUFFIX, 14, SQL_VARCHAR=12, 32, 0, SQL_NULLABLE=1
6, CREATE_PARAMS, 13, SQL_VARCHAR=12, 32, 0, SQL_NULLABLE=1
7, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
8, CASE_SENSITIVE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
9, SEARCHABLE, 10, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
10, UNSIGNED_ATTRIBUTE, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
11, FIXED_PREC_SCALE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
12, AUTO_UNIQUE_VALUE, 17, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
13, LOCAL_TYPE_NAME, 15, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
14, MINIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
15, MAXIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
16, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
17, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
```

```
18, NUM_PREC_RADIX, 14, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
19, INTERVAL_PRECISION, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
20, USER_DATA_TYPE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
21, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

Old driver columns:

```
1, TYPE_NAME, 9, SQL_VARCHAR=12, 39, 0, SQL_NO_NULLS=0
2, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
3, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
4, LITERAL_PREFIX, 14, SQL_VARCHAR=12, 11, 0, SQL_NULLABLE=1
5, LITERAL_SUFFIX, 14, SQL_VARCHAR=12, 18, 0, SQL_NULLABLE=1
6, CREATE_PARAMS, 13, SQL_VARCHAR=12, 18, 0, SQL_NULLABLE=1
7, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
8, CASE_SENSITIVE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
9, SEARCHABLE, 10, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
10, UNSIGNED_ATTRIBUTE, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
11, FIXED_PREC_SCALE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
12, AUTO_UNIQUE_VALUE, 17, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
13, LOCAL_TYPE_NAME, 15, SQL_VARCHAR=12, 39, 0, SQL_NULLABLE=1
14, MINIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
15, MAXIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
16, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
17, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
18, NUM_PREC_RADIX, 14, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
19, INTERVAL_PRECISION, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
20, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

SQLProcedureColumns

The old driver uses SQL_VARCHAR for string columns. The new driver uses SQL_WVARCHAR for string columns and returns two additional custom columns, at index 20 and 21.

New driver columns:

```
1, PROCEDURE_CAT, 13, SQL_VARCHAR=12, 1024, 0, SQL_NULLABLE=1
2, PROCEDURE_SCHEM, 15, SQL_VARCHAR=12, 30, 0, SQL_NULLABLE=1
3, PROCEDURE_NAME, 14, SQL_VARCHAR=12, 30, 0, SQL_NO_NULLS=0
4, COLUMN_NAME, 11, SQL_VARCHAR=12, 30, 0, SQL_NO_NULLS=0
5, COLUMN_TYPE, 11, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
6, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
7, TYPE_NAME, 9, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
8, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
9, BUFFER_LENGTH, 13, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
10, DECIMAL_DIGITS, 14, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
```

```
11, NUM_PREC_RADIX, 14, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
12, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
13, REMARKS, 7, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
14, COLUMN_DEF, 10, SQL_VARCHAR=12, 4000, 0, SQL_NULLABLE=1
15, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
16, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
17, CHAR_OCTET_LENGTH, 17, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
18, ORDINAL_POSITION, 16, SQL_INTEGER=4, 10, 0, SQL_NO_NULLS=0
19, IS_NULLABLE, 11, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
20, IS RESULT SET COLUMN, 20, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
21, USER_DATA_TYPE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
22, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

Old driver columns:

```
1, PROCEDURE_CAT, 13, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
2, PROCEDURE_SCHEM, 15, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
3, PROCEDURE_NAME, 14, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
4, COLUMN_NAME, 11, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
5, COLUMN_TYPE, 11, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
6, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
7, TYPE_NAME, 9, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
8, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
9, BUFFER_LENGTH, 13, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
10, DECIMAL_DIGITS, 14, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
11, NUM_PREC_RADIX, 14, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
12, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
13, REMARKS, 7, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
14, COLUMN_DEF, 10, SQL_VARCHAR=12, 60, 0, SQL_NULLABLE=1
15, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
16, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
17, CHAR_OCTET_LENGTH, 17, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
18, ORDINAL_POSITION, 16, SQL_INTEGER=4, 10, 0, SQL_NO_NULLS=0
19, IS_NULLABLE, 11, SQL_VARCHAR=12, 3, 0, SQL_NULLABLE=1
20, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

SQLTables

When using pattern matching with a wildcard character (%), the default catalog metadata is null, so the new driver returns SQL_INTEGER as the SQL Type for some columns.

For example, for the following call:

```
SQLTables(<empty string>, %, <empty string>, <null pointer>)
```

New driver returns:

```
icol, szColName, *pcbColName, *pfSqlType, *pcbColDef, *pibScale, *pfNullable
1, TABLE_CAT, 9, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
2, TABLE_SCHEM, 11, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
3, TABLE_NAME, 10, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
4, TABLE_TYPE, 10, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
5, REMARKS, 7, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
```

Old driver returns:

```
icol, szColName, *pcbColName, *pfSqlType, *pcbColDef, *pibScale, *pfNullable
1, TABLE_CAT, 9, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
2, TABLE_SCHEM, 11, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
3, TABLE_NAME, 10, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
4, TABLE_TYPE, 10, SQL_VARCHAR=12, 17, 0, SQL_NULLABLE=1
5, REMARKS, 7, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
```

Results

UDTs information from results is available through the SQLColAttribute as shown in the following table:

Field Identifier	Information returned in	Description
SQL_COLUMN_TD_UDT_INDICATOR	NumericAttributePtr	A numeric value that specifies the UDT indicator. If the SIP is disabled UDT Indicator is unknown, this variable contains a value zero.
SQL_COLUMN_TD_UDT_NAME	CharacterAttributePtr	Fully Qualified Type Name (for example, "SYSUDTLIB.ST_GEOMETRY", "SYSUDTLIB.SSN", and so on). If the SIP is disabled UDT data type name is unknown, this variable contains an empty string.

Restrictions

When creating UDMs, the “CREATE METHOD...” statement must indicate that the file(s) containing the definitions of the methods are located on the server; creating UDMs from file(s) located on the client is not supported. If you try to create a UDM from file(s) on the client, a message is returned indicating that UDMs created from file(s) on the client are not supported. This restriction is similar to the restriction for UDFs.

Parameter Arrays

ODBC Driver for Teradata supports parameter arrays with limitations as described in Restrictions.

Parameter arrays are directly supported by Teradata Database as a single SQL statement with a set of parameter values. This is also called a DML array request since parameter arrays are only allowed for DML statements.

Prior to Teradata Database support of DML array requests, ODBC Driver for Teradata emulated parameter array support by executing a multi-statement request with a single SQL statement for each set of parameter values.

Such emulation is still used for multi-statement requests containing one or more statements using parameter arrays. For example, the following code executes two SQL statements in a single request:

```
"SELECT * from T; INSERT INTO T values(?,?)"
```

where INSERT uses a parameter array.

In this case, an application may want to execute the two statements separately instead of executing a multi-statement request. In other cases, the application may force the emulated mode by appending a no-operation SQL statement to surmount a restriction for DML array requests.

Performance

DML array support performance is superior to emulation using multi-statement requests. Teradata Database can optimize a parameter array request, reducing the request size and number of steps, while increasing the number of possible iterations. The best performance is seen for small parameter sets where the array pack factor can be increased.

Using Parameter Arrays

To use arrays of parameters, the application needs to do the following:

1. (Optional, the default is column-wise.) Call SQLSetStmtAttr with an argument of SQL_ATTR_PARAM_BIND_TYPE to specify column-wise or row-wise binding.

   For column-wise binding, the value is SQL_BIND_BY_COLUMN. For row-wise binding, the value is set to the size of() a row in the array holding the parameters.

2. (Optional, default is 1.) Call SQLSetStmtAttr with an argument of SQL_ATTR_PARAMSET_SIZE to specify the number of sets of parameters.

   An array request with only 1 parameter set is equivalent to a “non-array” request.

3. (Required only if SQL_ATTR_PARAMSET_SIZE is greater than 1.) Call SQLSetStmtAttr with an argument of SQL_ATTR_PARAM_STATUS_PTR to point to an array which contains return status information for each set of parameter values.
4. (Optional) Call SQLSetStmtAttr with an argument of SQL_ATTR_PARAM_OPERATION_PTR to point to an array used to exclude or include sets of parameter values.
5. (Optional) Call SQLSetStmtAttr with an argument of SQL_ATTR_PARAMS_PROCESSED_PTR to specify the address of a variable in which the driver can return the number of sets of parameters processed, including error sets.
6. (Optional) Call SQLSetStmtAttr with an argument of SQL_ATTR_PARAM_BIND_OFFSET_PTR to specify the address of a variable containing an integer offset to be added to the ParameterValuePtr and StrLen_or_IndPtr parameters to SQLBindParameter.
7. Call SQLBindParameter for each parameter to bind arrays to parameters.
8. Call one of the execution functions: SQLExecDirect or SQLPrepare/SQLExecute.

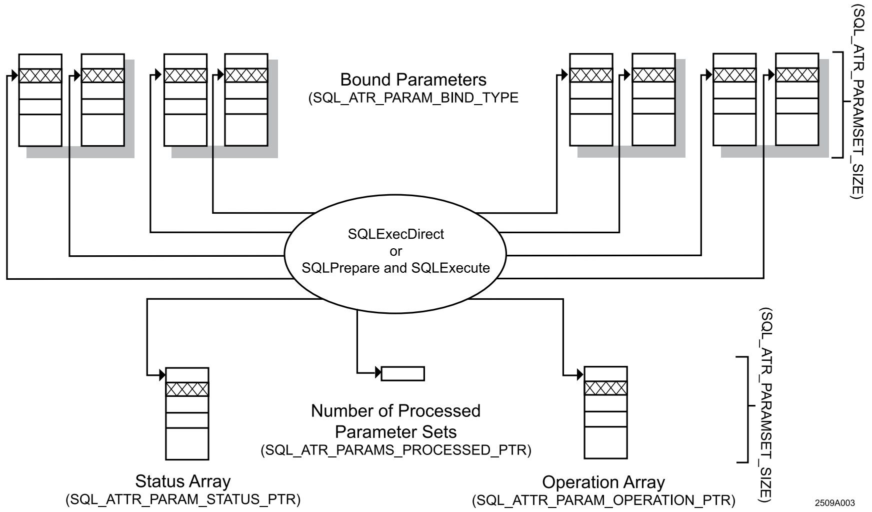
Note:

Only steps 7 and 8 are required, all other steps are optional.

Note:

It is not possible to distinguish an array request with array size 1 from a non-array parameterized request.

The following figure illustrates the use of parameters.



When the statement is executed, ODBC Driver for Teradata uses the information it stored to retrieve the parameter values and send them to the data source.

The array pointed to by the SQL_ATTR_PARAM_OPERATION_PTR statement attribute can be used to ignore rows of parameters. If an element of the array is set to SQL_PARAM_IGNORE, the set of parameters corresponding to that element is excluded from the SQLExecute or SQLExecDirect call. If fetched rows are used as input parameters, the values of the row status array can be used in the parameter operation array.

If the SQL_ATTR_PARAM_STATUS_PTR statement attribute has been set, SQLExecute or SQLExecDirect returns the parameter status array, which provides the status of each set of parameters. The parameter status array is allocated by the application and filled in by the driver. Its elements indicate whether the SQL statement was executed successfully for the row of parameters or whether an error occurred while processing the set of parameters. If an error occurs, the driver sets the corresponding value in the parameter status array to SQL_PARAM_ERROR and returns SQL_ERROR. The application can check the status array to determine which parameter rows were processed. Using the row number, the application can often correct the error and resume processing.

The table that follows lists the types of DML statements with array support information.

DML Statement	Description
ABORT	Only when WHERE clause is present
DELETE	All forms, except “Positioned”
EXECUTE macro	The macro must be a single statement qualified for iteration
INSERT	Includes INSERT … SELECT
LOCKING Modifier	Modified request must be qualified for iteration
MERGE	Similar to UPDATE (Upsert Form)
ROLLBACK	Only when WHERE clause is present (alias of ABORT)
SELECT	Responses returned as in unfolded request
UPDATE (Searched Form)	Includes complex and “unreasonable” updates
UPDATE (Upsert Form)	UPDATE … ELSE INSERT (Atomic UPSERT)

Restrictions

Array Requests Limit

Array requests are limited to 1 MB. This impacts the number of parameter sets that can be applied to an array request. Assuming an SQL request text size of T bytes, a parameter row size of R bytes, and N parameter rows, the limiting conditions are listed in the following table:

Mode	Limiting Conditions
Emulation mode	N * R <= 64 KB and N * (T + R)<= 1 MB
Array support	R <= 64 KB and T + N * R <= 1 MB

Each row is sent in a separate data parcel and is limited to 64 KB. The request text is not replicated. The whole request must fit within the 1 MB request limit. With DML Array Support, the number of rows that can be sent with a parameter array request is increased at least 16 times and up to 31 times for a large row of 33 KB.

No Positional UPDATE/DELETE and SELECT INTO

The positional UPDATE/DELETE and the SELECT INTO statements cannot be used as DML array requests.

No CALL Statements

Any statement used in an iterated request must be usable in a multi-statement request. Because CALL statements are not permitted in multi-statement requests, CALL statements and so on are not supported with the new DML array support feature or the old emulated mode.

No DML Array Requests with Triggers

The DML array feature does not support tables with triggers. As a work around, the request can be turned into a multi-statement request by appending a no-operation SQL statement. This causes ODBC Driver for Teradata to emulate the array request as a series of SQL requests that work on tables with triggers, but without the performance benefit of the DML array request.

Error Handling and Transaction Semantics

In the database, transaction commit, as well as request-level and transaction-level abort for an array request matches the equivalent multi-statement request. Transaction semantics are the same for emulated and non-emulated parameter array requests. The entire parameter array request is rolled back, including those parameter rows already processed.

Errors are handled exactly the same as they would be for the equivalent multi-statement request. The error information returned from the database to the client in the resulting failure parcel matches that from the unfolded request, and the parameter set that generated the failure is identified by the statement number in the failure parcel.

If an error occurs while executing a parameter array statement, the execution function returns an error and sets the row number variable to the number of the row containing the error. It is data source-specific whether all rows except the error set execute or all rows before (but not after) the error set execute. In case of an error, ODBC Driver for Teradata always fails the whole parameter array request.

If an error occurs, the statement number in the failure parcel is used to derive the row number of the failing parameter set and to set the corresponding SQL_DIAG_ROW_NUMBER field of the diagnostic record.

The SQL_DIAG_COLUMN_NUMBER field of the diagnostic record is set by the driver to SQL_COLUMN_NUMBER_UNKNOWN because the driver cannot determine the parameter number for the failure.

The buffer specified by the SQL_ATTR_PARAMS_PROCESSED_PTR statement attribute is set to the number of the failed parameter set.

Requests That Do Not Generate Result-Sets

For requests that do not generate result-sets, when errors are encountered, ODBC Driver for Teradata sets the status value in the status array to SQL_PARAM_ERROR for the parameter set that failed and all the other elements are set to SQL_PARAM_DIAG_UNAVAILBLE, indicating that status is not available. This is because the request can execute in parallel and there is no way to determine which parameter sets were successfully executed before the failure occurred and the request was aborted.

The application can identify the failed parameter set and correct it if possible or resubmit the request with the failed set by marking it as SQL_PARAM_IGNORE in the operations array. The latter is more efficient than just submitting each parameter set one at time upon failures.

Result-Set Generating Requests

For result-set generating requests, the availability of status array values is driver-defined. Status array values might be available after the statement has been executed or as result sets are fetched. ODBC Driver for Teradata provides the status as results are fetched. For example, the initial status for a given parameter set is undefined until the results corresponding to the parameter set have been fetched.

New Parser

A new parser has been introduced in the ODBC Driver for Teradata 16.20 which permits the database to handle more of the parsing. The Legacy Parser option is deprecated in ODBC 16.20.

Large Decimal and BIGINT Support

ODBC Driver for Teradata supports the use of DECIMAL data types with a precision of 38. An application can determine the maximum precision by calling the ODBC API function SQLGetTypeInfo.

ODBC Driver for Teradata supports 19-digit precision for BIGINT data. The BIGINT data type has an exact numeric value with precision 19 and scale 0. The BIGINT is signed and has a range between $-2^{63} \leq n \leq 2^{63} - 1$ or from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. It includes most (but not all) values with a decimal precision of 19.

64-bit Support

The following reference is useful when working with ODBC on 64-bit systems:

- INFO: ODBC 64-Bit API Changes in MDAC 2.7, Article ID: 298678

  http://support.microsoft.com/default.aspx?scid=kb;en-us;298678
- ODBC header files, specifically `sql.h`.
- README file delivered with ODBC Driver for Teradata. This file contains information about compilation and linking with ODBC Driver for Teradata on 64-bit systems. The file is located in the <*InstallDir*> on UNIX systems and available in the **All Programs > ODBC Driver for Teradata** menu on Windows systems. On Apple OS X, the file is located at:

  `/Library/Application Support/teradata/client/<TTU version>/odbc/README`.

When compiling 64-bit applications for ODBC Driver for Teradata on the Linux/UNIX system, it is important to use -DODBC64 to enable the 64-bit definitions in the Driver Manager include files.

Extended Object Names (EON)

Object names include User, Database, Table, View, Macro, Column names, and many more. Teradata Database 14.10 extends object names up to 128 characters and contains an extended variety of characters, which includes almost all Unicode characters supported by Teradata database, except pass-through characters. Additionally, object names may be represented using the Unicode delimited identifier notation in different contexts.

Note:

> User name, password, account string, and default database names can be used with both the extended length and the extended set of characters in the connection string. The configuration dialogs and the `odbc.ini` settings support the extended length and the same character set as earlier releases of the ODBC driver.

SQLGetInfo

The following table lists info items in SQLGetInfo and reflect the extended object names:

Info Item	Pre-14.10 Versions	Version 14.10 and Later
SQL_MAX_COLUMN_NAME_LEN	30 Characters	128 Characters
SQL_MAX_SCHEMA_NAME_LEN	30 Characters	128 Characters
SQL_MAX_TABLE_NAME_LEN	30 Characters	128 Characters
SQL_MAX_USER_NAME_LEN	30 Characters	128 Characters
SQL_MAX_CURSOR_NAME_LEN	30 Characters	128 Characters
SQL_MAX_PROCEDURE_NAME_LEN	30 Characters	128 Characters
SQL_MAX_IDENTIFIER_LEN	30 Characters	128 Characters

7

Network Security

Overview

This chapter provides information about selecting and enabling network security.

This chapter is divided into the following sections:

Section Heading	Contents
Password Encryption	Software encryption of the logon password
Single Sign-On (Windows and Apple OS X)	How to set up ODBC Driver for Teradata to be able to log on to a computer/workstation once
Data Encryption	Encryption of data transmitted between ODBC Driver for Teradata and Teradata gateway
Extensible Authentication, Authorization, and Encryption	Details about the security enabled in Teradata systems
Authentication Mechanisms	Information about the authentication available
Determination of Authentication Mechanism	How the authentication mechanisms are selected
Configuring Authentication Mechanisms	Where to find information to configure your system
Connecting to Teradata Database	SQLConnect(), SQLDriverConnect(), and SQLBrowseConnect()
Enhancing Security	Recommended actions for enhancing system security
Constraints	Security constraints with this version of the software
Teradata Wallet	How to use Teradata Wallet with ODBC Driver

Password Encryption

Logon encryption is used automatically if the server for the application supports the feature. This is not a user-defined setting at the client level, but the feature can be set as a gateway option using the GTW control utility.

For more information, see the Teradata Database documentation.

Single Sign-On (Windows and Apple OS X)

SSO permits a user to log on to a computer or workstation one time and thereafter access Teradata Database without repeatedly entering a username and password. This feature saves time and adds an additional level of security, as the username and password are entered once, and a modern authentication mechanism such as Kerberos sends passwords across the network.

Enabling SSO

SSO is enabled by checking the Integrated Security option while configuring a Teradata DSN. If the Integrated Security option is not checked, then Conventional Sign-On (CSO) is enabled.

SSO can also be enabled at runtime by adding the ***`UseIntegratedSecurity = Y`*** in the connection string used by the connection APIs. The runtime setting overrides this option during DSN configuration time.

SSO Usage

When performing SSO, ODBC Driver for Teradata performs reverse DNS lookup and must succeed. The reverse lookup result must be the FQDN of the Teradata Database node.

Note:

Third Party Sign-Ons are not supported.

There are two SSO scenarios:

- Direct Sign-On: If SSO is enabled and if username and password are *not* specified in the ODBC Connection APIs (SQLConnect, SQLBrowseConnect, SQLDriverConnect), then a Direct Sign-On is considered.
- Third-Party Sign-On: If SSO is enabled and if username and password are specified in the ODBC Connection APIs (SQLConnect, SQLBrowseConnect, SQLDriverConnect), then a Third-Party Sign-On is considered.

SSO Error Messages

The following table lists the error messages that ODBC Driver for Teradata can return regarding SSO usage:

Error Message	Description
SSO request cannot be honored – SSO not available	An SSO request is received, and the client, the gateway, or Teradata Database does not support SSO
Couldn't acquire memory on behalf of caller	Terasso DLL was unable to acquire needed memory
Problem in passed parameter(s)	Terasso DLL detected error(s) in the passed parameter(s)
Couldn't load security DLL	Terasso DLL was unable to load the security DLL
Couldn't find security entry point	Terasso DLL was unable to find a security entry point in the security DLL
Couldn't initialize security DLL	Terasso DLL could not initialize the security DLL
SSPI call to EnumerateSecurityPackages failed	Terasso DLL was unable to enumerate the available security packages using SSPI. The reason code (SSPI return code) is included.
Couldn't find a compatible package match between client and gateway	Terasso DLL was unable to find a security package compatible with both client and gateway
SSPI call to AcquireCredentialsHandle failed	Terasso DLL was unable to acquire a credentials handle using SSPI. The reason code (SSPI return code) is included.
SSPI call to InitializeSecurityContext failed	Terasso DLL was unable to initialize the security context using SSPI. The reason code (SSPI return code) is included.
SSPI call to CompleteAuthToken failed	Terasso DLL was unable to complete the authorization token using SSPI. The reason code (SSPI return code) is included.
SSPI call to FreeBufferContext failed	Terasso DLL was unable to free the buffer context using SSPI. The reason code (SSPI return code) is included.
Terasso DLL is not in the path	Terasso DLL is not present in the system32 directory

Data Encryption

Data encryption provides secure transmission of data between Teradata clients and the Teradata gateway. Data encryption is implemented in ODBC Driver for Teradata to address the security considerations for data transmission between Teradata ODBC clients and the Teradata gateway. Therefore, Teradata ODBC clients can perform encrypted communication to and from the Teradata gateway.

ODBC applications can control the encryption of data transmitted between ODBC Driver for Teradata and the Teradata Gateway. This can be controlled at the connection level or the statement level.

Connection Level

This connection level setting is applicable to the particular connection handle and, by default, all statement handles created under it. This option cannot be set at connection level after the connection to the database has been established.

DSN Option

- On DSN Setup GUI dialog, checking **Enable Data Encryption** turns on data encryption. By default, this option is unchecked.
- In `odbc.ini` the option ***`UseDataEncryption=YES/NO`*** controls this data encryption in ODBC driver. The default is NO.

SQLDriver/BrowseConnect() Keyword

The connection string keyword-value pair ***`DATAENCRYPTION=YES/NO`*** or ***`USEDATAENCRYPTION=YES/NO`*** can be used in the connection string of SQLDriver/ BrowseConnect(), to turn data encryption ON or OFF.

Statement Level

Data encryption can also be turned ON or OFF at the statement level. The data encryption behavior set at the statement level is applicable only for the particular statement handle, and transactions done for that statement handle only.

At the statement level, data encryption can be turned ON/OFF by setting ODBC Driver for Teradata defined attribute SQL_ATTR_DATA_ENCRYPTION(13008) to:

- SQL_DATA_ENCRYPTION_ON(1) or
- SQL_DATA_ENCRYPTION_OFF(0)

This statement-level attribute overrides the connection-level data encryption option. For example, encryption can be turned ON or OFF with this attribute for a particular statement handle irrespective of the connection level setting. All other statement handles under the same connection handle use data encryption as per the connection level setting as the default.

Backward Compatibility

Existing applications or an application using an existing DSN can have no impact, since the default for this option is OFF.

Gateway Dependency

The matrix below describes the database dependencies.

If	and if ODBC Data Encryption=ON	and if ODBC Data Encryption=OFF
Teradata gateway supports encryption	All communication between ODBC Driver for Teradata and the Teradata gateway will be encrypted	All communication between ODBC Driver for Teradata and the Teradata gateway will be in plain text, except during logon
Teradata gateway does *not* support encryption	The connection will not be established	All communication between ODBC Driver for Teradata and the Teradata gateway will be in plain text

Performance Considerations

There will be some impact in overall performance if encryption is turned ON, since processing is required on both the client and database for data encryption and decryption.

The statement-level option is provided to allow applications to behave “smart” and turn ON encryption only when required. Therefore, applications that can behave “smart,” (decide which request and response must be encrypted) should rely on the dynamic statement-level attribute.

Extensible Authentication, Authorization, and Encryption

Authentication

Authentication is the indisputable establishment of identities between two mutually suspicious parties when faced with adversaries with a malicious intent. In other words, authentication answers a very simple question: *Who are you?*

In research literature, authentication is defined as a proof of authenticity; it determines if the source of a message is genuine. Authentication says nothing about capabilities; that is, it does not determine if a source has the right to access certain resources within the destination.

User authentication in Teradata Database using ODBC Driver for Teradata uses the following mechanisms:

- **Conventional Teradata Mechanism** – consists of a username and a password, which are validated by the database during logon. This is sometimes referred to as CSO.
- **SSO** – the system is trusted, and the user is logged on without providing a username and password. The user's identity, which is obtained through a network or domain login, is transmitted to Teradata and verified.

  SSO is only supported when the server is running Windows and the client running a Windows or Apple OS X version supporting SSPI. SSO is supported using Kerberos as the authentication mechanism.

The Extensible User Authentication feature expands these authentication mechanisms to include LDAP, Teradata-defined, and other user-defined mechanisms. It also provides support for Kerberos on all platforms that support Kerberos.

Authorization

Authorization consists of providing access to resources based on authenticated user's privileges within the system. This is controlled by the gateway authorizing access to the database server, and by the users' privileges specific to the database (GRANT/REVOKE).

Authorization is outside the scope of ODBC Driver for Teradata.

Confidentiality

Confidentiality (or protection) is related to the network providing a secure transport of information between the database and ODBC Driver for Teradata. This is accomplished by encrypting the network information, such as logon and data encryption. The Extensible User Authentication feature allows different confidentiality services, such as encryption and decryption to be associated with the authentication mechanisms.

Integrity

Integrity ensures that information is not tampered with or otherwise altered without proper authorization while in transit between the client and the gateway. The Extensible User Authentication feature allows different integrity services–such as message integrity code calculation and verification–to be associated with the authentication mechanisms.

Authentication Mechanisms

To enable network security, an Authentication Mechanism that corresponds to one of the mechanisms configured on the gateway by a system administrator must be specified, and other parameters such as password and username must be passed from the client application through network security. The following table summarizes the authentication mechanisms.

Authentication Mechanism	Name	Client/ Servers	Description
Teradata 2	TD2	All	The Teradata 2 (TD2) mechanism provides authentication using a Teradata Database username and password. The difference between Teradata 1 and Teradata 2 is that the Teradata 2 encryption key offers a higher degree of security. **Encryption**: When the Teradata 2 (TD2) mechanism is selected, both logon string and data are encrypted. **Availability**: The Teradata 2 mechanism is available on all supported client and server platforms. **Username and Password**: A valid Teradata username and password are always required.
TDNEGO	TDNEGO	All	A security mechanism that automatically determines the actual mechanism required, based on policy, without user's involvement. The actual mechanism is determined by the TDGSS server configuration and by the security policy's mechanism restrictions. This simplifies user logons, because the user does not need to specify which logon mechanism to

Authentication Mechanism	Name	Client/ Servers	Description
			use. It also provides better ease of use and improved support for applications and tools which do not support specification of logon mechanisms. The Client and Server versions of TDNEGO automatically negotiate and select an appropriate TDGSS security mechanism to use. **Encryption**: Depends based on the chosen mechanism. **Availability**: The TDNEGO mechanism is available on all supported client and server platforms. **Username and Password**: Depends based on the chosen mechanism.
LDAP	LDAP	All w/ LDAPv3 Library	When the LDAP authentication mechanism is employed, the server authenticates by binding to the LDAP directory using the username and password. **Encryption**: When LDAP is selected, both the logon string and data are encrypted. **Availability**: The LDAP mechanism is available on all supported client and server platforms, which provide an LDAPv3 compliant library. **Username and Password**: The application supplies a username, password, and domain or realm. When the user has been authenticated, an implicit logon will proceed using a Teradata username derived from the directory. The gateway directory maps the username to a specific Teradata username or to the system-defined username EXTUSER.

Authentication Mechanism	Name	Client/ Servers	Description
			If the directory maps the username to a specific Teradata username, then that user must have previously been granted the logon with null password privilege. If the directory maps to EXTUSER, then the characteristics of the user (role, rights, space, and so forth) are determined from settings in the directory.
Kerberos	KRB5	Windows Linux Apple OS X AIX Solaris OP Solaris SP	Once the identity of the user has been verified by Kerberos (KRB5), the KRB5 mechanism implicit logon proceeds using the same username as the Teradata username. **Encryption**: When KRB5 is selected, both the logon string and data are encrypted. **Availability**: The KRB5 mechanism is available on all supported client and server platforms. **Username, Password, Domain and Realm**: The application supplies a username, password, and domain or realm. The username must have previously been granted the logon with null password privilege. **Single Sign On**: The Kerberos (KRB5) mechanism supports SSO where no username and password are provided explicitly by the application, but both are derived from the security context of the application. For KRB5 authentication, ODBC Driver for Teradata performs reverse DNS lookup and must succeed. The reverse lookup result must be the FQDN of the Teradata Database node.
JSON Web Token	JWT	All	The client’s credentials (username and password) is authenticated by UDA User Service. The UDA User Service returns a JSON Web Token containing encrypted

Authentication Mechanism	Name	Client/ Servers	Description
			credentials. The client uses this token to connect to the database. The client needs to provide the following two parameters to the ODBC Driver in the connection string when using JWT authentication mechanism: `AUTHENTICATION=JWT; AuthenticationParameter={token=<JWT token>};` where *`<JWT token>`* is the token obtained from the UDA User Service. **Encryption**: When JWT is selected, both the logon string and data are encrypted. **Availability**: The JWT mechanism is available on all supported client and server platforms. **Username, Password, Domain and Realm**: The username and password are encrypted in the JWT token. **Single Sign On**: The JWT mechanism supports SSO as long as the JWT token is not expired.
Other	To be determined	All	Users can define other authentication mechanisms. **Encryption**: A user-defined mechanism can also provide logon and data encryption. **Username and Password**: Input to a mechanism will be the username, password, and possibly authentication information specific for the particular mechanism.

Determination of Authentication Mechanism

ODBC Driver for Teradata allows specification of the authentication mechanism in the connection string and in the ODBC data source configuration.

The authentication mechanism for a connection is determined as follows:

- If an authentication mechanism is specified in the connection string passed to SQLDriverConnect() or SQLBrowseConnect(), then that authentication mechanism is selected. The SQLBrowseConnect() can return a set of authentication mechanisms supported by both the client and the database to allow the application to make a choice.
- If no mechanism is specified in the connection string or if connecting using SQLConnect(), then the driver looks in the ODBC data source configuration. If an authentication mechanism is specified in the DSN settings–in the [Named Data Source] section or the [Default Data Source] section–then that mechanism is selected. The specification in the [Named Data Source] section takes precedence over any specification in the [Default Data Source] section.
- If no mechanism is specified in the connection string or DSN settings, then the selected authentication mechanism is the authentication mechanism configured for TeraGSS on the client, provided that it is supported by the gateway.
- Otherwise, the selected mechanism is the default mechanism configured for TDGSS on the gateway, provided that it is supported by the client.
- Otherwise, the selected mechanism is Teradata 2 if the gateway supports it, or else it is Teradata 1 if the gateway supports it, or else it is the conventional Teradata mechanism.
- If the selected authentication mechanism is not supported by the client and the gateway, then the connection function call (SQLConnect(), SQLDriverConnect(), or SQLBrowseConnect()) fails and returns SQL_ERROR.

The USEINTEGRATEDSECURITY Connection String Attribute

The USEINTEGRATEDSECURITY or INTEGRATEDSECURITY connection string attribute and the corresponding setting in the ODBC data source configuration is available for Windows, Apple OS X, and UNIX OS clients. In Teradata V2R6.0 and later, there is no authentication method supporting SSO on a UNIX system.

If USEINTEGRATEDSECURITY or INTEGRATEDSECURITY is set and the selected authentication mechanism does not support SSO, then the connection function call fails.

Configuring Authentication Mechanisms

Authentication Mechanisms can be configured by:

- configuring the `odbc.ini` file (see Teradata DSN Options.)
- using the **ODBC Driver Setup for Teradata** dialog box (see Configuring a Data Source.)

Connecting to Teradata Database

Information on how to connect to Teradata Database using SQLConnect(), SQLDriverConnect(), and SQLBrowseConnect() selecting the Authentication Mechanisms can be found in the section ODBC Connection Functions and Dialog.

Username and Password Requirements

The following table shows the Authentication Mechanisms that require and don't require usernames and passwords.

Authentication Mechanism	Mechanism Parameter	Logon Encryption	Data Encryption	Username	Password
TD2	Never	Yes	Yes	Must	Must
KRB5	Optional	Yes	Yes	Never	Never
LDAP	Must	Yes	Yes	Never	Never
JWT	Must	Yes	Yes	Never	Never

Enhancing Security

Recommendations for enhancing security in an ODBC Driver for Teradata environment include:

- When using integrated security (SSO) applications, do not pass usernames and passwords across the wire
- An application should programmatically prevent ODBC tracing from displaying sensitive information, such as function call parameters
- System administrator can disable ODBC tracing

Constraints

Disconnect After Security Context Expiration

Some security mechanisms, such as Kerberos, provide for the expiration of security contexts. For example, a Kerberos ticket will expire. If the gateway or the client detects such an expiration during encryption or decryption, it immediately closes the virtual circuit.

The security context expiration and associated disconnect is limited to Windows clients and servers using Kerberos authentication.

If reconnect is enabled in the DSN settings, then the driver attempts to reconnect by presenting the same credentials that were used to obtain the original connection. The driver will not try to recover any pending request or provide a status of the request to the application.

Disconnects might also be caused by security context expiration. The advice to users is to try to avoid security context expiration by ensuring that the lifetime of the context is sufficient to allow a given request to complete.

Connection Pooling (Windows and Apple OS X)

Be aware the authentication method and authentication string are not among the criteria for matching connections in the connection pool, and if they contain information that further qualifies the authentication information, then incorrect matches can occur.

One such case is in an application server scenario, if the username and password are identical, but the authentication mechanisms differ, one user can get another's connection, although this is very unlikely to happen.

TDGSS Support for UTF16

Authentication Parameter

The values for an authentication parameter can be supplied in the following locations:

- DSN
- Connection string
- Dialog box

These values should be in the character set specified by the Windows application code page setting on Windows, the ODBC application code page setting on a UNIX system, or the locale setting (LC_TYPE) on Apple OS X.

Teradata Wallet

Teradata Wallet provides access to a client system user’s stored Teradata Database passwords to the user while protecting these passwords from access by other users of the same client system.

Each entry in wallet has two parts, a reference string and value (reference string) and real password. Teradata Wallet returns the password when queried with its associated string to the user who created it. For detailed information on using Teradata Wallet, see the most recent version of *Security Administration* (B035-1100).

ODBC connect functions can process Teradata Wallet reference strings when used in/as password or authentication parameter. Reference string must be enclosed in `$tdwallet()` token.

Saving a Wallet String in DSN:

- On DSN Setup dialog, enter the Wallet reference string in **Teradata Wallet String** field.
- In `odbc.ini`, enclose the Wallet reference string in `$tdwallet()` token and use it as password under the DSN in `odbc.ini`.

Using Wallet String in connect functions:

A wallet reference string can be used in place (or part of) password or authentication parameter.

Examples

For a connect function call containing the Teradata Wallet reference string:

- `SQLConnect(hdbc, "mydsn", SQL_NTS, "myuid", SQL_NTS,"$tdwallet(RefString)", SQL_NTS);`
- `SQLDriverConnect(hdbc, NULL, "DSN=mydsn;UID=myid;PWD=$tdwallet(RefString);", SQL_NTS, szConnStrout, cbConnStrOutMax, &cbConnectStrOutLen, NULL);`
- `SQLDriverConnect(hdbc, NULL, "DRIVER={Teradata}; DBCNAME=platinum; AUTHENTICATION=LDAP;AUTHENTICATIONPARAMETER=authcid=$tdwallet(RefString1) password=$tdwallet(RefString2); ", SQL_NTS, szConnStrout, cbConnStrOutMax, &cbConnectStrOutLen, NULL);`

Password Expiration and Teradata Wallet

On Windows systems, if Teradata Database user password expires, Teradata Database returns an error message. ODBC driver detects the error and prompts for a new password (if Quiet Mode is OFF). At this point, the user is not allowed to enter another Teradata Wallet string, but must type a new password for the actual database user.

Similarly, the value for `PWD2` in a connection-string cannot be a Teradata Wallet string.

8

ODBC Driver for Teradata Application Development

Overview

This chapter provides information needed to use ODBC Driver for Teradata to access Teradata features and extensions to the ODBC standard.

This chapter is divided into the following sections:

Section Heading	Contents
Teradata Extensions to the ODBC Standard	Extensions above and beyond ODBC that ODBC Driver for Teradata supports for Teradata Database
Stored Procedures	A set of control and condition handling statements that provide a procedural interface to Teradata Database
Auto-Generated Key Retrieval	Information about using the auto-generated key configuration option
SQL Descriptor Fields	Information on how ODBC Driver for Teradata uses the extended statement information configuration option
International Character Set Support	How to configure the driver for character set support
Atomic UPSERT	Support for UPSERT in ODBC Driver for Teradata
ANSI Date and Time Restrictions	Restrictions that occur for ANSI Date/Time
Period Data Types	Information about using Teradata Period data types
Geospatial Types	Information about using geospatial and GeoSequence types
Restrictions	Current restrictions for ODBC Driver for Teradata
ANSI Migration Issues	Migration issues, including: transaction semantics, data truncation, duplicate rows, updatable cursors, and case sensitivity
Configuration Characteristics	Access locks and password expiration
SQL Considerations	SQL compatibility issues

Section Heading	Contents
DSN Settings for Third-Party Applications	Lists third-party DSN settings and their specific application

Teradata Extensions to the ODBC Standard

This section describes extensions above and beyond ODBC that ODBC Driver for Teradata supports for Teradata Database. The symbolic names associated with the extensions are defined in the `tdsql.h` file in *`<InstallDir>`*/include.

Note:

> When using driver-defined attributes it is necessary to indicate the type of attribute to the Driver Manager so it can properly pass the value to or from the driver. This is typically done by setting the `BufferLength` argument ODBC API function call getting or setting the value. For example, `BufferLength` should be `SQL_IS_SMALLINT` for a Teradata ODBC column attribute like `SQL_DESC_TD_ODBC_TYPE`.

Connection Attributes

The following table lists the connection attributes and their associated values.

Connection Attributes	ValuePtr Contents
SQL_ATTR_TDATA_HOST_ID	An SQLUINTEGER value in which the logical host ID for the session is returned.
SQL_ATTR_TDATA_SESSION_NUMBER	An SQLUINTEGER value in which the logical session number is returned.
SQL_ATTR_TDATA_SESSION_CHARSET	A null-terminated character string containing the name of the session character set is returned.
SQL_ATTR_AGKR	An SQLUINTEGER value that determines the result from requests that insert into identity columns (INSERT, INSERT … SELECT, UPSERT, MERGE-INTO). These requests can optionally return a result set containing identity column values (also known as auto-generated keys) for the inserted rows.

Connection Attributes	ValuePtr Contents
	Values supported are as follows: • SQL_AGKR_NO • SQL_AGKR_IDENTITY_COLUMN • SQL_AGKR_WHOLE_ROW The acronym AGKR is defined as Auto-Generated Key Retrieval. This attribute can be set to SQL_AGKR_NO(0), SQL_AGKR_IDENTITY_COLUMN(1), or SQL_AGKR_WHOLE_ROW(2), meaning respectively that no keys are retrieved, only the identity column is retrieved, or the whole row is retrieved by ODBC Driver for Teradata after an insertion into a table containing an identity column. An error is returned if the application tries to enable the auto-generated key retrieval and the database does not support the feature. The error returned is SQL_ERROR with SQLSTATE HY024 and the message is: Invalid attribute value.

Statement Attributes

The following table lists the statement attributes and their associated values.

Statement Attribute	ValuePtr Contents
SQL_ATTR_AGKR	An SQLUINTEGER value that determines the result from requests that insert into identity columns (INSERT, INSERT … SELECT, UPSERT, MERGE-INTO). These requests can optionally return a result set containing identity column values (also known as auto-generated keys) for the inserted rows. Values supported are as follows: • SQL_AGKR_NO • SQL_AGKR_IDENTITY_COLUMN

Statement Attribute	ValuePtr Contents
	- SQL_AGKR_WHOLE_ROW This statement level attribute inherits and overrides the connection level SQL_ATTR_AGKR setting.
SQL_ATTR_DATA_ENCRYPTION	This SQLINTEGER statement attribute turns ON/OFF data encryption at statement level. All the transactions done under a statement handle are affected. Values supported are as follows: - SQL_DATA_ENCRYPTION_ON - SQL_DATA_ENCRYPTION_OFF This statement-level attribute overrides the connection-level data encryption option. For example, encryption can be turned ON or OFF with this attribute for a particular statement handle regardless of the connection-level setting. All other statement handles under the same connection handle use data encryption as per the default connection level setting.
SQL_ATTR_TDATAODBC_SBU_ROWCOUNT	This attribute is an unsigned bigint (SQLUBIGINT) and lets 32-bit applications obtain the row count even if it is greater than the maximal signed integer (0x7FFFFFFF).
SQL_ATTR_TRUSTED_SQL	SQL_ATTR_TRUSTED_SQL is an SQLUINTEGER value. Values supported are as follows: - SQL_TRUE - SQL_FALSE SQL_ATTR_TRUSTED_SQL attribute can be used to specify if the next SQL that is executed(either through SQLExecute or SQLExecDirect) is trusted or not trusted.

Statement Attribute	ValuePtr Contents
	Note that the value of SQL_ATTR_TRUSTED_SQL is SQL_FALSE by default, and it will be reset back to SQL_FALSE after any SQLExecute or SQLExecDirect is made. The SQL_ATTR_TRUSTED_SQL attribute is a statement attribute. When calling SQLSetStmtAttr() or SQLGetStmtAttr() you must pass in the value SQL_IS_UINTEGER for the StringLength or BufferLength argument respectively. See also the section on Trusted Sessions .

SQL Column Attributes

`SQLColAttribute` supports several Teradata ODBC driver-defined column attributes as the `FieldIdentifier` argument. Additionally, using `SQL_COLUMN_NAME` returns the Teradata *`TITLE`* (if available) instead of the name because this makes most report writers work more sensibly.

When using SQLColAttribute to obtain driver-defined attribute values the type of the attribute must be specified in the `BufferLength` argument. In addition, `SQLColAttribute` returns the value through one of two pointers depending on the type: Integer information is returned in `*NumericAttributePtr` as a `SQLLEN` value; all other formats of information are returned in `*CharacterAttributePtr`. The pointer not being used for a particular attribute should be set to `NULL` in the call. For example, to obtain the value of the attribute `SQL_DESC_TD_ODBC_TYPE`:

```
rc = SQLColAttribute(hstmt, colno, SQL_DESC_TD_ODBC_TYPE, NULL, SQL_IS_SMALLINT,
NULL, &NumAttrPtr);
```

The pointer used for each of the attributes defined by the Teradata ODBC driver is indicated in the list below.

Driver-defined FieldIdentifier values are:

- SQL_COLUMN_ACTIVITY_TYPE is an integer that specifies the kind of SQL statement executed. Value is returned in *`NumericAttributePtr`*.

- SQL_COLUMN_COST_ESTIMATE is an integer with a cost estimate for running the SQL statement. The value returned represents the time estimate in seconds. Value is returned in ***NumericAttributePtr***.
- SQL_COLUMN_FORMAT returns the Teradata FORMAT clause associated with the column. Value is returned in ***CharacterAttributePtr***.
- SQL_COLUMN_ACTUAL_NAME is the name associated with the result column. This FieldIdentifier value is necessary because the meaning of SQL_COLUMN_NAME has changed. Value is returned in ***CharacterAttributePtr***.
- SQL_COLUMN_CHARACTER_SET is an integer containing the character set of the column. Value is returned in ***NumericAttributePtr***.
- SQL_COLUMN_EXPORT_WIDTH is an integer containing the database export width for character columns. Value is returned in ***NumericAttributePtr***.
- SQL_COLUMN_EXPORT_WIDTH_ADJ is an integer containing the database export width adjustment for character columns. Value is returned in ***NumericAttributePtr***.
- SQL_COLUMN_EXPORT_BYTES is an integer containing the number of bytes that the database provides for a character column. Value is returned in ***NumericAttributePtr***.
- SQL_DESC_TD_ODBC_TYPE is an SQLSMALLINT that contains the Teradata ODBC-specific SQL data type code. If multiple database types have the same standard ODBC SQL type, then the Teradata ODBC-specific SQL data type code can be used to distinguish between the types. Value is returned in ***NumericAttributePtr***.

For additional information about these attributes, see SQL Descriptor Fields.

When Making SQLColAttribute Calls

For an ANSI application (that is, compiled without UNICODE defined), the Driver Manager on the UNIX OS will not convert:

- SQLColAttribute calls into SQLColAttributeW calls in ODBC Driver for Teradata
- Output parameters from UTF-8 back to the application code page

Because of this, the output parameters from SQLColAttribute are delivered back to the ANSI application in the internal character set used by the driver. If the internal character set is different from the application code page, the application receives data back from SQLColAttribute in a different character set from what was expected.

This is a problem if, for example, an ANSI application using ISO 8859-1 requests non-ASCII meta data (such as a column name with Danish characters) and the session character set is UTF-8. The application gets the column name back in UTF-8. In general, if an ANSI application uses a Unicode session character set, it gets data back from SQLColAttribute in UTF-8, regardless of the application code page.

To avoid this problem, use the old SQLColAttributes function (with an 's' at the end).

SQL Descriptor Fields

The following descriptor fields are exposed by ODBC Driver for Teradata:

- SQL_DESC_TD_ACTIVITY_TYPE is an SQLINTEGER that indicates the type of SQL statement that was executed.
- SQL_DESC_TD_COST_ESTIMATE is an SQLINTEGER with a cost estimate for running the SQL statement. The value returned is the time estimate in seconds.
- SQL_DESC_TD_FORMAT is a character string which is the Teradata FORMAT column.
- SQL_DESC_TD_ACTUAL_NAME is a character string containing the column name associated with the result column.
- SQL_DESC_TD_CHARACTER_SET is an SQLINTEGER that contains the character set of a given column. Values are:
    - SQL_TD_CS_UNDEFINED
    - SQL_TD_CS_LATIN
    - SQL_TD_CS_UNICODE
    - SQL_TD_CS_KANJISJIS
    - SQL_TD_CS_GRAPHIC
    - SQL_TD_CS_KANJI1

      If both the Extended Statement Information and LOB support options are disabled, the ODBC driver is unable to determine the server character set, resulting in the return value SQL_TD_CS_UNDEFINED.
- SQL_DESC_TD_EXPORT_WIDTH is an SQLINTEGER that contains the database export width for character columns, while SQL_DESC_TD_EXPORT_WIDTH_ADJ is an SQLINTEGER that contains the database export width adjustment for character columns.

  These fields are only valid if the column character set is different from SQL_TD_CS_UNDEFINED; otherwise they have the value -1, which indicates an undefined value.

  The fields can be used to calculate the number of bytes exported by Teradata Database for a given character column. The formula is:

  ```
  <Number of bytes > = <export width * N > + <export width adjustment>
  ```

  where N is the number of characters in the column.
- SQL_DESC_TD_EXPORT_BYTES is an SQLINTEGER that contains the number of bytes the database provides for a character column.
- SQL_DESC_TD_ODBC_TYPE is an SQLSMALLINT that contains the Teradata ODBC-specific SQL data type code. If multiple database types have the same standard ODBC SQL type, then the Teradata ODBC-specific SQL data type code can be used to distinguish between the types. For Example, see Number Data Types.

SQLGetTypeInfo

The result set returned by SQLGetTypeInfo has an additional Teradata ODBC-specific column after the standard columns. The specific column information is listed in the table below.

Column Name	Column Number (ODBC 3.X)	Data Type	Comments
USER_DATA_TYPE	20	Smallint	Always NULL
TDODBC_DATA_TYPE	21	Smallint not NULL	Teradata ODBC-specific SQL data type code used by the ODBC driver. If multiple database types have the same standard ODBC SQL type, then the Teradata ODBC-specific SQL data type code can be used to distinguish between the types. For an example, see Number Data Types.

SQLColumns

SQLColumns has additional Teradata-specific columns in the result set as listed in the following table.

Column Name	Column Number (ODBC 3.X)	Data Type	Comments
LABEL	19	Varchar (256) See (1) at end of table	Returns the column label (Teradata title) if one is provided for the column or null.
FORMAT	20	Varchar(30)	Returns the Teradata format for a column or null.
CHAR_TYPE	21	Varchar(30)	Returns a value describing character type.

Column Name	Column Number (ODBC 3.X)	Data Type	Comments
			Examples include: LATIN, KANJI1, UNICODE, and so on
TDODBC_DATA_TYPE	22	Smallint not NULL	Teradata ODBC-specific SQL data type code used by the ODBC driver. If multiple database types have the same standard ODBC SQL type, then the Teradata ODBC-specific SQL data type code can be used to distinguish between the types. For an example, see Number Data Types.
(1) Varchar(256) if EON enabled, otherwise Varchar(60).			

SQLProcedureColumns

The result set returned by SQLProcedureColumns has an additional Teradata ODBC-specific column after the standard columns. This information is listed in the following table.

Column Name	Column Number (ODBC 3.X)	Data Type	Comments
IS_RESULT_SET_COLUMN	20	Smallint	Always NULL
USER_DATA_TYPE	21	Smallint	Always NULL
TDODBC_ DATA_TYPE	22	Smallint not NULL	Teradata ODBC-specific SQL data type code used by the ODBC driver. If multiple database types have the same standard ODBC SQL type, then the Teradata ODBC-specific SQL data type code can be used to distinguish between the

Column Name	Column Number (ODBC 3.X)	Data Type	Comments
			types. For an example, see Number Data Types.

Teradata ODBC Driver SQL Types

ODBC Driver for Teradata is limited to Period SQL database types as listed in the following table.

ODBC SQL Type Identifier	Teradata SQL Data Type	Definition in ODBC Driver for Teradata
SQL_PERIOD_DATE	PERIOD(DATE)	An anchored duration of dates
SQL_PERIOD_TIME	PERIOD(TIME) *or* PERIOD(TIME(n))	An anchored duration of times. The precision n is from 0-6, default = 6.
SQL_PERIOD_TIME_WITH_TIME_ZONE	PERIOD(TIME WITH TIME ZONE) *or* PERIOD(TIME(n) WITH TIME ZONE)	An anchored duration of times, including time zone. The precision n is from 0-6, default = 6.
SQL_PERIOD_TIMESTAMP	PERIOD(TIMESTAMP) *or* PERIOD(TIMESTAMP(n))	An anchored duration of time stamps. The precision n is from 0-6, default = 6.

ODBC SQL Type Identifier	Teradata SQL Data Type	Definition in ODBC Driver for Teradata
SQL_PERIOD_TIMESTAMP_WITH_TIME_ZONE	PERIOD(TIMESTAMP WITH TIME ZONE) *or* PERIOD(TIMESTAMP(n) WITH TIME ZONE)	An anchored duration of time stamps, including time zone. The precision n is from 0 to 6, default = 6.
SQL_TD_FIXED_NUMBER	NUMBER(p) *or* NUMBER(p, s)	Fixed Number data types are mapped to the standard SQL_DECIMAL data type
SQL_TD_FLOATING_NUMBER	NUMBER *or* NUMBER(*) *or* NUMBER(*,s)	Floating Number data types are mapped to SQL_DOUBLE.
SQL_TD_XML	XML	XML documents, and also non-well-formed text entities such as fragments of XML documents.

The ODBC SQL type values are defined in the header file `tdsql.h` in `<InstallDir>/include`, which defines Teradata-specific attributes for ODBC connection, statement, and descriptor objects.

Stored Procedures

Stored procedures, which are called Persistent Stored Modules in the ANSI SQL-92 specifications, consist of a set of control- and condition-handling statements that provide a procedural interface to Teradata Database. These statements are specified using the Stored Procedure Language (SPL).

A stored procedure can be created from ODBC and some other client utilities, and stored within the user database on the Teradata Database server. It can be executed using the SQL CALL statement.

For a complete description of stored procedures and SPL, refer to *SQL Stored Procedures and Embedded SQL* (B035-1148).

Stored Procedure Creation from ODBC

Create (define) a stored procedure from ODBC, using the SQL CREATE PROCEDURE or REPLACE PROCEDURE DDL statement.

The application (ODBC) submits the SPL statements comprising a stored procedure (called the SPL source text) to the Teradata Database server. The statements are compiled and saved on the server for subsequent execution.

For the CREATE PROCEDURE or REPLACE PROCEDURE statement syntax and other information on creating stored procedures, refer to *SQL Data Definition Language* (B035-1144).

Checking for Stored Procedure Support

Before creating a stored procedure, the ODBC application must first check whether Teradata Database supports stored procedures. This is done using the SQLGetInfo API to retrieve the details shown below. In all these cases, the term *procedure* means a stored procedure.

SQLGetInfo API specified with	Returns

SQL_PROCEDURES (Y or N)	Y = Teradata Database supports procedures
	N = Not supported

SQL_ACCESSIBLE_PROCEDURES (Y or N)	Y = User can execute all procedures returned by SQLProcedures N = Some of the procedures returned are unavailable to the user
SQL_PROCEDURE_TERM (PROCEDURE)	PROCEDURE in Teradata Database

Print and SPL Options

Stored procedures can specify two storing options relating to the creation-time attributes:

- Whether to save SPL PRINT statements in the compiled procedure
- Whether to store the SPL source text in Teradata Database

Print Option

The possible values are P and N. The value P indicates that the SPL PRINT statements specified in the stored procedure body will be saved in the compiled stored procedure. The value N, which is the default, indicates that the SPL PRINT statements are not to be saved. Rather, they are preserved in the SPL source text if the SPL source text is stored in Teradata Database (ProcedureWithSPLSource = Y).

ProcedureWithSPLSource

The possible values are Y and N. The value Y is the default. It indicates that the SPL source text needs to be stored in Teradata Database. The value N indicates that the SPL source text should not be stored in the server.

These are options that can be set in ODBC DSN.

SPL Compilation Errors and Warnings

Upon successfully completing the creation of a stored procedure, SQL_SUCCESS is returned to the client application.

SPL compilation errors and warnings are reported by Teradata Database as part of the SUCCESS return code only as a direct response to the CREATE PROCEDURE or REPLACE PROCEDURE request.

The application has to fetch the compilation errors and warnings with SQLFetch, followed by SQLGetData, until SQLFetch returns SQL_NO_DATA_FOUND.

If compilation errors are found, the stored procedure is not created or replaced.

Structure of Error and Warning Messages

SPL compilation error or warning messages are uniquely identifiable. The error code begins with SPL, followed by a four-digit number.

This number is followed by an E that indicates an error, or W that indicates a warning. A line number in parenthesis (L) indicates where in the SPL source text the error or warning was detected. An appropriate error/warning text follows the code.

Executing Stored Procedures from ODBC

Stored procedures can be executed from any ODBC application using the SQL CALL statement.

IN, INOUT, or OUT parameters can be submitted with the CALL statement. A CALL statement must have the same number of call arguments as the called stored procedure has parameters. For other rules governing the stored procedure parameters, see the CALL section in *SQL Stored Procedures and Embedded SQL* (B035-1148).

Rules for Input and Output Arguments

The following rules apply to the input and output arguments submitted with the SQL CALL statement in ODBC:

- An IN or INOUT argument must be a question mark (? is used as an input placeholder) or value expression, with the following conditions:

Argument	Condition
A value expression	Must not contain colon-preceded identifiers. It must be a constant expression. The value of the expression is treated as the input value for the corresponding parameter in the called stored procedure. A NULL value expression can be used to initialize the corresponding parameter to NULL.
?	The value for the corresponding IN or INOUT parameter of the called procedure must be set using ODBC-specific calls prior to calling the stored procedure.

- An OUT argument must be an ***`OUT call placeholder`*** or a “?” character. A placeholder can consist of a Teradata data definition and the Teradata Database-supported FORMAT, TITLE, and NAMED phrases. If the argument is a “?” character, the value for the corresponding OUT parameter of the called procedure must be set using ODBC-specific calls prior to calling the stored procedure.

For the other rules and details governing the CALL statement and usage of stored procedure parameters, refer to *SQL Stored Procedures and Embedded SQL* (B035-1148).

Stored Procedures Dynamic Result Sets

Teradata Database has the capability for a stored procedure to return one or more result sets in addition to the output parameters being returned.

A stored procedure returns a result set to the client by creating a database cursor and not closing it before returning. A cursor opened by a stored procedure is positioned where the stored procedure left it off. A result set returned from a stored procedure is presented by ODBC as an ODBC cursor just like any other result set.

Example

The following example describes the usage of the stored procedure CALL statement in an ODBC application when the OutputAsResultSet option is set to N:

Assume a stored procedure spODBC has three parameters: p1 of type OUT, p2 of type INOUT, and p3 of type IN.

The“?” character acts as a placeholder for IN, OUT, and INOUT arguments. The “?” character placeholder arguments need to be bound with the application local variables using the SQLBindParameter ODBC SDK API call.

```
{
char *request = "CALL spODBC(?, ?, ?)";
...
SQLBindParameter(..., 1, SQL_PARAM_OUTPUT, ..., SQLINTEGER,
..., ..., AppVar1, sizeof(AppVar1), ...);
SQLBindParameter(..., 2, SQL_PARAM_INPUT_OUTPUT, ..., SQLINTEGER,
..., ..., AppVar2, sizeof(AppVar2), ...);
SQLBindParameter(..., 3, SQL_PARAM_INPUT,..,SQLINTEGER, ...,...,AppVar3,
sizeof(AppVar3),...);
...
SQLExecDirect(hstmt, request);
...
}
```

where the following is true:

AppVar1, AppVar2, and AppVar3 are the ODBC application-specific local variables of INTEGER data type and these contain certain values such as input data while sending the request and output data while retrieving the results.

The second argument in the SQLBindParameter() is the “?” number ordered sequentially from left to right, starting at 1.

Retrieving output parameter value:

The values of INOUT and OUT parameters need to be retrieved from the response by directly printing the local variables (that were bound using SQLBindParameter) after a SQLFetch API call, or by using the SQLBindCol ODBC SDK API followed by SQLFetch API call.

```
SQLBindCol(..., 1, ..., AppVar1, ..., ...);
SQLBindCol(..., 2, ..., AppVar2, ..., ...);
```

where the following is true:

the second argument in the SQLBindCol() is the parameter number of result data, ordered sequentially from left to right, starting at 1.

The values of INOUT and OUT parameters also can be retrieved using SQLFetch ODBC SDK API followed by SQLGetdata API.

External Stored Procedures

An External Stored Procedure is written in a language other than SQL, and is defined by the ANSI SQL: 1999 standard. ODBC support of Teradata Database External Stored Procedures (XSP) is transparent to the ODBC user, and should have no effect on existing user applications.

The full specification of the syntax, format, and rules for both creating and invoking XSPs is beyond the scope of this document. See *SQL External Routine Programming* (B035-1147) for details on how to create and invoke user-defined functions.

XSPs are similar to UDFs except the CALL statement is used in the same manner as a stored procedure to invoke an XSP.

Restrictions

The ability to create a XSP function with the XSP source located on the client is restricted. If you try to create a XSP from source on the client, a message will be returned indicating XSPs created from source on the client are not supported.

Auto-Generated Key Retrieval

When the Identity Column Teradata Database column attribute is associated with a column, it causes the database to generate a table-level unique number for the column for every inserted row. Starting with Teradata Database V2R6.2, requests that insert into identity columns (INSERT, INSERT … SELECT, UPSERT, MERGE-INTO) can optionally return a result set containing identity column values (also known as auto-generated keys) for the inserted rows.

An application can specify that auto-generated keys are to be returned from requests that insert into identity columns in the following ways:

- Using the Return Generated Keys configuration option on DSN setup dialog or the ReturnGeneratedKeys option in `odbc.ini` file.
- Using the ReturnGeneratedKeys connection string keyword in a call to SQLDriverConnect or SQLBrowseConnect
- Using the SQL_ATTR_AGKR connection attribute in a call to SQLSetConnectAttr
- Using the SQL_ATTR_AGKR statement attribute in a call to SQLSetStmtAttr

In all cases, the application can specify that only the identity column is returned for the inserted rows, that all columns of the inserted rows are returned, or that no inserted rows are returned (the default behavior).

When auto-generated key retrieval is enabled, a request that inserts into tables containing identity columns returns two results: a row count with the number of inserted rows and a result set containing the auto-generated keys as a single column or the complete rows inserted, depending on the configuration. The insert request becomes similar to a macro that first inserts and then selects the identity column or all columns of the rows just inserted. The application should call SQLMoreResults to position to the second result and call SQLBindCol/SQLFetch or SQLFetch/SQLGetData to retrieve the result set containing the generated keys.

SQL Descriptor Fields

Teradata Database Versions from V2R6.2 and up support improved meta data for parameters used in SQL requests and for columns in result sets. The improved meta data is collectively known as Extended Statement Information. ODBC Driver for Teradata uses the extended statement information if it is supported by the database, but the use of extended statement information in ODBC can also be explicitly disabled through a configuration option (Enable Extended Statement Information check box in the DSN setup dialog, and EnableExtendedStmtInfo in the `odbc.ini` file).

When extended statement information is available in ODBC Driver for Teradata, a conforming application should not experience any differences. The application can do the following:

- Discover that SQLDescribeParam is supported and start using it
- Receive SQL_UNKNOWN_TYPE when calling SQLGetDescField for some parameters in certain SQL expressions where the database cannot determine the type
- Find that some descriptor fields that were not set are now set or have different values
- Discover that information about datetime types and intervals is now more precise

The different values of some of the descriptor fields in ODBC Driver for Teradata when extended statement information is available might affect applications that rely on the old behavior. If necessary, such applications can obtain the old behavior by disabling extended statement information in the ODBC configuration.

The following sections describe differences in the values of descriptor fields when extended statement information is available.

SQL_DESC_UNSIGNED is set regardless of FORMAT

The SQL_DESC_UNSIGNED ODBC descriptor record field is set to SQL_TRUE if the column type is unsigned or non-numeric. SQL_DESC_UNSIGNED is set to SQL_FALSE if the column type is signed.

An application can ask for the signed or unsigned characteristics of a column in a result set by using SQL_DESC_UNSIGNED in a call to SQLColAttribute or SQLGetDescField. An application can ask for the signed or unsigned characteristics of a parameter marker associated with a prepared SQL statement by using SQL_DESC_UNSIGNED in a call to SQLGetDescField.

When extended statement information is not available, ODBC Driver for Teradata determines the signed or unsigned characteristics by looking at the database type and the display format string.

If the format string includes a sign character, then the column or parameter is classified as signed and SQL_DESC_UNSIGNED is set to SQL_FALSE. If the format string does not include a sign character, then SQL_DESC_UNSIGNED is set to SQL_TRUE. For example, if the column type is INTEGER and the format 'ZZZ9', then SQL_DESC_UNSIGNED is SQL_TRUE, but if the format is '+ZZZ9', then the SQL_DESC_UNSIGNED is SQL_FALSE.

In ODBC Driver for Teradata with extended statement information available, the driver as default obtains the signed or unsigned characteristics from the extended statement meta data provided by the database. For example, if the column type is INTEGER, then it is always signed and SQL_DESC_UNSIGNED is SQL_FALSE, regardless of any format string.

The behavior in ODBC Driver for Teradata with extended statement information available is considered more correct because it reflects the database meta data more closely.

SQL_DESC_BASE_COLUMN_NAME when there is an alias

When extended statement information is not available, the value of the SQL_DESC_BASE_COLUMN_NAME descriptor field will always be the same as the value of the SQL_DESC_NAME descriptor field. The value is the alias, if there is one; otherwise, the column name, if there is one, or the empty string, if there is no alias or column name.

When extended statement information is available, the database returns the alias and base column name as separate meta data items and ODBC is therefore able to distinguish between and SQL_DESC_NAME. The values will differ if there is an alias.

Consider, for example:

```
SELECT k AS kalias FROM SomeTable;
```

With extended statement information available, ODBC Driver for Teradata will return "k" for SQL_DESC_BASE_COLUMN_NAME and "kalias" for SQL_DESC_NAME.

SQL_DESC_CASE_SENSITIVE

The SQL_DESC_CASE_SENSITIVE descriptor record field contains SQL_TRUE if the column or parameter is treated as case-sensitive for collations and comparisons or if it is a noncharacter column.

When extended statement information is not available, the value of the SQL_DESC_CASE_SENSITIVE descriptor field will always be SQL_TRUE for a character column, regardless of the column definition (“CASESPECIFIC” or “NOT CASESPECIFIC”).

When extended statement information is available, the value of the SQL_DESC_CASE_SENSITIVE descriptor field matches the column definition; for example, the value is SQL_TRUE if the column is defined as “CASESPECIFIC” and the value is SQL_FALSE if the column is defined as “NOT CASESPECIFIC”.

SQL_DESC_TD_UDT_INDICATOR

The SQL_DESC_TD_UDT_INDICATOR descriptor record field contains UDT Indicator of the column. When the column is defined as UDT, then the SQL_DESC_TD_UDT_INDICATOR describes type of the UDT defined. When the column is defined as a non-UDT, then the SQL_DESC_TD_UDT_INDICATOR will always have a value zero. The value of SQL_DESC_TD_UDT_INDICATOR descriptor field are exposed in the header file `tdsql.h`.

When extended statement information is not available, the value of the SQL_DESC_TD_UDT_INDICATOR descriptor field will always be zero.

With extended statement information available, the value of the SQL_DESC_TD_UDT_INDICATOR descriptor field will have one of the UDT Indicator values from the following table based on the column definition.

UDT Indicator Value	Definition	Description
1	SQL_TD_UDT_STRUCTURED	Structured UDT
2	SQL_TD_UDT_DISTINCT	Distinct UDT
3	SQL_TD_UDT_INTERNAL	Internal UDT
0	SQL_TD_UDT_BASE	For all other data types

SQL_DESC_TD_UDT_NAME

The SQL_DESC_TD_UDT_NAME descriptor record field contains the Fully Qualified Type Name of the UDT column.

When extended statement information is not available, the value of the SQL_DESC_TD_UDT_NAME descriptor field will always be empty string.

With extended statement information available, the value of the SQL_DESC_TD_UDT_NAME descriptor field will have Fully Qualified Type Name based on the column definition.

International Character Set Support

Introduction

ODBC Driver for Teradata supports the Unicode Data Dictionary. It is a Unicode Driver that seamlessly supports both Unicode and ANSI applications.

ODBC Driver for Teradata communicates with Teradata Database using the following supported session character sets:

- Unicode UTF8 and UTF16 session
- Double-byte Chinese and Korean
- EUC and SJIS double-byte Kanji and single-byte Katakana
- ASCII and Latin session
- Single byte Thai, Cyrillic, Hebrew, Turkish and Vietnam session

The following table contains a complete list of supported session character sets.

Supported Session Character Sets
ASCII
UTF8
UTF16
LATIN1252_0A
LATIN9_0A
LATIN1_0A
LATIN1252_3A0
KANJISJIS_0S
KANJIEUC_0U
KANJIEBCDIC5035_0I
KANJI932_1S0
TCHBIG5_1R0
SCHGB2312_1T0
SCHINESE936_6R0
TCHINESE950_8R0

Supported Session Character Sets
HANGULKSC5601_2R4
HANGUL949_7R0
ARABIC1256_6A0
CYRILLIC1251_2A0
HEBREW1255_5A0
LATIN1250_1A0
LATIN1254_7A0
LATIN1258_8A0
THAI874_4A0

Unicode ODBC API plus Unicode session character sets combine to provide full support for Unicode applications. Unicode data flows from a Unicode ODBC application to the driver and onto Teradata Database without any loss of data due to conversions.

ODBC C Character Data Types

ODBC Driver for Teradata conforms to ODBC 3.52 specifications and supports SQL_C_CHAR and SQL_C_WCHAR C data types as listed below:

ODBC C Data Type	Windows Platform	UNIX Platforms	Apple OS X
SQL_C_CHAR	Character string encoded in the code page of the application (Windows application code page)	Character string encoded in the code page of the application (ODBC application code page)	Character string encoded in the code page of the application (current locale LC_TYPE category)
SQL_C_WCHAR	Unicode (UTF16) character string	Unicode (UTF8) character string	Unicode (UTF32) character string

An ODBC application can pass and retrieve character data encoded in the application code page (SQL_C_CHAR) or Unicode (SQL_C_WCHAR) character data to or from the ODBC driver. ODBC Driver for Teradata converts the character string to or from the session character set when required; see Conversion and Error Handling for additional information.

ODBC SQL Character Data Types

The ODBC specification supports six SQL character data types while Teradata supports three character data types. The following table shows how ODBC Driver for Teradata maps the Teradata character data type to ODBC SQL data types.

ODBC SQL Character Data Types

Teradata Data Type	ODBC SQL Data Type Non-Unicode Session Character Set	ODBC SQL Data Type Unicode Session Character Set
CHAR	SQL_CHAR	SQL_WCHAR
VARCHAR	SQL_VARCHAR	SQL_WVARCHAR
CLOB	SQL_LONGVARCHAR	SQL_WLONGVARCHAR

For additional information, see “Application Model With Single Form-of-Use” in *International Character Set Support* (B035-1125).

Conversion and Error Handling

This section describes how ODBC Driver for Teradata does the following:

- Converts Parameter and Result set data to or from C data types to or from session character set
- Converts SQL-Text to session character set
- Handles conversion errors

Parameter and Result Set Data

The following table lists parameter and result set data.

Note:

The new driver strictly adheres to the exact conversion from the application character set to the session character set regardless of *unicode vs non-unicode.* The following table shows the conversion for **Parameter**, and **Result Set** is the opposite.

	Session Character Set ASCII or User-defined	**Session Character Set Non-Unicode**	**Session Character Set Unicode**
SQL_C_CHAR	No Conversion	Convert from Application Code Page to Unicode Note: See Application Code Page.	Convert from Application Code Page to Unicode Note: See Application Code Page.
SQL_C_WCHAR	Convert from Application Code Page to Platform Unicode representation	Convert from Session Character Set to the Platform Unicode representation (UTF8 on the UNIX OS, UTF32 on Apple OS X, and UTF16 on Windows)	No Conversion when session character set matches the Platform Unicode representation (UTF8 on the UNIX OS, and UTF16 on Windows. On Apple OS X, convert from session character set to UTF32 Unicode representation.

Error Handling

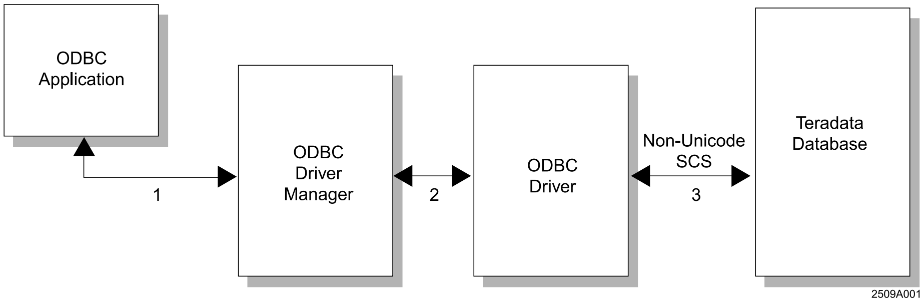
Not all characters can be converted between Unicode and other non-Unicode code pages. Therefore, some characters are converted into error characters. For additional information on error characters, see Chapter 8 in *International Character Set Support* (B035-1125). In most cases, conversion errors result in the following Teradata Database errors:

- 6706 – The string contains an untranslatable character
- 6705 – An illegally formed character string was encountered during translation

In some cases, the “?” is used as the error character. In these cases, Teradata Database cannot detect conversion errors. Therefore, SQL-Text and character strings should match the session character set repertoire to avoid conversion errors.

ANSI ODBC Applications

In the following section, assume that the ANSI application is limited to SQL_C_CHAR data binding and a non-Unicode session character set is used. The following figure depicts the programming model for ODBC Driver for Teradata:

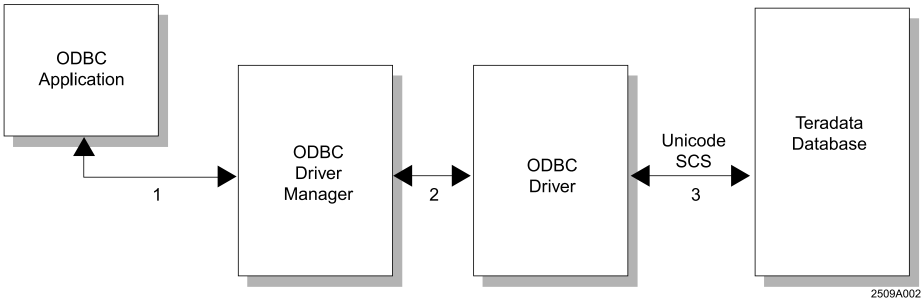


1. SQL-Text and SQL_C_CHAR character data flows between the ODBC Application and the ODBC Driver Manager
2. The ODBC Driver Manager:
    - Converts the SQL-Text to Unicode and passes it to the ODBC Driver
    - Does not convert the SQL_C_CHAR data and passes it as is to the ODBC Driver
3. The ODBC Driver for Teradata Database:
    - Converts the SQL-Text from Unicode to Session Character Set
    - Does not convert the SQL_C_CHAR data and passes it as is to Teradata Database

Unicode ODBC Applications

In this subsection, assume that the Unicode application is limited to SQL_C_WCHAR data binding and a Unicode session character set is used.

The following figure depicts the programming model for Unicode applications accessing ODBC Driver for Teradata.



1. SQL-Text and SQL_C_WCHAR character data flows between the Unicode ODBC application and the ODBC Driver Manager

2. The ODBC Driver Manager does not convert SQL-Text or the character strings and passes them as is to the ODBC driver
3. ODBC Driver for Teradata does not convert SQL-Text or character strings and passes them as is to Teradata Database

Application Considerations

The following subsections discuss the use of Unicode in applications accessing ODBC Driver for Teradata.

UNICODE Symbol Definition

If an application is compiled with the UNICODE symbol defined, then calls to ODBC API functions are mapped to their corresponding W-functions through macro substitution in the `sqlucode.h` header file. For example, a call to SQLExecDirect is mapped to a call to SQLExecDirectW.

If the UNICODE symbol is undefined, then the application uses Unicode string arguments by explicitly calling W-functions.

Applications can be written to be compiled as either Unicode or ANSI applications. In that case, the character data type can be declared as SQL_C_TCHAR. This is done using a macro that inserts SQL_C_WCHAR if the application is compiled as a Unicode application (with UNICODE symbol defined), or inserts SQL_C_CHAR if compiled as an ANSI application. The application programmer must be careful of functions taking SQLPOINTER as an argument. In addition, the size of the length argument changes for string data types, depending on whether the application is ANSI or Unicode.

On Windows, definitions in the `tchar.h` include file are useful for applications built as Unicode or ANSI. Unicode definitions in `tchar.h` are controlled by the _UNICODE #define function (preceded by an underscore).

See the MSDN ODBC programmer's manual for additional information.

Unicode Character Types

Windows

On Microsoft Windows, the Unicode character type is a distinct C/C++ type called wchar_t. Strings of this type use UTF16 Unicode encoding, and many string support functions can be applied to them.

UNIX/Linux

On a UNIX system, there is no distinct C/C++ type for UTF8 encoded Unicode strings. Strings of type char are commonly used to represent character strings encoded in UTF8, but care must be used when applying string manipulation functions, specifically with respect to lengths of strings that can be measured in bytes and characters.

Most UNIX system implementations also have a type wchar_t, but it is usually a 32-bit type used for fixed length character encodings such as UTF32, and not UTF8. For such systems, another approach is to use wchar_t internally within the application, and then convert strings of that type to UTF8 and back whenever they are passed to external interfaces such as ODBC Driver for Teradata.

Whenever possible, the SQLWCHAR ODBC character type should be used for Unicode strings instead of the wchar_t type, since SQLWCHAR and wchar_t are not the same on all operating systems.

On a UNIX system, Unicode encoding for strings and data passed to ODBC Driver for Teradata can be changed from the default UTF8 to UTF16 as follows:

1. Define SQLWCHARSHORT. For example, add the following to your code:

   ```
   #define SQLWCHARSHORT
   ```

   Note:

   SQLCHARSHORT changes definitions of SQL_WCHAR from char* to short * and must be defined before ODBC include files are specified.

2. Set the SQL_ATTR_APP_UNICODE_TYPE environment attribute to SQL_DD_CP_UTF16. For example, add the following to your code:

   ```
   // Specify the unicode encoding for the application. SQL calls and
   // data are both affected. No other environment variables or
   // connection options (including DSN options) are needed.
   rc = SQLSetEnvAttr
       (m_henv, SQL_ATTR_APP_UNICODE_TYPE,
       (void *) SQL_DD_CP_UTF16, SQL_IS_INTEGER);
   ```

Apple OS X

On Apple OS X, the wchar_t type is available, and is a 32-bit type. Since ODBC Driver Manager expects Unicode strings in UTF-32 encoding, the wchar_t type can be used to represent Unicode strings.

Length Arguments for Unicode ODBC Functions

Many ODBC interface functions expect string arguments that specify the length of character string input and output values. While some functions expect Unicode arguments to specify such lengths in bytes, others expect lengths to be specified as character counts. This varies by platform and Unicode encoding.

- **UTF16/UTF32 Encoded Unicode Strings:** The following paragraph from the Unicode section of Chapter 17, "Programming Considerations," of *Microsoft ODBC 3.0 Programmer's Reference* states the ultimate rule regarding the specification of length arguments in Unicode functions:

  "Unicode functions that always return or take strings or length arguments are passed as count of-characters. For functions that return length information for server data, the display size and precision are described in number of characters. When a length (transfer size of the data) could refer to string or non-string data, the length is described in octet lengths. For example, SQLGetInfoW will still take the length as count-of-bytes, but SQLExecDirectW will use count-of-characters."
- **UTF8 Encoded Unicode Strings on a UNIX System:** UTF8 is the default Unicode encoding for ODBC applications running on a UNIX system. All string length arguments to ODBC interface functions should be specified as count of bytes.

User-Defined Session Character Set Support

ODBC Driver for Teradata uses the application code page for Unicode to user-defined session character set conversions unless a translation DLL is specified. A user-defined session character set can be defined using the setup program. The following figure provides an example, using the **ODBC Driver Setup for Teradata Database** dialog box. A character set called *userdefined* is set.

Note:

During logon, the session character set does not read ***`userdefined`***. This is correct because translation DLLs are loaded after logon. For instance, ASCII is used to logon to the database and then the translation DLL is loaded and used for conversions between Unicode and the session character set.

Translation DLLs

A translation library is a dynamic linked library that contains functions for translating all the data flowing between the Teradata server and the driver. Translation DLLs are used if local character sets are *not* supported by the ODBC Driver or Teradata.

After a translation DLL has been specified, the driver loads it and calls it to translate all data flowing between the application and data source.

This includes the following:

- All SQL statements
- Character parameters being sent to the data source
- All character result set data
- Character meta data such as table and column names
- Error messages retrieved from the data source

Connection data is not translated, because the translation DLL is not loaded until after the application has connected to the data source.

Refer to the *ODBC Programmer's Reference* for information on how to write a translation DLL.

For UNIX OS clients, use the CharacterSet keyword instead.

To define the translation DLL name and option, use the **Teradata ODBC Driver Advanced Options** dialog box. The example in the figure that follows shows that user uses `Translation.dll` as the translator.

For UNIX OS clients, use CharacterSet TransitionDLL, and TranslationOption instead. The Translation DLL path cannot be more than 255 characters. Translation Option is used by translation DLL and optional. The driver will convert it to a 32-bit integer and pass it to the translation DLL. Translation DLL can be used for a supported session character set but it is strongly discouraged.

For an example of translation DLL, refer to sample translator that comes with the Driver Manager.

Application Code Page

The ODBC Driver Manager handles the conversion needed when an ANSI application calls the Unicode ODBC driver.

- On a UNIX system, the Driver Manager uses the ODBC application code page setting in the DSN or connection string when converting an ANSI string argument passed to the ANSI ODBC function calls to or from Unicode. An example of this conversion is when an ANSI application calls SQLExecDirect.
- On a UNIX system, ODBC Driver for Teradata uses the ODBC application code page specified in the DSN or connection string for conversions to and from Unicode. These conversions are done for data that is bound with the SQL_C_CHAR type.
- On Windows, the Driver Manager uses the Windows application code page when converting to or from Unicode.
- On Apple OS X, the ODBC driver manager uses the current locale LC_TYPE category when converting to or from Unicode.

ODBC Application Code Page Values (Linux/UNIX and Apple OS X)

ODBC application code page values can be set either in the DSN or the connection string. Any setting in the connection string takes precedence over settings in the DSN.

On Apple OS X, only the following code page value is supported due to a restriction imposed by iODBC Driver Manager:

ODBC Application Code Page Value	Description
4	ISO 8859-1 Latin-1

The following table lists ODBC application code page values used in ODBC Driver for Teradata on a Linux/UNIX system.

ODBC Application Code Page Values	Description
3	ISO 646 7-bit ASCII
4	ISO 8859-1 Latin-1
2009	CP 850 - European code page
2011	CP 437 - US code page
2004	HP ROMANS
2027	Standard Macintosh Roman
17	Shift-JIS proper
18	EUC-JIS encoding
2025	EUC-CNS encoding
2024	Microsoft CP 932 = Win32J-DBCS
5	ISO 8859-2 Latin-2 Eastern Europe
8	ISO 8859-5 Latin/Cyrillic
9	ISO 8859-6 Latin/Arabic
10	ISO 8859-7 Latin/Greek
11	ISO 8859-8 Latin/Hebrew

ODBC Application Code Page Values	Description
12	ISO 8859-9 Latin-5 Turkish
13	ISO 8859-10 Latin-6 Nordic
6	ISO 8859-3 Latin/Esperanto/Galician
7	ISO 8859-4 Latin/Estonian/Latvian
16	JIS_Encoding
30	ISO_646_IRV
37	ISO_2022_KR
38	EUC_KR
39	ISO_2022_IP
40	ISO_2022_IP_2
57	GB_2312_80
104	ISO_2022_CN
105	ISO_2022_CN_EXT
109	ISO 8859-13
110	ISO 8859-14
111	ISO 8859-15 Latin-9 Western Europe with Euro sign
2084	KOI8 - Cyrillic
2259	TIS 620 - Thai standard
2026	Big5 Traditional Chinese
2028	IBM EBCDIC (8859-1 convertible)
2030	IBM EBCDIC Germany/Austria
2033	IBM EBCDIC Denmark/Norway
2034	IBM EBCDIC Finland/Sweden
2035	IBM280

ODBC Application Code Page Values	Description
2037	IBM EBCDIC Spain/Latin America
2038	IBM EBCDIC U.K.
2039	IBM EBCDIC Katakana for DB2
2040	IBM EBCDIC France
2041	IBM EBCDIC Arabic bilingual
2043	IBM424
2044	IBM EBCDIC Western Europe
2045	IBM851
2010	PC Eastern Europe
2046	PC Cyrillic
2047	PC Turkish
2048	PC Portuguese
2049	PC Icelandic
2050	PC Canadian French
2051	PC Arabic
2052	PC Nordic
2054	PC Greek
2055	IBM EBCDIC Eastern Europe
111	Microsoft Thai SB code page
17	Japanese IBM J-DBCS: CP 897 + CP 301
113	PC Simplified Chinese
36	PC (MS) Korean, similar to EUC-KSC
2026	PC (MS) Traditional Chinese (~Big5)
18	EUC-JIS

ODBC Application Code Page Values	Description
3063	IBM EBCDIC Turkish
2056	IBM871
2062	IBM918
2063	IBM1026
2085	HZ_GB_2312
2086	IBM866
2087	IBM775
2089	IBM00858
2091	IBM01140
2092	IBM01141
2093	IBM01142
2094	IBM01143
2095	IBM01144
2096	IBM01145
2097	IBM01146
2098	IBM01147
2099	IBM01148
2100	IBM01149
2102	z/OS Open Edition
2250	MS Windows 3.1 Eastern European
2251	MS Windows 3.1 Cyrillic
2252	MS Windows 3.1 US (ANSI)
2253	MS Windows 3.1 Greek
2254	MS Windows 3.1 Turkish

ODBC Application Code Page Values	Description
2255	MS Windows Hebrew
2256	MS Windows Arabic
2257	MS Windows Baltic
2258	MS Windows Vietnamese

UTF8 Pass Through Functionality

Some existing Unicode applications on Windows have been using the UTF8 encoding of Unicode to be able to pass Unicode data through to the Database using ODBC ANSI function calls. These applications utilize the UTF8 session character set. They will face the following issues with the Unicode driver:

- Supplying Latin or Kanji object names through the ANSI API will fail because the Driver Manager cannot translate to UTF-16 using the application code page as the string will contain invalid characters. For example, working on a Japanese PC, the Driver Manager will attempt to convert UTF8 characters thinking it is SJIS.
- Supplying Latin or Kanji data in the SQL request, for example: “INTO T values …” will fail because the Driver Manager cannot translate to UTF16 using the application code page.
- Character data of type SQL_C_CHAR. The new driver will convert data to and from the UTF-8 session character set using the applications code page. This will fail for non-ASCII characters.

Therefore, supplying and retrieving UTF8 data using SQL_C_CHAR data binding is not supported.

Restrictions

- ODBC Driver for Teradata does not support EBCDIC session character sets for the Unicode ODBC driver. Note that KANJIEBCDIC5035_0I is supported on Windows. See Session Character Sets and Translation DLLs for more information.
- Retrieving fixed character fields (for example, C01 CHAR(10)) utilizing the UTF8 session character set can result in padded strings due to the database export factor utilized when translating characters to the session character set. Use UTF16 session character set which contains an export factor of 2 for all characters, or utilize varchar() variables instead of char().
- Kanji User ID and Password are not supported on an English-enabled client machine. Kanji User ID and passwords can be used on English language PCs if the database

release is 12 or higher, the database is Kanji-enabled, and the session character set is UTF16 or UTF8. This is a Unicode Data Dictionary enhancement.

- Characters not found during translations will be substituted by an error character. For example, given a session on a Japanese Windows machine, the string “abecedný” would convert the “ý” to “0xfcfc”, indicating a conversion error from the ODBC code converter. The character, “ý”, is not a valid SJIS character. See the “Conversion and Error Handling” chapter in *International Character Set Support* (B035-1125).
- ANSI applications requesting column names or other meta data containing non-ASCII characters through the SQLColAttribute method, using a Unicode session character set, will retrieve the non-ASCII characters in UTF8 encoding (not encoded by the application code page). ANSI applications using non-Unicode sessions will not encounter this problem. It could be a problem if a customer is in the process of migrating applications to Unicode having some ANSI and some Unicode applications, all running over Unicode sessions. A workaround is to use the old SQLColAttributes convention ('s' at the end).
- Calling SQLColumns against a table with a 2 GB CLOB column can cause an error because the value in the BUFFER_LENGTH column overflows when bound to unsigned integer types. The maximum CLOB size recommended for use with UTF8 (export factor 2) and UTF16 session character sets is 1 GB.

Atomic UPSERT

UPSERT is a composite of UPDATE and INSERT operations, applied to a single row in a table, so that, if the UPDATE fails because the target row does not exist, the INSERT is automatically executed.

An UPSERT-like feature was formerly supported by load utilities such as Teradata TPump and Teradata MultiLoad, but the implementation was a two-pass one (first UPDATE will be executed; if it fails, then INSERT will be executed). For better performance of the UPSERT-like queries, the Atomic UPSERT feature was implemented on the server.

To support UPSERT SQL for Teradata Database, ODBC Driver for Teradata was enhanced to support the feature in the Teradata Tools and Utilities. UPSERT allows all client applications to use this UPSERT functionality as a single query (or one-pass strategy), which has better performance when compared to Teradata TPump and Teradata MultiLoad batch logic.

While establishing a connection to Teradata Database, ODBC Driver for Teradata determines whether the UPSERT feature is supported by Teradata Database. If the database does not support the UPSERT feature, any UPSERT statements submitted by the application return the following general error to the application:

```
("UPDATE..ELSE..INSERT (UPSERT) is not supported by Database").
```

Syntax

The syntax for Atomic UPSERT is straightforward and self-explanatory, consisting of an UPDATE statement and an INSERT statement separated by an ELSE keyword, as follows:

```
UPDATE <update-operands> ELSE INSERT <insert-operands>
```

The <update-operands> and <insert-operands> are operands for regular UPDATE and INSERT SQL statements, respectively.

```
UPDATE Sales SET ItemCount = ItemCount + 1 WHERE (ItemNbr = 20 AND SaleDate =
'05/30/2000') ELSE INSERT INTO Sales (20, '05/30/2000', 1);
```

ANSI Date and Time Restrictions

Some of the restrictions for using ODBC Driver for Teradata include:

- The SET SESSION DateForm command is not allowed in a multi-statement request
- The time zone value for TIME WITH TIME ZONE and TIMESTAMP WITH TIME ZONE must be converted to an ODBC C character type. The standard ODBC C time and timestamp types do not have a time zone component
- (Windows) Since the ODBC driver does not support PMPC commands, asynchronous niching (priority of execution) of a query is not supported, but self-niching queries are supported.
- The Teradata TIME type allows fractional seconds, but there is limited support for fractional seconds of TIME in standard ODBC. As an extension to standard ODBC, the Teradata ODBC Driver preserves fractional seconds in conversions between character ODBC C data types and SQL_TYPE_TIME.

ANSI DateTime Feature

The Teradata ANSI DateTime feature, including definitions, usage, and limitations, is described in Teradata Database documentation, including information on how ODBC Driver for Teradata supports this feature, and how to use DATEs, TIMEs, and TIMESTAMPs.

Change Your DSN Configuration

You can call the SQLDriverConnect or the SQLBrowseConnect API function, and use the new DATETIMEFORMAT = option. The DateTimeFormat is only valid for the duration of the new session. You can also use one of the following SQL statements to permanently set it:

```
SET SESSION DATEFORM = ANSIDATE
SET SESSION DATEFORM = INTEGERDATE
```

The DateTimeFormat is only valid for the duration of the session. In addition, the SET SESSION DATEFORM SQL statement must not be included in a multi-statement request. Finally, this method only sets the DATE format.

DateTimeFormat Compatibility and Precision

If you set the session DateTimeFormat to old-style and your tables were created with old-style DATEs, TIMEs, and TIMESTAMPs, you will get the same results. The integer time format is not recommended, because it has been deprecated. For more information, see Integer Time.

A more interesting case is when you want to set the session DateTimeFormat to old-style, but the tables were created with ANSI DATEs, TIMEs, and TIMESTAMPs.

When the session DateTimeFormat is ANSI, the following SQL statement creates a column of type ANSI TIME with a fractional second precision of 6 (hh:mm:ss.ffffff). This is because the Teradata default fractional second precision is 6, not 0.

```
CREATE TABLE tablename (c1 TIME)
```

TIME and TIMESTAMP precision between these data types, in fractional seconds, is depicted in the following table.

Time/Timestamp Type	Precision Limits
Teradata ANSI TIME	up to 15 characters (hh:mm:ss.ffffff)
Teradata ANSI TIMESTAMP	up to 26 characters (yyyy-mm-dd hh:mm:ss.ffffff)

If requested to fetch a character string from Teradata and convert it to a TIMESTAMP, the driver will truncate the TIMESTAMP at 26 characters, and generate a warning. Similarly, when fetching a character string and converting it to a TIME, the driver will truncate the TIME at 15 characters and generate a warning. If requested to fetch a TIMESTAMP from Teradata and convert it to a TIME, the driver returns SQL_SUCCESS_WITH_INFO and 01S07 as required by the ODBC specification when converting from SQL_TIMESTAMP to SQL_C_TIME. The driver truncates the fraction silently only when fetching a character string containing a fraction field and converting it to SQL_C_TIME (converting from SQL_CHAR to SQL_C_TIME) with the DateTimeFormat DSN option for Time set to Integer. Otherwise, the truncation is not silent and the driver returns SQL_SUCCESS_WITH_INFO and 01S07 sqlstate, as required by the ODBC specification.

If the user application calls SQLBindParameter, and passes a pointer to a buffer that contains (or will later contain) an oversize character string to be used as a TIME or TIMESTAMP, truncation of the fractional seconds can occur silently.

If the user application calls SQLBindParameter, and the session DateTimeFormat is old style, it is not allowed to pass pointers to buffers that contain (or will later contain) Teradata ANSI DATEs, TIMEs, or TIMESTAMPs with prefixes (for example, TIME '14:25:00'). ANSI DATEs, TIMEs, and TIMESTAMPs that do not have these prefixes will work. ODBC syntax DATE, TIME, or TIMESTAMP literals will always work, regardless of the DateTimeFormat value.

You cannot do integer arithmetic on ANSI TIMEs and TIMESTAMPs. The following SQL statement fails:

```
SELECT ((TIME '12:13:14')(integer))/10000
```

When considering the trade-offs of setting the session DateTimeFormat to old style or to ANSI, you need to consider whether you need to do integer arithmetic on TIMEs or TIMESTAMPs.

If you use parameterized SQL against ANSI TIMEs or TIMESTAMPs, you will need to use CASTs in your SQL statements:

```
SELECT * FROM table WHERE columnname = CAST ((?) AS TIME(6))
```

Again, when deciding how to set the session DateTimeFormat, you need to consider whether adding these CASTs to your SQL statements is feasible.

Teradata Database supports the ANSI CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP functions.

Teradata limits ***n***, the fractional second precision, in a call to CURRENT_TIME(n) or CURRENT_TIMESTAMP(n), to 6. ODBC allows ***n*** to be 9. The driver passes it unaltered to Teradata, which will generate an error if it is larger than 6.

You can still use the ODBC scalar functions CURDATE and CURTIME, as well as the SQL statements DATE and TIME.

The EXTRACT SQL statement produces a result set that has an ODBC style title. If you prefer the title generated by Teradata, use SQLSetStmtOption or SQLSetStmtAttr to turn on the SQL_NOSCAN option, and do not use ODBC syntax for DATE, TIME, and TIMESTAMP literals. Alternately, you can specify your own titles after the EXTRACT SQL statement.

You cannot use EXTRACT to obtain the default time zone from an ANSI TIME or TIMESTAMP that does not include a time zone. When using EXTRACT to obtain the time zone from an ANSI TIME or TIMESTAMP that does have a time zone, you must use the TIME or TIMESTAMP prefix:

```
SELECT EXTRACT (TIMEZONE_HOUR FROM TIME '12:13:14+07:00')
```

The ODBC Driver for Teradata maps both TIME WITH TIME ZONE and TIMESTAMP WITH TIME ZONE to SQL_TYPE_TIME. Time zone values are truncated when converting values to SQL_C_TYPE_TIME or SQL_C_TYPE_TIMESTAMP, but are preserved when mapping to any of the ODBC C character types. Therefore, an application that requires time zone values should use the ODBC C character types.

Period Data Types

A period is an anchored time duration. It represents a set of contiguous time granules within that duration. It has a beginning bound (defined by the value of a beginning element) and an ending bound (defined by the value of an ending element). The representation of the period is inclusive-exclusive (for example, the period extends from the beginning bound up to–but not including–the ending bound).

The element type of a Period data type is the data type of the beginning and ending elements of a value of that Period data type. The element type can be any DateTime data type. The DateTime data types are DATE, TIME, and TIMESTAMP. The TIME and TIMESTAMP data types have a number (0-6) of fractional seconds in the seconds field which can be specified, or the default is 6; for example, TIME(3) and TIMESTAMP(6). TIME and TIMESTAMP can also explicitly include a time zone field by specifying WITH TIME ZONE (if WITH TIME ZONE is not specified, a time zone field is implicitly not included). Note that the element type must be the same for both the beginning and ending elements of a period.

ODBC driver supports the following ODBC SQL types:

- SQL_PERIOD_DATE
- SQL_PERIOD_TIME
- SQL_PERIOD_TIME_WITH_TIME_ZONE
- SQL_PERIOD_TIMESTAMP
- SQL_PERIOD_TIMESTAMP_WITH_TIME_ZONE

These types are referred to as Period ODBC SQL types.

No new ODBC C types are added; the ODBC standard does *not* allow for that.

Availability of Period Data Types

Use of Period data types in the ODBC Driver follows the database; that is, the types are available if the database supports them and are not available if the database does not support them.

It is required that the ODBC Driver time format (see the DateTimeFormat=[A|I]A A description in Teradata DSN Options) is set to ANSI when using Periods. The Date format can be Integer or ANSI. The Timestamp format is always ANSI.

Retrieving Period Data

Period data received from the database can be retrieved in a character format (SQL_C_CHAR and SQL_C_WCHAR) or as binary data (SQL_C_BINARY).

The default conversion as specified by SQL_C_DEFAULT is the same as the conversion specified by SQL_C_BINARY.

Period SQL to Character C type Conversion

The following table shows the format of returned data when retrieving it in a character format:

Period SQL Type	Conversion to SQL_C_CHAR or SQL_C_WCHAR

SQL_PERIOD_DATE	The result is a character string:
	`(‘yyyy-mm-dd’, ‘yyyy-mm-dd’)`
	where the beginning and ending date bounds are converted to the ‘yyyy-mm-dd’ format that is always used by the ODBC specification when converting date SQL data (SQL_TYPE_DATE) to the character C data.
	Example:
	`(‘2007-05-01’, ‘2007-06-01’)`

SQL_PERIOD_TIME	The result is a character string in the following format:
	`(‘hh:mi:ss[.f....]’, ‘hh:mi:ss[.f...]’)`
	where the beginning and ending time bounds are converted to the “hh:mi:ss” format that is always used by the ODBC specification when converting time SQL data (SQL_TYPE_TIME) to character C data.
	The “[.f...]” indicates the optional fractional seconds. It is only included if the Period timestamp precision is non-zero. Up to 6 digits can be used for fractional seconds (ODBC allows for 9, but Period timestamps are limited to 6).
	Example:
	`(‘06:20:21’, ‘17:12:00’)`

Period SQL Type	Conversion to SQL_C_CHAR or SQL_C_WCHAR

SQL_PERIOD_TIME_WITH_TIME_ZONE	The result is a character string in the following format:
	`(‘hh:mi:ss[.f...]+-hh:mi’’, ‘hh:mi:ss[.f...]+-hh:mi’’)`
	where the beginning and ending time bounds are converted to the “hh:mi:ss+-hh:mi’” format that is used by the ODBC driver when converting TIME WITH TIME ZONE to character C data.
	Example:
	`(‘06:20:21.22+02:00’, ‘17:12:00.22+02:00’)`

SQL_PERIOD_TIMESTAMP	The result is a character string:
	`(‘yyyy-mm-dd hh:mi:ss[.f...]’, ‘yyyy-mm-dd hh:mi:ss[.f...]’)`
	where the beginning and ending timestamp bounds are converted to the “yyyy-mm-dd hh:mi:ss[.f....]” format that is always used by the ODBC specification when converting timestamp SQL data to character C data.
	Example:
	`(‘2007-07-04 22:11:43.37’, ‘2007-07-04 22:11:50.00’)`

SQL_PERIOD_TIMESTAMP_WITH_TIME_ZONE	The result is the same as SQL_PERIOD_TIMESTAMP with the addition of the ‘+-hh:mi’ as described for SQL_PERIOD_TIME_WITH_TIME_ZONE.

ODBC Driver for Teradata returns SQL_ERROR from the conversion if the receiving buffer is too small. This is unlike DateTime types where a conversion to C character can result in SQL_SUCCESS_WITH_INFO and a truncated string. The reason for the difference is that a truncated DateTime string might be useful for the applications (for example: a truncated SQL_DATE might be used to obtain just the year), but a truncated Period string is not.

Period SQL to Binary C type Conversion

The following table shows the format of returned data when retrieving it in a binary format.

Period Data Type	Number of Bytes	Members Passed	Members Detail (in the order specified)
PERIOD(DATE)	8	2 DATE members	Date: 4-byte, signed integer. This integer represents a date in the same manner as for a DATE data type (for example, 10000*(year-1900)) + (100*month) + day)
PERIOD(TIME(n))	12	2 TIME members	Second: 4-byte, signed integer. This integer represents the number of seconds scaled by 10**6 (for example, 12.56 seconds is returned as 12560000). Hour: 1 unsigned byte. This byte represents the number of hours. Minute: 1 unsigned byte to client form. This byte represents the number of minutes.
PERIOD(TIME(n) WITH TIME ZONE)	16	2 TIME WITH TIME ZONE members	Second: 4-byte, signed integer. This integer represents the number of seconds scaled by 10**6 (for example, 12.56 seconds is returned as 12560000). Hour: 1 unsigned byte. This byte represents the number of hours. Minute: 1 unsigned byte. This byte represents the number of minutes. Time Zone Hour: 1 unsigned byte. This byte represents the hours portion of the time zone

Period Data Type	Number of Bytes	Members Passed	Members Detail (in the order specified)
			displacement, along with whether the displacement is positive or negative. A value of 16 represents 0 hours. A value less than 16 represents a negative time zone displacement for the specified hours; for example, if this is 10, the time zone is displaced by -10 hours. If the value is greater than 16, it specifies a positive time zone displacement of (Time Zone Hour - 16) hours; that is, a value of 20 implies a +4 hour displacement.) Time Zone Minute: 1 unsigned byte. This byte represents the minutes portion of the time zone displacement.
PERIOD(TIMESTAMP(n))	20	2 TIMESTAMP members	Two TIMESTAMP members containing: Second: 4-byte, signed integer. This integer represents the number of seconds scaled by 10**6 (for example, 12.56 seconds is returned as 12560000). Year: 2-byte, signed short integer. This byte represents the year value. Month: 1 unsigned byte. This byte represents the month value. Day: 1 unsigned byte. This byte represents the day of the month.

Period Data Type	Number of Bytes	Members Passed	Members Detail (in the order specified)
			Hour: 1 unsigned byte. This byte represents the number of hours. Minute: 1 unsigned byte. This byte represents the number of minutes.
PERIOD(TIMESTAMP(n) WITH TIME ZONE)	24	2 TIMESTAMP WITH TIME ZONE members	Two TIMESTAMP members containing: Second: 4-byte, signed integer. This integer represents the number of seconds scaled by 10**6 (for example, 12.56 seconds is returned as 12560000). Year: 2-byte, signed short integer. This byte represents the year value. Month: 1 unsigned byte. This byte represents the month value. Day: 1 unsigned byte. This byte represents the day of the month. Hour: 1 unsigned byte. This byte represents the number of hours. Minute: 1 unsigned byte. This byte represents the number of minutes. Time Zone Hour: 1 unsigned byte. This byte represents the time zone displacement in hours, along with whether the displacement is positive or negative. A value of 16 represents 0 hours. A value less than 16 represents a negative time zone displacement for the

Period Data Type	Number of Bytes	Members Passed	Members Detail (in the order specified)
			specified hours; for example, if this value is 10, the time zone is displaced by -10 hours. If the value is greater than 16, it specifies a positive time zone displacement of (Time Zone Hour - 16) hours; that is, a value of 20 implies a +4 hour displacement.) Time Zone Minute: 1 unsigned byte. This byte represents the minutes portion of the time zone displacement.

Any target precision set by the application in the application record descriptor is ignored by the driver. The Second values returned by the database are always 4-byte signed integers with the seconds scaled by 10**6. The precision of the source data is available through SQLDescribeCol/SQLColAttribute or directly in the Implementation Record Descriptor.

Using Period Parameters

Character C Type to Period SQL Types

The source data must be in the format described in Period SQL to Character C type Conversion.

The driver also supports the use of ODBC escape sequences occurring in the character string representation of the period value.

Binary C Type to Period SQL

The source data must be in the format described in Period SQL to Character C type Conversion.

An application can provide its own value for the precision of the period source data by setting the SQL_DESC_PRECISION field of the application descriptor. The default precision for the source data is the same as the precision of the target as indicated in the

DecimalDigits argument to SQLBindParameter. For example, a value of 14250000 in the Second field and DecimalDigits=2 would allow the value to be inserted into a PERIOD(TIME(2)).

The ColumSize argument to SQLBindParameter is ignored for Period data types.

Period Literals

A Period literal is used to specify a period constant. It has the form of the PERIOD keyword followed by a quote string in a specific format. The element type of a period literal is derived from the format of the DateTime values specified in the quoted string. The general format of period literals is:

PERIOD '(element1,element2)'

Examples:

Date literal: PERIOD '(2005-02-03, 2006-02-04)'

Time literal: PERIOD '(12:12:12.340000, 13:12:12.340000)'

Note that the character representation of a period value from the database includes single quotes around the elements and can therefore not be directly used in the construction of a literal.

Geospatial Types

Geospatial types (ST_Geometry, MBR, and so on) are supported by the ODBC driver in a transparent manner by making us of the import/export transform functionality of the Teradata Database. For example, a value of type ST_Geometry is imported/exported as a CLOB and a value of type MBR as a VARCHAR (256). Please refer to *Teradata Database, SQL Geospatial Types* (B035-1181), for more information about the transform types for the different Geospatial types.

An ODBC application sees the Geospatial types in the database as the standard SQL types SQL_LONGVARCHAR or SQL_VARCHAR depending on the database transform type. Values of the Geospatial types are transformed by the database to or from values of these standard SQL types. All the syntax for types, methods, functions, and expressions as used for Geospatial fall within the standard SQL syntax.

The specific Geospatial type behind the database transform type can be obtained through the SQL_DESC_TD_UDT_NAME descriptor field, which returns the fully qualified name of the Geospatial type. Please refer to SQL_DESC_TD_UDT_NAME for more information.

Note:

> Large object (LOB) support should not be disabled when using Geospatial data types. LOB support is enabled by default and the option to disable LOB support is deprecated.

Number Data Types

The Teradata Database Number data types are available in ODBC if supported by the database. A Teradata Database Number type is either Fixed or Floating as listed in the following table.

Number variant	Database type	Description
Fixed Number	NUMBER (p, s)	Fixed decimal of ***p*** digits with ***s*** fractional digits
	NUMBER (p)	Equivalent to NUMBER(p, 0)
Floating Number	NUMBER (*, s)	Floating decimal of ***s*** fractional digits
	NUMBER (*)	Floating decimal
	NUMBER	Equivalent to NUMBER(*)

In ODBC the Number types are treated as standard SQL types. Fixed Number data types are mapped to the standard SQL_DECIMAL data type and Floating Number data types are mapped to SQL_DOUBLE. This means conforming applications do not need to know that the database type corresponding to an SQL_DECIMAL or SQL_DOUBLE is a Number data type.

If an application needs information indicating the database type is a Number type, then the application can examine the Teradata ODBC-specific data type code. This code is returned as a descriptor field in SQLColAttribute and SQLGetDescField. Additionally, the code is returned as a Teradata-specific column in the result set from SQLGetTypeInfo, SQLColumns, and SQLProcedureColumns.

The Teradata ODBC-specific data type codes are as follows:

- SQL_TD_FIXED_NUMBER for a Fixed Number
- SQL_TD_FLOATING_NUMBER for a Floating Number

Note:

The names SQL_TD_FIXED_NUMBER and SQL_TD_FLOATING_NUMBER are defined in the **`tdsql.h`** file. For more information, see Teradata Extensions to the ODBC Standard.

SQLGetTypeInfo

The following table shows how Number data types are reflected in selected columns from the result set returned by SQLGetTypeInfo for SQL_DECIMAL and SQL_DOUBLE.

DATA_TYPE	TYPE_NAME	COLUMN_SIZE	CREATE_PARAMS	NUM_PREC_RADIX	TDODBC_DATA_TYPE
SQL_DECIMAL	DECIMAL	38	precision, scale	10	SQL_DECIMAL
SQL_DECIMAL	NUMBER	38	precision, scale	10	SQL_TD_FIXED_NUMBER
SQL_DECIMAL	NUMBER	38	Precision	10	SQL_TD_FIXED_NUMBER
SQL_DOUBLE	FLOAT	15	Null	2	SQL_DOUBLE
SQL_DOUBLE	NUMBER	15	Null	10	SQL_TD_FLOATING_NU

Retrieving Number Data

The Teradata Database Fixed Number types are mapped to SQL_DECIMAL. The standard SQL_DECIMAL to ODBC C type conversions apply with the following notes:

- Conversion to SQL_C_BINARY yields the binary representation of the Number value consisting of 1-byte length, 2-byte scale, and 1-17 bytes unscaled value.
- Conversion to SQL_C_DOUBLE or SQL_C_FLOAT may lose precision. In this case the return code from the conversion will be SQL_SUCCES_WITH_INFO and SQLSTATE 01S07 generates.

Teradata Database Floating Number types are mapped to SQL_DOUBLE. The standard SQL_DOUBLE to ODBC C type conversions apply with the following notes:

- Conversion to SQL_C_CHAR or SQL_C_WCHAR preserves the full precision of the Floating Number, which may be greater than the current maximum of 15. The format of the string is an ODBC numeric literal using either exponential notation or non-exponential notation depending on the actual value.
- Conversion to SQL_C_DOUBLE (default conversion) or SQL_C_FLOAT may lose precision. In this case the return code from the conversion is SQL_SUCCES_WITH_INFO and SQLSTATE 01S07 is generated.
- Conversion to SQL_C_BINARY yields the binary representation of the Number value consisting of 1-byte length, 2-byte scale, and 1-17 bytes unscaled value.

Using Number Parameters

The standard ODBC conversions apply with the following notes:

Conversion from SQL_C_CHAR to SQL_DOUBLE preserves a precision of up to 38 digits of the number passed as a string (instead of just the precision of 15 when the underlying type is a FLOAT).

Array Data Types

Teradata Database Array data type values appearing in result sets and used as statement parameters are of ODBC SQL Variable Character type (SQL_VARCHAR or SQL_WVARCHAR). The format of an array as ODBC SQL Variable Character is the transformed format as described in *SQL Data Types and Literals* (B035-1143).

Briefly, the format is a string of each array element value separated by comma and enclosed in parenthesis as shown as below, assuming the array has N elements:

(<element1>,<element2>,...<elementN>)

where each element is transformed depending on its type.

Arrays are visible through the ODBC Catalog functions as shown in the following table.

Catalog function	Description
SQLTables	TABLE_TYPE column: "TYPE". TABLE_NAME column: Contains type name, i.e. the name of the ARRAY type from the CREATE TYPE.
SQLColumns	DATA_TYPE column: SQL_UNKNOWN_TYPE (zero) TYPE_NAME column: Type name.
SQLProcedureColumns	As for SQLColumns

XML Data Type

XML data type values include:

- XML documents.
- Non-well-formed text entities, including XML document fragments, such as sequences.
- XML schema predefined simple type values, such as atomic values.

The Teradata Database XML data type is available in ODBC if supported by the database. The XML data type is shown in ODBC as a Teradata-defined SQL type named `SQL_TD_XML`.

The name SQL_TD_XML is defined in the `tdsql.h` file. For information, see Teradata Extensions to the ODBC Standard.

Note:

> LOB support should not be disabled when using XML data types. LOB support is enabled by default and the option to disable LOB support is deprecated.

SQLGetTypeInfo

The following table shows how the XML data type returns in selected columns from the result set returned by `SQLGetTypeInfo`.

DATA_TYPE	TYPE_NAME	COLUMN_SIZE	CREATE_PARAMS	NULLABLE	TDODBC_DATA_TYPE
SQL_TD_XML	XML	2097088000	NULL	SQL_NULLABLE	SQL_TD_XML

XML Data Type Values and Conversions

The ODBC driver supports conversions of XML values to and from both ODBC Character C types (SQL_C_CHAR and SQL_C_WCHAR) and the Binary C type SQL_C_BINARY.

Note:

> XML data type values retrieved from the database may be complete XML documents, but also non-well-formed text entities, including XML document fragments, such as sequences or atomic values. XML data type values inserted into the database must be XML documents (well-formed).

The recommended conversion of XML is to or from SQL_C_BINARY or SQL_C_WCHAR because it eliminates conversion to a non-Unicode code page used by the application, which can result in conversion errors. The default ODBC C type for SQL_TD_XML is SQL_C_BINARY. SQL_TD_XML values represented as ODBC C types are listed below.

XML represented as	Description
SQL_C_BINARY	The value is a sequence of characters encoded in UTF8 regardless of any encoding declaration in the XML text.
SQL_C_CHAR	The value is a sequence of characters encoded in the application code page regardless of any encoding declaration in the XML text.
SQL_C_WCHAR	The value is a sequence of characters encoded in the Unicode encoding in effect for the application (UTF8 or UTF16 for Linux/ Unix. UTF16 for Windows, UTF32 for Apple OS X). The encoding is independent of any encoding declaration in the XML text.

For a given XML document the recommended ODBC C type for working with the document in ODBC depends on the XML document encoding as determined by the XML declaration in the document. Recommended ODBC C types for XML document encoding are listed below.

XML Document Encoding Declaration	Recommended ODBC C type for working with the document in an ODBC application		
	Windows	**Linux/Unix**	**Apple OS X**
UTF8	SQL_C_BINARY or SQL_C_WCHAR. Note that when using SQL_C_WCHAR the encoding will be UTF16	SQL_C_BINARY or SQL_C_WCHAR.	SQL_C_BINARY or SQL_C_WCHAR
UTF16	SQL_C_WCHAR	SQL_C_WCHAR with UTF16 Unicode encoding as described in Unicode Character Types.	SQL_C_WCHAR
Other	SQL_C_CHAR using application code page matching document encoding	SQL_C_CHAR using an application code page matching the XML document encoding.	SAL_C_CHAR with locale LC_TYPE setting matching the document encoding.

JSON Integration

JSON (JavaScript Object Notation) is a data-interchange format.

It is a text based type that is used to encapsulate a set of name/value pairs, or an array of values. This flexible format allows for convenient storage of varied data that is related in some manner. Since this type is text based, it can be defined as accepting text in CHARACTER SET UNICODE or LATIN. Other character sets are not supported.

The JSON data type can be used as a column of a table, as a parameter to or return from a UDF/UDM/XSP/TDSP/Table Operator, and may be used to declare a local variable inside of a TDSP. In each particular use of the JSON data type, the character set must be specified (or the character set of the database will be used) and the maximum length must be set.

JSON data types may be used as the attribute of an ANSI Structured UDT, but not as the base type of an ANSI Distinct UDT or as the element type of a Teradata ARRAY type.

The JSON data type can define a maximum length on a per instance basis. Variable Maximum Length means the JSON type has some default maximum length, but that length may be adjusted in places where that type is used in a manner analogous to the VARCHAR data type. Thus the length may never exceed the absolute maximum length, but a particular instance of the JSON type may define a shorter maximum length.

That absolute maximum is equal to 16776192 bytes, so if the JSON type utilizes the LATIN character set, that is equal to 16776192 characters and if it utilizes the UNICODE UTF16 character set, that is equal to 8388096 characters.

Note:

> LOB support should not be disabled when using JSON data types. LOB support is enabled by default and the option to disable LOB support is deprecated.

SQLGetTypeInfo

The ODBC function SQLGetTypeInfo function can be used to enumerate all ODBC SQL types supported by a data source or retrieve information for a single ODBC SQL type. In both cases it returns various information about the ODBC driver and Database specifics for each ODBC SQL type. More than one data type supported by the data source can map to a single ODBC SQL type identifier.

The function returns the results as a standard result set, ordered by DATA_TYPE and then by how closely the data type maps to the corresponding ODBC SQL data type.

To support the JSON data type, the Teradata specific column TDODBC_DATA_TYPE will be set to SQL_TD_JSON or SQL_TD_WJSON depending on the session character set. The JSON data type is only exposed if LOB support is enabled.

The following table shows how JSON will be exposed by SQLGetTypeInfo using a non-Unicode session. Note that the data type is the "key" passed to SQLGetTypeInfo, thus two rows are returned for SQLGetTypeInfo(SQL_LONGVARCHAR).

Type Name	Data Type	Column Size	Create Params	TDODBC_DATA_TYPE
CLOB	SQL_LONGVARCHAR	2097088000	Max length	SQL_LONGVARCHAR
JSON	SQL_LONGVARCHAR	16776192	Max length	SQL_TD_JSON (18004)

In a Unicode session, the table looks like the following:

Type Name	Data Type	Column Size	Create Params	TDODBC_DATA_TYPE
CLOB() CHARACTER SET UNICODE	SQL_WLONGVARCHAR	1048544000	Max length	SQL_WLONGCHAR

Type Name	Data Type	Column Size	Create Params	TDODBC_DATA_TYPE
JSON() CHARACTER SET UNICODE	SQL_WLONGVARCHAR	8388096	Max length	SQL_TD_WJSON (18005)

Note:

> JSON will be returned as NULLABLE by SQLGetTypeInfo. This is the same as is returned for XML. CLOB and BLOB data types are still returned as non-NULLABLE.

DATASET Data Type

The DATASET data type is a new Complex Data Type (CDT) supported by Version 16.0 of the Teradata database that provides a mechanism to store self-describing data. Self-describing data contains both raw data and matching schema. A schema describes how the data is structured. The schemas do not need to be predefined, for example, during table creation time. Schemas can be defined just before inserting the data, or they can be packaged together with the data as data containers; this data architecture can loosely be called *late binding*.

In the initial 16.0 release, the adopted storage format of a DATASET data type is the Apache Avro format. Starting from 16.20, in addition to Avro, it supports CSV storage format as well. DATASET data can be inserted or imported into a DATASET column in the Avro binary format or in CSV format.

The name `SQL_TD_DATASET_AVRO` and `SQL_TD_DATASET_CSV` is defined in the `tdsql.h` file.

DATASET data can be retrieved as Avro binary or as CSV format and can be converted to JSON format.

SQLGetTypeInfo Result Set

The following table shows how the DATASET data type returns in selected columns from the result set returned by SQLGetTypeInfo.

Type Name	Data Type	Column Size	Create Params	TDOBC_DATA_TYPE
DATASET STORAGE FORMAT AVRO	SQL_TD_DATASET_AVRO (18006)	2097088000	<null>	SQL_TD_DATASET_AVRO (18006)

SET TRANSFORM GROUP FOR TYPE Statement

The SET TRANSFORM GROUP FOR TYPE feature is implemented to support changing the active transform for Teradata CDTs that have multiple transforms at a session level. This feature allows the user to use the appropriate transform without having to create separate USER account with different transform settings.

A new DDL statement SET TRANSFORM GROUP FOR TYPE is added. The syntax follows:

```
<set transform group statement> ::= SET TRANSFORM GROUP FOR TYPE <cdt name>
<transform group name>
```

For more information on this feature, refer to *SQL Data Definition Language - Syntax and Examples* (B035-1144).

Examples

```
SET TRANSFORM GROUP FOR TYPE ST_GEOMETRY TD_GEO_VARCHAR;
```

```
SET TRANSFORM GROUP FOR DATASET STORAGE FORMAT AVRO TD_DATASET_AVRO_VARBYTE;
```

```
SET TRANSFORM GROUP FOR TYPE JSON CHARACTER SET LATIN TD_JSON_VARCHAR;
```

Usage

- Currently, the feature supports the following complex data types CDTs as they have multiple transform groups:
  - JSON
  - XML
  - ST_GEOMETRY
  - DATASET

  Note:

  The only time JSON and XML uses transforms are when in Field, Record, or Indicator modes. For MultipartRecord mode and External JSON/XML data transfer, transforms are not used.

- This feature cannot be applied to user-defined types.
- In the same session, this statement can be executed multiple times for the same CDT to switch to different transforms.
- Any transform group change for a CDT using SET TRANSFORM GROUP FOR TYPE should occur before a statement's preparation or after the prepared statement's execution. The transform group for a CDT cannot be changed between a statement's

preparation and its execution as it would lead to undefined behavior. The ODBC driver will throw an error message (mentioned in Error Message) in this scenario.

- If the logged-on USER already has transform settings, the SET TRANSFORM GROUP FOR TYPE statement will modify the transform setting only for the current session.

Error Message

Below is the error message that the ODBC driver returns for the scenario described in Usage.

Error Message	Description
`Error occurred as a SET TRANSFORM GROUP FOR TYPE statement was executed between PREPARE and EXECUTE.`	The driver returns this error on the execution of a prepared statement if a SET TRANSFORM GROUP FOR TYPE statement has been executed between the preparation and execution of this statement.

NoPI Tables

Tables with no primary index (NoPI tables) are supported in ODBC Driver for Teradata. NoPI tables are included in the result set from SQLTables with the TABLE_TYPE “TABLE” as are tables with primary indexes.

A NoPI table can be recognized by the absence of a row for the primary index in the result set from SQLStatistics (no rows with index TYPE equal to SQL_INDEX_CLUSTERED).

Trusted Sessions

Trusted Sessions provides more security to applications that interface between users and the database, especially in cases where users can submit their own SQL query commands. It prevents a user from possibly submitting "SET QUERY_BAND" commands to change a proxy user. Trusted sessions are supported in Teradata database 13.10 or later.

The database does not allow SET QUERY_BAND SQL to set/change a proxy user on a session having GRANT CONNECT THROUGH privilege with the ‘WITH TRUST_ONLY’ option, unless it is submitted as a trusted SQL request. The ODBC Driver submits an SQL as trusted only when an SQL_ATTR_TRUSTED_SQL(13010) statement attribute is set by calling SQLSetStmtAttr() with a value SQL_TRUE. Immediately after SQLExecute() or SQLExecDirect() is called, SQL_ATTR_TRUSTED_SQL will be reset to the default value SQL_FALSE.

For more information on the GRANT CONNECT THROUGH privilege, please refer the database manual *SQL Data Control Language* (B035-1149).

The following is sample code to send a trusted SQL:

```
……….
……….
RETCODE result;

result = SQLSetStmtAttr(StatementHdl, SQL_ATTR_TRUSTED_SQL, (SQLPOINTER)SQL_TRUE,
SQL_IS_UINTEGER);
if (SQL_SUCCESS == result)
{
        / After SQL execution, the SQL_ATTR_TRUSTED_SQL attribute value will be
reset to SQL_FALSE
        // to prevent any further use of SET QUERY_BAND until you change the
attribute to SQL_TRUE once again.

        result = SQLExecDirect(yourStatementHdl, (SQLTCHAR *) "SET
QUERY_BAND='proxyuser=user1;'FOR SESSION; ", SQL_NTS);vt
        if (SQL_SUCCESS != result)
        {
                // Handle error
        }
}
else
{
        // Handle error
}
```

Restrictions

The following section describes current known restrictions for the use of ODBC Driver for Teradata.

Multi-Statement SQL Requests

ODBC Driver for Teradata permits multi-statement SQL requests. However, Teradata Database enforces the restriction of only one Data Definition Language (DDL) statement per request, and it must be the last statement of the request.

Note:

A Call procedure request in the new driver cannot be in a multi-statement request.

Teradata String Constants

Teradata string constants cannot exceed 255 characters. To handle strings with length greater than 255 characters, you must use the ODBC parameter passing model through SQLBindParameter().

Teradata Column Limitation

If the number of columns in the result set of a SQL Query exceeds the Teradata Database limit of 2048, then ODBC Driver for Teradata returns the following error:

```
"Number of Columns Exceed the Database Limit"
```

WITH Clause

Teradata WITH...BY clauses on SELECT statements are not supported.

ODBC Driver for Teradata ignores aggregated results, returned from Teradata Database.

Asynchronous Operation

The purpose of allowing multiple statements per connection is for the user to retrieve multiple result sets from a single connection. The application must allow the asynchronously executing function to complete before switching to another statement on the same connection (hdbc) and executing another function.

For multiple statements to execute in parallel asynchronously, the application must open a separate connection for each statement.

This operation is consistent with the “Executing SQL Statements” chapter description in the Release 2 *ODBC Programmer’s Reference* specification on “Executing Functions Synchronously” (note 3).

ANSI Migration Issues

Transaction Semantics

Teradata and ANSI transactions are defined differently.

Data Truncation

In previous releases, Teradata did not generate errors when truncating character data during the process of character string assignment. In ANSI mode, character truncation always generates an error.

Duplicate Rows

ANSI allows duplicate rows. Duplicate rows are possible only if the table does not contain a primary key or any columns with unique constraints. In previous releases of Teradata, duplicate rows generated from an INSERT … SELECT statement were ignored and no error or warning was generated. If a table permits duplicate rows, the duplicate rows are inserted and an error is generated.

Updatable Cursors

ODBC Driver for Teradata does not implement the Teradata updatable cursor facility, because certain aspects of the Teradata ANSI transaction modes are incompatible with the ODBC manual auto commit modes. Applications can be serviced through the ODBC Cursor library. ODBC Driver for Teradata supports read-only and forward-only cursors.

Case Sensitivity

Character data in ANSI is case specific. The ANSI UPPER function should be used for case blind character comparisons.

Configuration Characteristics

Database Password Expiration

ODBC Driver for Teradata supports the Teradata Database password expiration security feature. When ODBC Driver for Teradata detects that the user database password has expired during a connect to a data source on Windows, a dialog box will display prompting the user for a new password.

If a new password is specified, ODBC Driver for Teradata then sends a message to Teradata Database setting the user password in the database to the new value.

ODBC Driver for Teradata will not change any password stored in the data source entry. Even though the data source entry will not contain the new password, the user will be logged into Teradata Database. If the user disconnects and then tries to reconnect with the same data source, the connection will fail with an “invalid password” message, since the data source will still contain the old password value. The user must manually change the value of the password using the ODBC Administrator.

SQL Considerations

Newname in SELECT Statement

In the select-list in SELECT statements, each expression can be followed by [AS] newname, where the AS keyword is optional, and newname is a name-title for the column. The newname can optionally be enclosed in double quotes (as can all names). For more information, see Teradata ODBC Specific Comparison Operators (deprecated in 14.10).

This syntax is compatible with X/OPEN™ SQL (X/Open requires the AS keyword), ANSI SQL 1992, Microsoft Access SQL, SyBase/SQL Server, and several others.

In addition, for better SQL Server compatibility, the newname can be enclosed in single quotes instead of double quotes, although the use of single quotes should be discouraged since it is not ANSI-compliant. We recommend that you do not use Teradata TITLE or NAMED clauses, and instead use this syntax when possible.

ANSI Comments in SQL Requests

The ANSI comment starting with two dashes and ending at end-of-line can be used in SQL requests. The comment is removed by the ODBC parser before the request is sent to Teradata Database.

DSN Settings for Third-Party Applications

ODBC Driver for Teradata has specific application uses defined for available DSN settings. The following table lists the DSN settings and their specific application use.

ODBC.INI Setting Name	**Name in DSN setup dialog**	**Specific Application Use**

AccountString=<account>
Or
Account=<account>
	Account String	–

CharacterSet=<*charset name*>
Or
CharSet=<*charset name*>
	Session Character Set	–

DateTimeFormat=[A|I]AA	DateTime Format	
DefaultDatabase=<*database-name*> Or	Default Database	–

ODBC.INI Setting Name	Name in DSN setup dialog	Specific Application Use
Database=<*database-name*>		

DontUseHelpDatabase=[*Yes*|*No*]
Or
DontUseHelpDB=[*Yes*|*No*]
	No HELP DATABASE	–

EnableExtendedStmtInfo=[*Yes*|*No*]	Enable Extended Statement Information	–
EnableReadAhead=[*Yes*|*No*]	Enable read-ahead	–
EnableReconnect=[*Yes*|*No*]	Enable Reconnect	–
EnableUDFUpload=[*Yes*|*No*]	Enable Client Side UDF Upload	–
IANAAppCodePage=<ODBC application code page>	–	This is set for application-specific code page number when using Unicode

IgnoreODBCSearchPattern=[*Yes*|*No*]
Or
IgnoreSearchPat=[*Yes*|*No*]
	Ignore Search Patterns	–

IntegratedSecurity=[*Yes*|*No*]
Or
UseIntegratedSecurity=[*Yes*|*No*]
	Use Integrated Security	–

–	Log Error Events	–
LoginTimeout=<*integer0*>	Login Timeout	Increase the default 20 seconds to higher value for troubleshooting purposes only. This is a

ODBC.INI Setting Name	Name in DSN setup dialog	Specific Application Use
		temporary workaround.
MaxRespSize=<*integer*=<16775168>	Maximum Response Buffer Size	The default is 64K; for large result sets, this value can be increased up to 16 MB.

MechanismName=<*MechanismName*>
Or
Authentication=<*MechanismName*>
	Mechanism	–

MechanismKey=<*Value*>
Or
AuthenticationParameter=<*Value*>
	Parameter	–

NoScan=[*Yes*|*No*]	Disable Parsing	–
PrintOption=[*N*|*P*]	ProcedureWithPrintStmt	–
retryOnEINTR	Retry system calls (EINTR)	KXEN
–	Return Empty string in CREATE_PARAMS columns for SQL_TIMESTAMP	MS Access
ReturnGeneratedKeys=<value>	Return Generated Keys	–
–	Return max. CHAR/ VARCHAR length as 32K	MS Access
SessionMode=[*Teradata*|*ANSI*]	Session Mode	Set to ANSI to resolve application getting too many end transaction messages.
SplOption=[*Yes*|*No*]	ProcedureWithSPLSource	–
TCPNoDelay=[*Yes*|*No*]	Use TCP_NODELAY	–

ODBC.INI Setting Name	Name in DSN setup dialog	Specific Application Use
Translation DLL=<*path*>	Translation DLL Name	–
Translation Option=<*option*>	Translation Option	–
TDMSTPortNumber=<*integer*>	TDMST Port Number	Set to another port number if port 1025 is already used. By default, the Teradata Gateway is listening to port 1025.
UDFUploadPath=<*path*>	UDF Upload Path	–
USE2XAPPCUSTOMCATALOGMODE=[*Yes*|*No*] Or 2XAPPCUSTOMCATALOGMODE=[*Yes*|*No*]	Enable Custom Catalog Mode for 2.x Applications	MS Excel Importing data into an Excel 2000 spreadsheet by SQL query (using Data>>Get External Data>>New Database Query)

UseColumnNames=[*Yes*|*No*]
Or
DontUseTitles=[*Yes*|*No*]
	Use Column Names	Crystal Reports

UseDataEncryption=[*Yes*|*No*]
Or
DataEncryption=[*Yes*|*No*]
	Enable Data Encryption	–

–	Use DATE data for TIMESTAMP parameters	MS Access and other applications that are using Microsoft Access Jet databases

ODBC.INI Setting Name	Name in DSN setup dialog	Specific Application Use
–	Use NULL for Catalog Name	–
–	Use Regional Settings for Decimal Symbol	–
UseXViews=[*Yes*|*No*]	Use X Views	–

A

odbc.ini File Examples

Overview

This appendix contains examples of release-independent 32-bit and 64-bit `odbc.ini` files created by the installation program.

Release-Independent 32-bit odbc.ini File Example

```
[ODBC]
# For Data Direct to load its error messages
# Data Direct Driver Manager looks for the messages here:
# "/opt/teradata/client/16.20/locale/xx_xx/LC_MESSAGES/"
InstallDir=/opt/teradata/client/ODBC_32
Trace=no

[ODBC Data Sources]
Teradata ODBC DSN=Teradata Database ODBC Driver 16.20

[Teradata ODBC DSN]
# This key is not necessary and is only to give a description of the data source.
Description=Teradata Database ODBC Driver 16.20

# Driver: The location where the ODBC driver is installed to.
Driver=/opt/teradata/client/ODBC_32/lib/tdataodbc_sb32.so

# Required: These values can also be specified in the connection string.
DBCName=
UID=
PWD=

# Optional
AccountString=
CharacterSet=ASCII
DatasourceDNSEntries=
DateTimeFormat=AAA
DefaultDatabase=
DontUseHelpDatabase=0
DontUseTitles=1
EnableExtendedStmtInfo=1
```

```
EnableReadAhead=1
IgnoreODBCSearchPattern=0
LogErrorEvents=0
LoginTimeout=20
MaxRespSize=65536
MechanismName=
NoScan=0
PrintOption=N
retryOnEINTR=1
ReturnGeneratedKeys=N
SessionMode=System Default
SplOption=Y
TABLEQUALIFIER=0
TCPNoDelay=1
TdmstPortNumber=1025
USE2XAPPCUSTOMCATALOGMODE=0
UseDataEncryption=0
UseXViews=0
```

Release-Independent 64-bit odbc.ini File Example

The installation program creates the release-independent 64-bit `odbc.ini` file in `/opt/teradata/client/ODBC_64`.

```
[ODBC]
InstallDir=/opt/teradata/client/ODBC_64
Trace=0
TraceDll=/opt/teradata/client/ODBC_64/lib/odbctrac.so
TraceFile=/usr/joe/odbcusr/trace.log
TraceAutoStop=0

[ODBC Data Sources]
default=tdata.so
testdsn=tdata.so

[testdsn]
Driver=/opt/teradata/client/ODBC_64/lib/tdataodbc_sb64.so
Description=Teradata running Teradata V1R5.2
DBCName=208.199.59.208
LastUser=
Username=
Password=
Database=
DefaultDatabase=
[default]
Driver=/opt/teradata/client/ODBC_64/lib/tdataodbc_sb64.so
Description=Default DSN is Teradata 5100
DBCName=208.199.59.208
LastUser=
Username=
Password=
Database=
DefaultDatabase=
```

Release-Independent odbc.ini File for Apple OS X Example

```
[ODBC]
Trace = 1
TraceFile = /tmp/dmtrace.log

[ODBC Data Sources]
testdsn = teradata

[testdsn]
Driver = /Library/Application Support/Teradata/Client/ODBC/lib/tdataodbc_sbu.dylib
DBCName = sdll4771.labs.teradata.com
Username = testuser
```

B

odbcinst.ini File Examples

Overview

This appendix contains examples of release-independent 32-bit and 64-bit `odbcinst.ini` files created by the installation program.

Note:

This appendix is not applicable for Apple OS X.

Release-Independent 32-bit odbcinst.ini File Example

```
[ODBC Drivers]
Teradata Database ODBC Driver 16.20=Installed

[Teradata Database ODBC Driver 16.20]
Description=Teradata Database ODBC Driver 16.20
Driver=/opt/teradata/client/ODBC_32/lib/tdataodbc_sb32.so
```

Release-Independent 64-bit odbcinst.ini File Example

The installation program creates the release-independent 64-bit `odbcinst.ini` file in `/opt/teradata/client/ODBC_64`.

```
[ODBC DRIVERS]
Teradata=Installed

[Teradata]
Driver=/opt/teradata/client/ODBC_64/lib/tdataodbc_sb64.so
APILevel=CORE
ConnectFunctions=YYY
DriverODBCVer=3.51
SQLLevel=1
```

Release-Dependent 32-bit odbcinst.ini File Example

```
[ODBC DRIVERS]
Teradata Database ODBC Driver 16.20=Installed

```

```
[Teradata Database ODBC Driver 16.20]
Driver=/opt/teradata/client/16.20/odbc_32/lib/tdataodbc_sb32.so
APILevel=CORE
ConnectFunctions=YYY
DriverODBCVer=3.51
SQLLevel=1
```

Release-Dependent 64-bit odbcinst.ini File Example

```
[ODBC DRIVERS]
Teradata Database ODBC Driver 16.20=Installed

[Teradata Database ODBC Driver 16.20]
Driver=/opt/teradata/client/16.20/odbc_64/lib/tdataodbc_sb64.so
APILevel=CORE
ConnectFunctions=YYY
DriverODBCVer=3.51
SQLLevel=1
```

ODBC Options Examples

Overview

This appendix provides example code from the ODBC options sections of the `odbc.ini` files. The examples cover both 32-bit and 64-bit systems for all operating systems.

ODBC Options Section

The ODBC Options section [ODBC] of the `odbc.ini` file specifies the ODBC installation directory and indicates whether Driver Manager tracing is enabled. When tracing is enabled, all ODBC function calls made from an application can be logged to a specified trace file.

ODBC Options: IBM AIX, Linux, Solaris

```
[ODBC]
InstallDir=/opt/teradata/client/ODBC_64/
Trace=1
TraceDll=/opt/teradata/client/ODBC_64/lib/odbctrac.so
TraceFile=/usr/odbcusr/joe/trace.log
TraceAutoStop=1
```

ODBC Options: Apple OS X

```
[ODBC]
Trace=1
TraceFile=/tmp/trace.log
TraceAutoStop=1
```

Data Source Specification Options Example

The following code sample is an example of the option entries in the Data Source Specification section named `[financial]`.

```
[financial]
Driver=/opt/teradata/client/ODBC_64/lib/tdataodbc_sb64.so
Description=Teradata 5550H running Teradata Database
DBCName=123.45.67.10
DBCName2=123.45.67.11
DBCName3=123.45.67.12
```

```
Username=odbcadm
Password=
Database=
DefaultDatabase=sales
```

DSN Tracing Options Example

Full tracing turned on in the 64-bit version of the `teradata.teradataodbc.ini` file:

```
[Driver]
ErrorMessagesPath=../odbc_64/ErrorMessages
LogLevel=6
LogPath=~/mylogs
```

Deprecated SQL Transformations

Overview

The new Teradata 16.20 driver now deprecates the SQL Transformations listed in this section in order to comply better with standard SQL, standard ODBC and the Teradata database.

Pseudo Type Mappings (deprecated in 16.20)

A number of pseudo SQL type names are deprecated as of 16.20. The names are not used in Teradata SQL and are not required in ODBC. They were introduced in the Teradata ODBC driver to facilitate migration of early applications from non-Teradata environments.

The names could be used in table column definitions and in certain type casts. The ODBC parser changed the pseudo-type to a suitable Teradata SQL data type. The following table summarizes the deprecated pseudo-types.

Instead of this pseudo SQL data type	Use the following Teradata SQL Data Type
BINARY[(n)]	BYTES[(n)]
BIT	BYTEINT FORMAT ‘9’
DOUBLE	FLOAT or DOUBLE PRECISION
LONG VARBINARY	VARBYTE(32000)
TINYINT	BYTEINT
VARBINARY(n)	VARBYTE(n)

Standard Type Mappings (deprecated in 16.20)

The following transformations of standard Teradata types are deprecated as of 16.20.

Teradata Data Type	Legacy Transformation by Teradata ODBC
BIGINT	When used with early databases that did not support BIGINT the Teradata ODBC driver would map BIGINT to DECIMAL(18,0).

Teradata Data Type	Legacy Transformation by Teradata ODBC
	All Teradata database releases supported by ODBC 15.10 now support the native BIGINT type and this mapping is obsolete.
DATE	ANSI Date mode is the default and is what an application should use. In Integer Date mode, the DATE type, when used in a column definition or type cast will be replaced by: `DATE FORMAT 'YYYY-MM-DD'` The replacement occurs only if no FORMAT phrase has explicitly been used. The Integer Date mode is supported by the database and the default database format is 'YY/MM/DD', which is different from the standard ODBC and ANSI format. An application that uses Integer Date mode and relies on ODBC to set the date FORMAT in the SQL text previously could use the EnableLegacyParser option, but this has been deprecated. Your application needs to use ANSI Date mode (preferred default) or set explicit FORMATs.
TIME[(n)]	If Integer Time mode is used then the TIME type when used in a column definition or type cast will be replaced by: `INTEGER FORMAT '99:99:99'` Any precision is ignored. The Integer Time mode is an ODBC TIME type implementation for database versions not supporting true ANSI TIME types. Integer Time mode is deprecated in ODBC 15.00 and is not recommended for use.

IN-List Expansion (deprecated in 16.20)

The parser transforms an IN-list containing ODBC Date, Time, Timestamp, or Interval literal escapes into a sequence of comparisons to work around a limitation in the database where only constants and not expressions are allowed in the in-list. This transformation is deprecated in 16.20.

For example, in Integer Date mode the ODBC Date literal `{d '2008-11-28'}` is expanded to:

```
('2008-11-28' (DATE, FORMAT 'YYYY-MM-DD'))
```

The resulting expansion is regarded as an expression by the database and is not allowed in the IN-list. The ODBC parser therefore transforms a construct like:

```
... x IN ({d '2008-11-28'}, {d '2009-12-29'})
```

into:

```
... (x = ('2008-11-28' (DATE, FORMAT 'YYYY-MM-DD'))
```

Or

```
x = ('2009-12-29' (DATE, FORMAT 'YYYY-MM-DD')))
```

With the new parser the ODBC date literal escape sequence is expanded as DATE 'YYYY-MM-DD'. For example in Integer Date mode the ODBC Date literal {d '2008-11-28'} is expanded to: DATE '2008-11-28'. New parser in-list expansion of Integer-Date works fine, but does will work in the case of Integer-Time, because the resulting Native-Syntax is an expression. The workaround is to switch to ANSI time.

ODBC-Style Named Indexes (deprecated in 15.00)

The ODBC driver extends the SQL grammar with alternate CREATE INDEX and DROP INDEX statements, allowing an index created using the alternate CREATE INDEX syntax to be dropped using the alternate DROP INDEX syntax, by just specifying the index name and not also the associated table name and either the column list or index name.

This feature was originally designed for earlier versions of ODBC, and will be deprecated in a future release. The alternate syntax for both CREATE INDEX and DROP INDEX is not standard and not portable. The recommendation is to use the Teradata SQL database syntax for these statements as described in *SQL Data Definition Language* (B035-1144).

CALL to EXEC Conversion (deprecated in 14.10)

By default, CALL statements are considered as the SQL for MACRO execution, and the statements are converted to EXEC with the ODBC driver. The CallSupport or DisableCALLToEXECConversion keywords are currently available to disable conversion of CALL statements to EXEC, so the statements are handled as CALL statements for stored procedure execution.

This feature was originally designed for earlier versions of ODBC, and will be deprecated in a future release.

The default setting has been changed from no to yes. It is strongly advised to retain the default CallSupport or DisableCallToExecConversion setting of yes for best compliance with the ODBC specification, and to prepare for the removal of this option in a future release.

Teradata ODBC Specific Comparison Operators (deprecated in 14.10)

The ODBC Driver for Teradata support for mapping != and ~= comparison operators to the standard SQL operator <> has been deprecated. Queries that use these operators will not be successful using other Teradata client access products and for this reason the Teradata

ODBC support of these operators is deprecated. Furthermore, this alternative will be unavailable in ODBC Driver for Teradata 16.00.

It is strongly advised to not use these Teradata ODBC-specific comparison operators, and to change existing SQL scripts to avoid the usage of these operators.

For example:

`SELECT p FROM t WHERE p <> 7 instead of SELECT p FROM t WHERE p != 7`

ODBC Scalar Functions Outside Escape Sequences (deprecated in 14.10)

The ODBC Driver for Teradata support for ODBC scalar functions outside escape sequences has been deprecated. This is a Teradata specific extension to the ODBC standard and its use is deprecated for compatibility reasons. Support will be discontinued in order to facilitate reuse of SQL between products. Support for this feature will be unavailable in ODBC Driver for Teradata 16.00.

ODBC scalar functions should always be used within escape sequences. For example:

```
SELECT {fn DAYOFWEEK(delivery_date)} FROM order_table;
```

ODBC Sample Program Usage Information

Overview

This appendix provides high-level information about the ODBC installation validation script, tdxodbc (also referred to as the ODBC sample program) that is installed as part of ODBC Driver for Teradata.

Location of the ODBC Sample Program

Upon successful execution installation of the ODBC Driver for Teradata, a sample program (tdxodbc) is available in the following location, depending on the operating system.

Platform	Location of tdxodbc	Example
Windows 32-bit	<InstallDir>\Teradata\Client \<*version*>\Bin	C:\Program Files (x86)\Teradata \Client\16.20\Bin
Windows 64-bit	<InstallDir>\Teradata\Client \<*version*>\Bin	C:\Program Files\Teradata\Client \16.20\Bin
UNIX/Linux 32-bit	<InstallDir>/teradata/client/ <*version*>/bin	/opt/teradata/client/16.20/bin
UNIX/Linux 64-bit	<InstallDir>/teradata/client/ <*version*>/bin	/opt/teradata/client/16.20/bin
Apple OS X	/Library/Application Support/ teradata/client/<*version*>bin	/Library/Application Support/ teradata/client/16.20/bin

Sample Output

The ODBC sample program (tdxodbc) provides usage information with the "-h" option. The command "tdxodbc -h" displays usage of the sample program with details of various arguments that can be sent to this executable. For example:

```
# ./tdxodbc -h

USAGE: tdxodbc
tdxodbc -c SQLConnect [-d <dsn>] [-u <user name>] [-p <password>]
tdxodbc -c SQLDriverConnect [ -C <connection string> | [[-S <server name>] [-u <user
name>] [-p <password>] [-T <driver name>]] ] [-t [-n <num of connections>]]
OPTIONS:
-c <connect type> [SQLConnect | SQLDriverConnect] default:SQLConnect
-d <dsn>
-u <user name>
-p <password>
-C <connection string> example: -C "DRIVER=Teradata; DBCNAME=MYCOP1; UID=dbc;
PWD=dbc;"
-T <driver name> default: -T Teradata
-S <server name> example: -S MYCOP1
-h Display usage
-t connect test only
-n <num of connections> applicable only with -t
```

Executing tdxodbc to Validate Installation

You can validate ODBC Driver installation by executing tdxodbc to connect to a database and submit sample SQLs for execution.

Windows

Sample Command

```
tdxodbc -c SQLDriverConnect -C "DSN=testdsn;UID=dbc;pwd=dbc;"
```

Restrictions and Considerations

- Before executing the command, ODBC Administrator needs to be configured with the required DSN. In the sample command, “testdsn” is the configured DSN name.
- After executing the command, it should successfully connect the database and prompt for SQL to be executed.
- You can submit any simple SQL statement such as “select date” to verify that ODBC Driver is properly installed.
- To ensure that the message catalog file is installed properly, you can submit an incorrect SQL statement such as “create table (int)” and verify the error message.
- Exit the program by entering “quit” when prompted for SQL.

Sample Output

```
C:\Program Files (x86)\Teradata\Client\16.20\bin>tdxodbc -c
SQLDriverConnect -C "DSN=testdsn;UID=dbc;pwd=dbc;"

Connecting with SQLDriverConnect("DSN=testdsn;UID=dbc;pwd=dbc;")...

.....ODBC connection successful.

ODBC version        = -03.80.0000-
DBMS name           = -Teradata-
DBMS version        = -16.20.0064 16U.20.00.64-
Driver name         = -TDATA32.DLLDriver
version             = -16.20.00.00-
Driver ODBC version = -03.80-

(type quit to terminate adhoc)
Enter SQL string : quit

'quit' command detected

ODBC connection closed.
```

UNIX/Linux/Apple OS X

On UNIX/Linux platforms the same steps can be followed to verify the installation, except that there is a prerequisite to set the ODBCINI environment variable.

1. Set the ODBCINI variable to a default odbc.ini file that comes as part of ODBC installation.

   Example: export ODBCINI=/opt/teradata/client/ODBC_32/odbc.ini
2. Modify this odbc.ini file with the required DBCName, etc.
3. After setting ODBCINI, follow the sample steps provided above (in case of Windows) to verify successful connection and executing sample SQLs.

More details about setting ODBCINI can be found in the *ODBC User Guide*.

MultiVersion Support

Overview

Multiple Version Support is a new feature that allows the coexistence of multiple driver versions on a system at the same time. The following table lists a series of installation scenarios.

Installation Scenario	Existing	Install	Remarks
Upgrade Windows	14.10.00.00 and greater up to 16.20	16.20	Pre-14.10 releases have to be removed before installing 14.10.00.00 and later versions.
Upgrade UNIX-with provided setup (.setup.sh)	Native installer upgrades tdodbc1620-16.20.00.00 to tdodbc1620-16.20.00.01	16.20 and later	The native installer cannot upgrade from tdodbc1510 to tdodbc1620 since the package name is different.
Fresh Install	N/A	16.20	No driver already exists on the system and then 16.20 is installed.
Side-by-Side	15.10.01.x and/or 16.10	16.20	15.10.01.xx or 16.10 already exists on the system and then 16.20 is installed in addition to 16.10.

Installation Scenario	Existing	Install	Remarks
			Note: Existing applications that use DSN-less connections with the driver name Teradata may connect with the 16.10 driver. If you want to connect with the 16.20 driver then you may have to either uninstall 16.10 and create a custom driver name or change your application to use a different connection string. Note: You can have 15.10.01x and/or 16.10 and 16.20 all installed side-by-side.
Side-by-Side	16.20	15.10.01.x and/or 16.10	15.10.01.x or later is installed on the system and then 16.10 is installed in addition to 16.20. Note: You can have 15.10.01x and/or 16.10 and 16.20 all installed side-by-side.

UNIX Operating System

The active TTU of the system at any given time is the version of the driver that was most recently installed on the system. For example, if 16.10 is installed and then 16.20 is installed, then 16.20 is the active TTU.

There are three different choices for `odbc.ini` and `odbcinst.ini`.

1. Inside the release independent directories. By default, those are `/opt/teradata/client/ODBC_32/` and `/opt/teradata/client/ODBC_64/`.

2. Inside the release specific directories. For 16.20, that would be **`/opt/teradata/client/16.20/odbc_32/`** and **`/opt/teradata/client/16.20/odbc_64/`**.
3. Custom file location. The user can create their own **`odbc.ini`** and **`odbcinst.ini`** files and put them in any location on the system and set the ODBCINI and ODBCINSTINI environment variables accordingly.

Inside the release independent directories (i.e., `/opt/teradata/client/ODBC_32/` and `/opt/teradata/client/ODBC_64/`) there will be symbolic links for locale, lib, and include which will be pointing to their corresponding directories for the active TTU.

16.20 as the Active TTU

If 16.20 is the active TTU, then we would have the following:

`/opt/teradata/client/ODBC_32/locale` -> `/opt/teradata/client/16.20/odbc_32/locale`

`/opt/teradata/client/ODBC_32/lib` -> `/opt/teradata/client/16.20/lib`

`/opt/teradata/client/ODBC_32/include` -> `/opt/teradata/client/16.20/include`

`/opt/teradata/client/ODBC_64/locale` -> `/opt/teradata/client/16.20/odbc_64/locale`

`/opt/teradata/client/ODBC_64/lib` -> `/opt/teradata/client/16.20/lib64`

`/opt/teradata/client/ODBC_64/include` -> `/opt/teradata/client/16.20/include`

Inside `odbc.ini` and `odbcinst.ini`, all file paths are given in terms of release independent paths which use the symbolic links given above.

For example, the driver location given in `/opt/teradata/client/ODBC_32/odbc.ini` is `Driver=/opt/teradata/client/ODBC_32/lib/tdataodbc_sbu.so` which corresponds to `/opt/teradata/client/16.20/lib/tdataodbc_sbu.so` if 16.20 is the active TTU.

Release-Independent odbc.ini File

If the ODBCINI environment variable is set to a release-independent `odbc.ini` file, then the application will be using the active TTU version of the driver. If the desired driver version is not the active TTU version, then you must either make the desired version the active TTU or set the ODBCINI environment variable to an odbc.ini inside the desired version's directory. See example below:

```
export ODBCINI=/opt/Teradata/client/16.20/odbc_32/odbc.ini
```

DSN-Less Connection

If you are using a DSN-Less connection, then you must use a driver name that exists in your odbcinst.ini file. If you are using release independent odbc.ini and odbcinst.ini files then the default driver name to use is ***`Teradata`***. Otherwise, for 16.20 and beyond, if you are using release dependent odbc.ini and odbcinst.ini files then the default driver name is ***Teradata Database ODBC Driver XX.YY*** where XX.YY is the version.

For example, `ODBCINI=/opt/Teradata/client/16.20/odbc_32/odbc.ini` then a successful connection would use a connection string like `"DRIVER={Teradata Database ODBC Driver 16.20};DBCName=156.43.66.95;UID=username;PWD=password;"`.

Creating a Custom Driver Name

If you want to use your own custom driver name, then you will have to modify the `odbcinst.ini` file to include more driver names.

Below is a sample **`odbcinst.ini`** file with a custom driver name for a 16.20 driver. The bold font indicates what is added to the **`odbcinst.ini`** file for the custom driver.

```
[ODBC DRIVERS]
Teradata=Installed
Custom Driver=Installed
[Teradata]
Driver=/opt/teradata/client/16.20/lib/tdataodbc_sb32.so
APILevel=CORE
ConnectFunctions=YYY
DriverODBCVer=3.51
SQLLevel=1
[Custom Driver]
Driver=/opt/teradata/client/16.20/lib/tdataodbc_sb32.so
APILevel=CORE
ConnectFunctions=YYY
DriverODBCVer=3.51
SQLLevel=1
```

```
*** 32bit version: ***
[ODBC Drivers]
Teradata Database ODBC Driver 16.10=Installed

[Teradata Database ODBC Driver 16.10]
Description=Teradata Database ODBC Driver 16.10
Driver=/opt/teradata/client/16.10/lib/tdataodbc_sb32.so



*** 64bit version: ***
[ODBC Drivers]
Teradata Database ODBC Driver 16.10=Installed

[Teradata Database ODBC Driver 16.10]
```

```
Description=Teradata Database ODBC Driver 16.10
Driver=/opt/teradata/client/16.10/lib64/tdataodbc_sb64.so
```

Removing a Custom Driver Name

To remove a custom driver name, you must modify the **`odbcinst.ini`** file and do the reverse of adding a custom driver name. Using the example provided to create a custom driver name, you would remove the lines in bold font to remove the custom driver.

Teradata ODBC Driver Installation

The Teradata ODBC Driver installation creates a set of symbolic links for the DataDirect ODBC Driver Manager. An application may be affected if the symbolic links are pointing to a newer or older version of the driver manager.

There are two ways to determine which driver manager is used.

- Set the driver manager symbolic link manually.

  Such a command might look something like:

  ```
  ln -sf /opt/teradata/client/16.20/lib/libodbc.so /usr/lib/libodbc.so
  ```

  This will affect all applications on the system using ODBC.
- Set the LD_LIBRARY_PATH to the directory containing the **`libodbc.so`** file you want to use.

  Such a command might look something like:

  ```
  export LD_LIBRARY_PATH=/directory/to/driver/manager/
  ```

  This will only apply to the particular session being run.

Refer to the README file for the DataDirect Driver Manager Compatibility.

Apple OS X

Inside the release independent directory (`/Library/Application Support/teradata/client/ODBC`) there is an **`odbc.ini`** file and symbolic links for lib and include which will be pointing to the corresponding directories for the active TTU.

16.20 as the Active TTU

If 16.20 is the active TTU, then we would have:

`/Library/Application Support/teradata/client/ODBC/lib` -> `/Library/Application Support/teradata/client/16.20/lib`

`/Library/Application Support/teradata/client/ODBC/include -> /Library/Application Support/teradata/client/16.20/include`

Inside `odbc.ini`, all file paths are given in terms of release independent paths which use the symbolic links given above.

For example, the driver location given in `/Library/Application Support/teradata/client/ODBC/odbc.ini` is `Driver=/Library/Application Support/teradata/client/ODBC/lib/tdataodbc_sbu.dylib` which corresponds to `/Library/Application Support/teradata/client/16.20/lib/tdataodbc_sbu.dylib` if 16.20 is the active TTU.

System DSNs can be found in the `odbc.ini` files in the default location of `/Library/ODBC/`.

User DSNs can be found in `~/Library/ODBC/`. Alternatively, the ODBCINI environment variable can point to an `odbc.ini` file present in a non-default location.

Release-Independent odbc.ini File

If the ODBCINI environment variable is set to a release-independent `odbc.ini` file, then the application will be using the active TTU version of the driver. If the desired driver version is not the active TTU version then you must either make the desired version the active TTU or set the ODBCINI environment variable to an `odbc.ini` file inside the desired version's directory. See example below:

```
export ODBCINI=/Library/Application\ Support/teradata/client/16.20/odbc/odbc.ini
```

For 16.20 and beyond, the default driver name is ***`Teradata Database ODBC Driver XX.YY`*** where XX.YY is the version. A successful connection may use a connection string like `"DRIVER={Teradata Database ODBC Driver 16.20};DBCName=156.43.66.95;UID=username;PWD=password"`.

Creating a Custom Driver Name

A custom driver name can be created using the instdrv tool. By default, the tool is located in `/Library/Application Support/teradata/client/16.20/odbc/bin/odbcinst.ini` file to include more driver names.

For example, suppose the driver name ***`Teradata`*** is hardcoded in your application and you need to connect using the driver name ***`Teradata`*** rather than the default name of ***`Teradata Database ODBC Driver 16.20`***. Navigate to the directory that contains the instdrv tool and execute the following command:

```
sudo ./instdrv -i "Teradata;Driver=/Library/Application Support/teradata/client/16.20/lib/tdata.dylib;Setup=/Library/Application Support/teradata/client/16.20/lib/TeradataODBCSetup.bundle/Contents/MacOS/TeradataODBCSetup"
```

Below is the resulting odbcinst.ini file with two driver names for the16.20 driver.

The odbcinst.ini for SEN on Mac OS:

```
[ODBC Drivers]
Teradata Database ODBC Driver 16.20=Installed

[Teradata Database ODBC Driver 16.20]
Description=Teradata Database ODBC Driver 16.20
Driver=/Library/Application Support/teradata/client/16.20/lib/tdataodbc_sbu.dylib
Setup=/Library/Application Support/teradata/client/16.20/lib/
TeradataODBCSetup.bundle/Contents/MacOS/TeradataODBCSetup
```

Removing a Custom Driver Name

To remove a custom driver name, you must modify the `odbinst.ini` file and remove the lines pertaining to the driver name you want to remove. Using the example provided to create a custom driver name, you would remove the lines in bold font to remove the driver name Teradata.

Windows

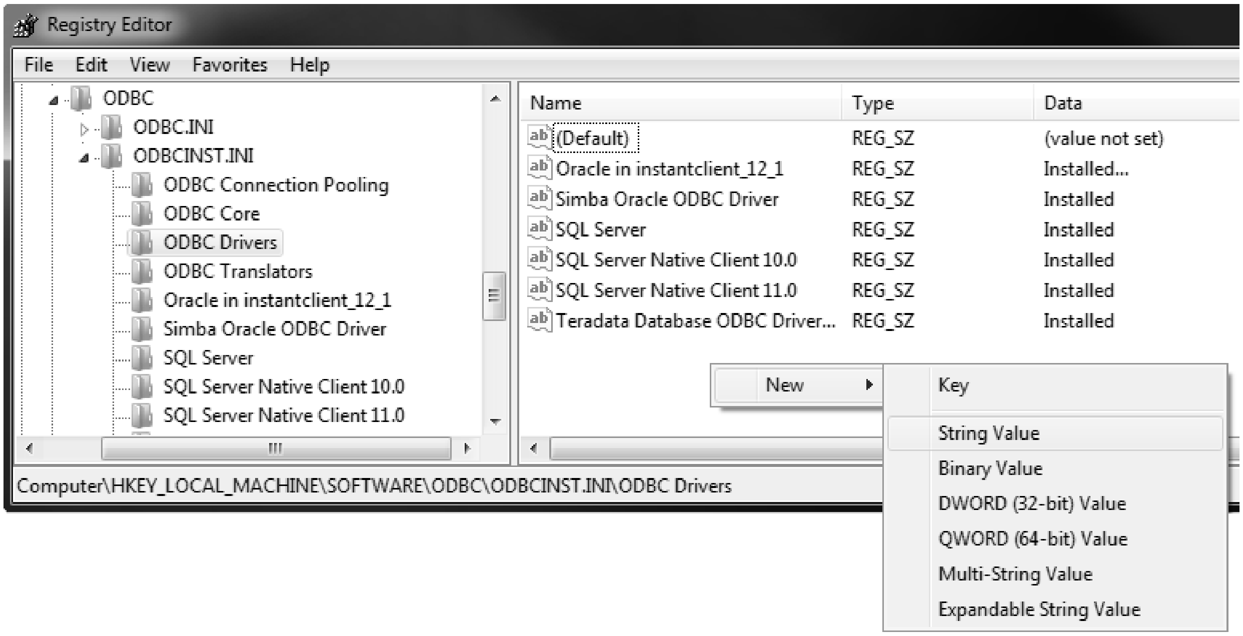
For 16.20 and beyond, the driver name is ***`Teradata Database ODBC Driver XX.YY`*** where XX.YY is the version. For earlier versions the driver name is Teradata. The appropriate driver name must be used for DSN-less connection strings and when creating new DSNs.

Creating a Custom Driver Name

On Windows, custom driver names can be created using the registry.

Suppose the driver name ***`Teradata`*** is hardcoded in your application and you need to connect using the driver name ***`Teradata`*** rather than the default name of ***`Teradata Database ODBC Driver 16.20`***.

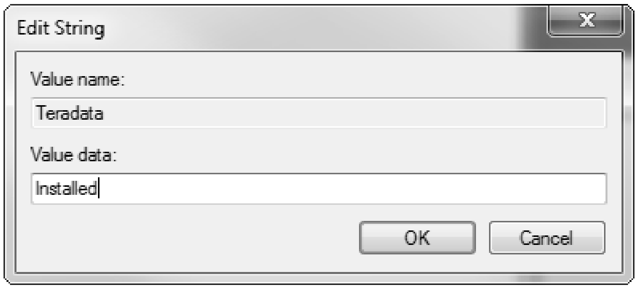
1. Open the registry.
   - For the 32 bit driver navigate to `HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ODBC\ODBCINST.INI`.
   - For the 64 bit driver navigate to `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI`.
2. In the subdirectory `ODBC Drivers`, right-click and select New > String Value and name it `Teradata`.



Name	Type	Data
(Default)	REG_SZ	(value not set)
Oracle in instantclient_12_1	REG_SZ	Installed...
Simba Oracle ODBC Driver	REG_SZ	Installed
SQL Server	REG_SZ	Installed
SQL Server Native Client 10.0	REG_SZ	Installed
SQL Server Native Client 11.0	REG_SZ	Installed
Teradata Database ODBC Driver...	REG_SZ	Installed

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers

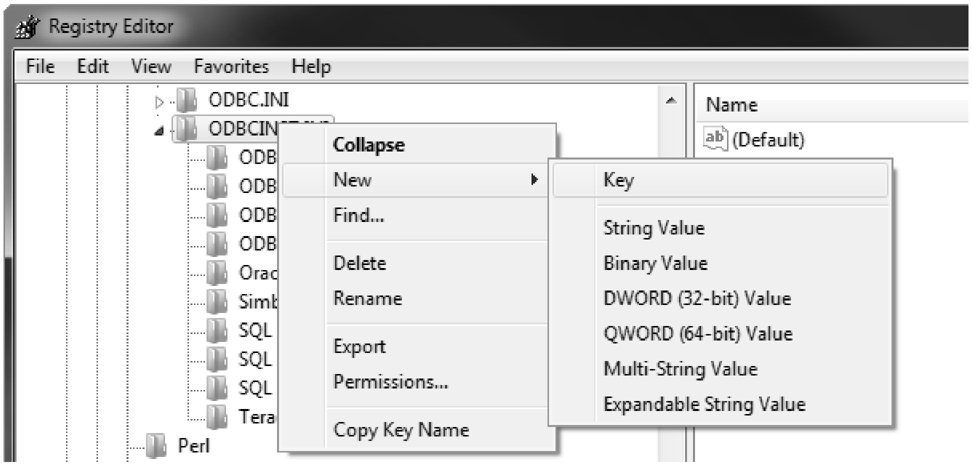
3. Right-click **Teradata** and select **Modify**.
   The **Edit String** dialog box appears.



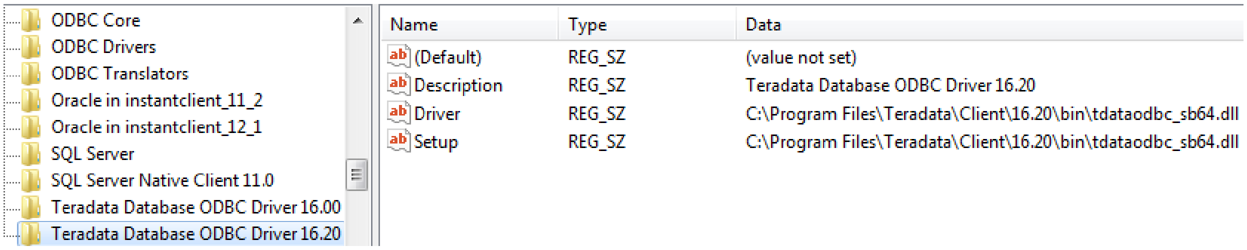
Value name: Teradata

Value data: Installed

4. Type `Installed` in the **Value data** field.
5. Right-click **ODBCINST.INI**, select **New > Key** and name it `Teradata`.



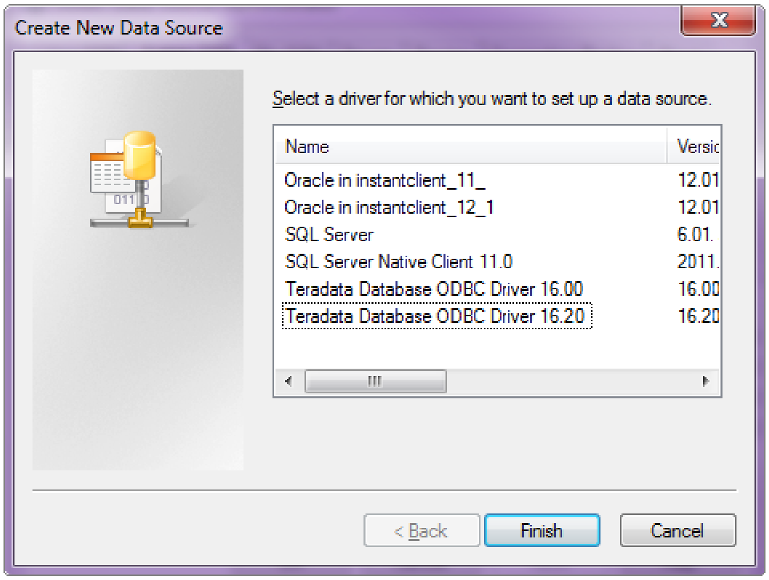
6. For each string value field in **`Teradata ODBC Driver 16.20`**, create a corresponding value in the new custom key, ***`Teradata`***. Under the new key, **`(Default)`**, **`Description`**, **`Driver`**, and **`Setup`** appear as shown in the figure below.



- ODBC Core
- ODBC Drivers
- ODBC Translators
- Oracle in instantclient_11_2
- Oracle in instantclient_12_1
- SQL Server
- SQL Server Native Client 11.0
- Teradata Database ODBC Driver 16.00
- Teradata Database ODBC Driver 16.20

Name	Type	Data
(Default)	REG_SZ	(value not set)
Description	REG_SZ	Teradata Database ODBC Driver 16.20
Driver	REG_SZ	C:\Program Files\Teradata\Client\16.20\bin\tdataodbc_sb64.dll
Setup	REG_SZ	C:\Program Files\Teradata\Client\16.20\bin\tdataodbc_sb64.dll

Now you can create a new DSN with the driver name Teradata.

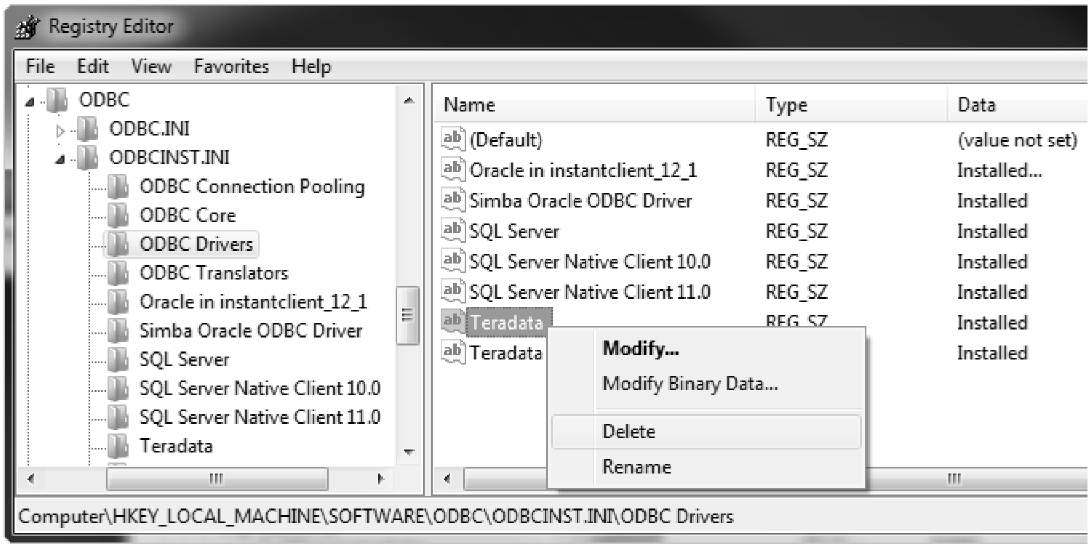


Select a driver for which you want to set up a data source.

Name	Versic
Oracle in instantclient_11_	12.01
Oracle in instantclient_12_1	12.01
SQL Server	6.01.
SQL Server Native Client 11.0	2011.
Teradata Database ODBC Driver 16.00	16.00
Teradata Database ODBC Driver 16.20	16.20

Removing a Custom Driver Name

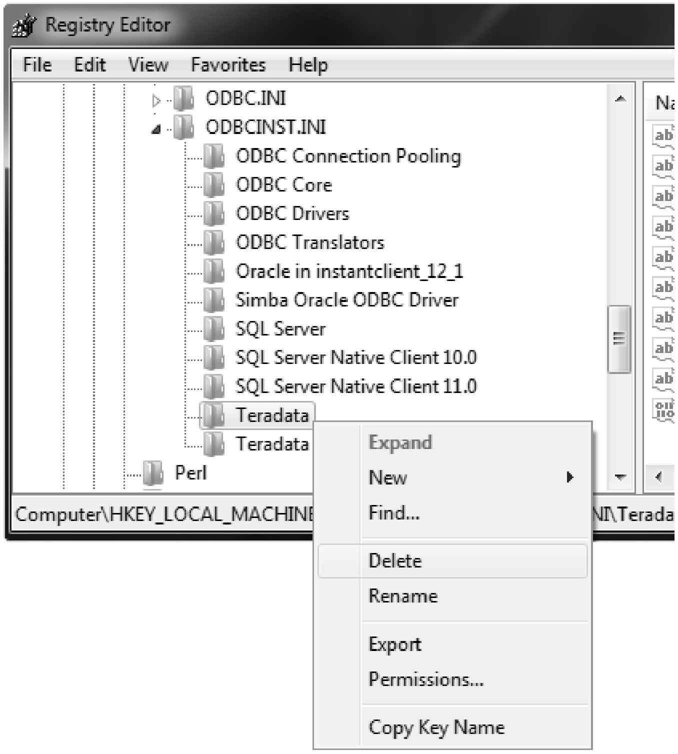
1. Under ODBC Drivers, delete the driver name.



Name	Type	Data
(Default)	REG_SZ	(value not set)
Oracle in instantclient_12_1	REG_SZ	Installed...
Simba Oracle ODBC Driver	REG_SZ	Installed
SQL Server	REG_SZ	Installed
SQL Server Native Client 10.0	REG_SZ	Installed
SQL Server Native Client 11.0	REG_SZ	Installed
Teradata	REG_SZ	Installed
Teradata		Installed

Computer\HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI\ODBC Drivers

2. Delete the subdirectory under ODBCINST.INI for the driver name you are removing.



Computer\HKEY_LOCAL_MACHINE NI\Terada

General Guidelines

- For DSN-less connections, try keeping the Connection string configurable as the driver name will change with each new release.

- Use release-independent directories when you don't expect your application to be negatively affected by installing new versions of the driver.
- Use release-specific directories when you are relying on a specific version of the driver for your application.

New Teradata ODBC Driver Compatibility Reference

Deprecated Features for New Teradata ODBC Driver

The following sections list and describe the deprecated features from the old Teradata ODBC driver not available in the new Teradata ODBC Driver.

Disable Dialog Box Popups

The old driver allowed the user to disable popup dialog boxes by using the configuration options **QuietMode** or **RunInQuietMode**. Popup dialog boxes are not present in the new driver, and these options are not available.

Integer Time

The old driver allowed the user to specify Integer Time in the DSN option DateTime Format or the configuration option **DateTimeFormat**. Integer Time mode has been removed from the new driver for compatibility reasons, and these options cannot be used to specify Integer Time in the new driver.

Bypass State Check Level

The old driver allowed the user to bypass the ODBC state check for open cursors by using the DSN option State Check Level or the configuration option `StCheckLevel`. These options are not available in the new driver.

Disable Prepare

The old driver allowed the user to disable the prepare step for SQLDirectExec statements by using the DSN option Disable Prepare For SQL or the configuration option `SQLWithCommentsOrParenthesis`. These options are not available in the new driver.

Application Catalog Database

The old driver allowed the user to include a specified database in table searches under specific circumstances by using the configuration option `AppCatalogDB`. This option invoked non-standard behavior, and is not available in the new driver.

Disable Asynchronous Operation

The old driver allowed the user to silently ignore attempts to enable asynchronous operation by using the configuration option `DisableAsync`. This option invoked non-standard behavior, and is not available in the new driver.

Disable Native Large Object Support

The old driver allowed the user to disable support for Native Large Object datatypes by using the DSN option Use Native Large Object Support or the configuration option `UseNativeLOBSupport`. These options are not available in the new driver.

Return Output Parameters as ResultSet

The old driver allowed the user to return INOUT and OUT parameters as a ResultSet rather than as parameters by using the configuration option `OutputAsResultSet`. The ODBC specification no longer has an option for returning these parameters as a ResultSet, and this option is not available in the new driver.

ClientKanjiFormat (UNIX/Linux only)

On UNIX and Linux systems, the old driver allowed the user to specify a character set format by using the configuration option `ClientKanjiFormat`. This option is not available in the new driver.

Redisplay Reconnect Wait

The old driver allowed the user to specify how long to display the ODBC continue/cancel reconnection prompt during a reconnection by using the `Redisplay Reconnect Wait DSN` option. This prompt is not displayed in the new driver, and this option is not available.

New Teradata ODBC Features

Overview

The following features are available in the new Teradata ODBC Driver.

Differences in Driver Implementation

The new driver defaults to Deferred Mode, as this is faster if you do not want to cache into memory. SLOB is better in certain cases, such as when working with geospatial data.

There are differences between the old and the new driver's SLOB implementation:

- The new driver can cache, or attempt to cache, up to 2GB of all SLOBs in a row.
- The old driver only caches two Response Buffers, up to 32MB.

The new driver has three configuration parameters:

- 1- Enable SLOB Random Access
- 2- Max size of one SLOB
- 3- Max size of all SLOBs in a Row

When #1 is set to True, the new driver will cache up to #3, potentially up to 2GB.

LightWeight Parser

The new driver uses a proprietary parsing mechanism to replace Teradata's LightWeightParser (LWP). This implementation of the LWP streamlines maintenance and overcomes non-trivial issues that exist in the original LWP.

Connection Testing

The new driver provides a mechanism for testing connections. When creating a DSN, you can click the **Test** button in the **Create New Data Source** dialog box to check if the driver can connect to the data store using the DSN.

NOTICE

You must provide a username and password to test the connection. However, make sure you do not save your credentials in the DSN, as this presents a security risk.

Connection String Syntax

If a value in a connection string is enclosed in braces ({}), and the value itself contains a closing brace (}) immediately followed by a semicolon (;), the old driver cannot parse it. The new driver can parse these values successfully, but the closing brace within the value must be escaped with another closing brace.

For example, in a connection string where the `UID` property is set to the value `{};`, you must specify it as follows: `UID= {{}};}`

Return Generated Keys

The Return Generated Keys option, or the `ReturnGeneratedKeys` connection property, enables the driver to return the RowCount and ResultSet from requests that insert data into identity columns. The old driver supports this feature for SQLExecute only, while the new driver supports this feature for both SQLExecute and SQLExecDirect.

ASCII Character Session Set

In the old driver, specifying the Client Character Set as ASCII indicates an extended ASCII character set. In the new driver, specifying the Client Character Set as ASCII indicates a standard 7-bit US ASCII character set.

In addition, the new driver follows the *International Character Set Support* (B035-1125) documentation. If any characters are used that are outside the 7-bit ASCII range, that is, characters with values from 0x80 to 0xff, then a language-specific session character set should be used.

Converting between CHAR and Numeric Values

When converting data from CHAR to numeric values, if the CHAR is not a standard valid representation of a number then the conversion fails. The old driver supported extra leading and trailing spaces as well as spaces between the sign and the number, for example, "+ 789", " - 504.E1", " + .123E1 ". The new driver supports leading and trailing spaces, but does not support spaces between the sign and number.

Row Count Results

When executing operations that return row counts (instead of result sets), the old driver returns a value. The new driver returns the value ROW_COUNT_UNKNOWN (-1) except when executing one of the following operations:

- Insert
- Update
- Delete
- Merge

Creating Stored Procedures

When creating stored procedures, the old driver reports any warnings that occur. The new driver does not report warnings.

Creating User-Defined Functions

When resolving a CREATE request for a UDF (user-defined function), the old driver returns SQL_SUCCESS_WITH_INFO. The new driver returns SQL_NO_DATA_FOUND.

Using Descriptors for SQL_C_NUMERIC

The old driver does not use descriptors to retrieve the precision and scale that SQL_C_NUMERIC data must use, while the new driver uses descriptors to get that information. If the descriptor does not specify these values, then the new driver uses the default values of 39 for precision and 0 for scale.

For more information, see *HOWTO: Retrieving Numeric Data with SQL_NUMERIC_STRUCT* from Microsoft Support.

Catalog Functions

Some catalog functions in the new driver behave differently compared to the old driver. The functions, and the differences between the new driver and the old driver are described in the following sections.

Catalog Function	Description
SQLTables	Returns the value “TYPE” in the TABLE_TYPE column for user-defined types. The type name is returned in the TABLE_NAME column.
SQLColumns	Returns the value SQL_UNKNOWN_TYPE (zero) in the DATA_TYPE for a UDT column. The UDT name is returned in the TYPE_NAME column.
SQLProcedures	Returns names of user-defined methods in addition to names of macros, procedures, and user-defined functions. The value of the PROCEDURE_TYPE column for a user-defined method is SQL_PT_PROCEDURE.
SQLProcedureColumns	Returns parameter information for user-defined methods. Also, parameter types might be UDTs and these are returned as for the SQLColumns catalog function (SQL_UNKNOWN_TYPE in DATA_TYPE column and UDT name in TYPE_NAME column). The output for TD_ANYTYPE parameters results in the value SQL_UNKNOWN_TYPE in the DATA_TYPE column and the

Catalog Function	Description
	string “TD_ANYTYPE” (without quotes) in the TYPE_NAME column.

All Catalog Functions

The old driver returns some column names differently depending on whether the driver is working in an ODBC 2.x or 3.x environment. The new driver always returns ODBC 3.x column names, even when working in an ODBC 2.x environment.

SQLBindParameter

Binding Date, Time, and Timestamp Literals

When binding any of these types of literals as a parameter, the old driver accepts literals that contain extra spaces. The new driver only accepts literals that are specified in the exact format specified in the ODBC specification. If you try to bind a literal that does not use the required format, the new driver returns the following error:

```
[Simba][Support] (40550) Invalid character value for cast specification.
```

For more information about the required format, see *Date, Time, and Timestamp Literals* in the *ODBC Programmers’ Reference* located at:

https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/date-time-and-timestamp-literals.

As an example, both drivers accept the value `{d '1995-01-15'}`, but only the old driver accepts the value `{ d '1995-01-15'}.` Note the missing space in the first value before 'd', but the inserted space in the second value.

Returning Error Information

The new driver responds to certain errors differently than the old driver as described below.

- The new driver does not support SQL_ARRAY_STATUS_PTR and SQL_DIAG_ROW_NUMBER for parameter sets, so the driver does not set these properties when an error occurs in a query that contains a parameter set.
- If no errors occurred, but some parameter sets were ignored, then the old driver sets SQL_ATTR_PARAMS_PROCESS_PTR to SQL_ATTR PARAMSET_SIZE minus the number of ignored sets. The new driver sets SQL_ATTR_PARAMS_PROCESSED_PTR to the exact value of SQL_ATTR_PARAMSET_SIZE.

  This behavior of the new driver is consistent with the ODBC specification.

For more information, see the *Error Information* in *SQLBindParameter Function* in the *ODBC Programmers' Reference:* https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/sqlbindparameter-function.

Using SQL_DEFAULT_PARAM

When SQL_DEFAULT_PARAM is specified as an indicator via the StrLen_or_IndPtr argument, the old driver ignores it and instead uses the value stored in the buffer. Depending upon the environment in which SQL_DEFAULT_PARAM is being used, the new driver returns one of the following:

- If it is specified in a stored procedure call, the new driver uses the value NULL to complete the stored procedure call and returns SQL_SUCCESS_WITH_INFO.
- If it is specified in something other than a stored procedure call, the new driver returns SQL_ERROR and does not execute the statement.

The behavior of the new driver is consistent with the ODBC specification, which states that SQL_DEFAULT_PARAM is valid only when used with a stored procedure call. Teradata Database does not support default parameters for stored procedures, so the new driver uses NULL as the value for SQL_DEFAULT_PARAM.

For more information, see *SQLBindParameter Function* in the *ODBC Programmers' Reference:* https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/sqlbindparameter-function.

Query Parameter Binding

When binding query parameters, the new driver supports different use cases than the old driver.

Old Driver	New Driver
When binding JSON or WJSON data to a parameter, the old driver returns an error.	The new driver supports binding JSON and WJSON data to parameters.
The old driver does not support binding for SQL_DECIMAL and SQL_NUMERIC data that has a negative scale.	The new driver supports binding for these types of values, which is consistent with the ODBC specification: https://docs.microsoft.com/en-us/sql/odbc/reference/appendixes/decimal-digits.
The old driver does not support binding for SQL_DECIMAL and SQL_NUMERIC data that has a precision that is less than 1.	The new driver supports binding for SQL_DECIMAL and SQL_NUMERIC data that has a precision of 0, in order to support binding for NULL values.

Old Driver	New Driver
When calling SQLBindParameter, the old driver verifies the column sizes of the data and then modifies the column sizes of the input if needed.	The new driver does not verify column sizes or modify the input from SQLBindParameter. When binding LOB data types, the new driver uses the values returned by `GetMaxLobBytes()` or `MaxJSONBytes()` as the maximum column size of the LOB data.

Output Parameter Binding

When binding output parameters, a data type conversion is sometimes required. In this case, the old driver converts the output data to its corresponding SQL type regardless of the data types specified in SQLBindParameter. In contrast, the new driver converts the output data to the specified SQL type, or returns a conversion error if the types are not compatible. The new driver's behavior is consistent with the ODBC specification.

For example, given the following procedure:

```
create procedure CharOutputStoredprocedure(OUT param1 CHAR)
cs1: BEGIN
SET param1 = 'A';
END cs1;
```

Assume that you bind the output parameters as follows:

```
SQLBindParameter(
        stmt,
        1,
        SQL_PARAM_OUTPUT,
        SQL_C_CHAR,
        SQL_CHAR,
        1024,
        0,
        &out,
        1024,
        &cbRetParam)
```

The given SQL type is the same as the SQL type of the original parameter, so the data does not need to be converted. The old driver and the new driver both successfully bind the data and return the value A.

However, if you bind the output parameter as follows, then the data must be converted from CHAR to SQL_INTEGER:

```
SQLBindParameter(
        stmt,
        1,
        SQL_PARAM_OUTPUT,
        SQL_C_CHAR,
        SQL_INTEGER,
        1024,
        0,
        &out,
        1024,
        &cbRetParam)
```

The conversion fails because A is not a valid SQL_INTEGER value. The old driver handles this situation by converting and binding the output data to SQL_CHAR instead. The new driver tries to convert the data to SQL_INTEGER and then returns a conversion error with SQL state 22018.

SQLBindParameter and Data Types with Fractional Seconds

When calling SQLBindParameter with a data type that contains fractional seconds, you must set the DecimalDigit to a value up to 6, the maximum the database supports. Previous to 16.20, this could be any number between 0-6 with the same result as if you were to set it to 6, but this is incorrect and no longer supported.

As an example, TIMESTAMP(0) is no longer valid if you are passing in a fractional second because according to the ODBC specification, you have specified zero decimal digits and will receive an error if you try to pass in a fractional second.

You can specify a maximum of up to 6 decimal digits, as this is the limit of the Teradata Database.

It is acceptable to send less than your specified number of decimal digits.

You can optionally pad out to 9 decimal digits with zeros without issue.

SQLCancel

The new driver responds to timing or processing state of the statement differently than the old driver.

The ODBC specification defines the following behavior for SQLCancel in situations where no processing has been done for the statement:

- In ODBC 3.5, SQLCancel has no effect on the statement. To close a cursor, applications need to call SQLCloseCursor instead of SQLCancel.

- In ODBC 2.x, SQLCancel has the same effect as SQLFreeStmt with the SQL_CLOSE option. This behavior is defined only for the sake of completeness; applications should call SQLFreeStmt or SQLCloseCursor instead to close cursors.

For more information, see *SQLCancel Function* in the *ODBC API Reference*: https://msdn.microsoft.com/en-us/library/ms714112%28v=vs.85%29.aspx.

Old Driver	New Driver
When executing statements asynchronously, if the execution is completed before SQLCancel is called, the old driver returns HY008.	The new driver returns the result of the statement execution, (SQL_SUCCESS or SQL_ERROR).
If SQLCancel is called before any processing has been done for the statement, the old driver closes the statement regardless of whether the driver is working in ODBC 2.x mode or ODBC 3.x mode. Closing the statement reflects behavior that is consistent with the ODBC 2.x specification, but not the ODBC 3.x specification.	The new driver does not close the statement when it is working in ODBC 3.x mode, and this behavior is consistent with the ODBC 3.x specification.

SQLForeignKeys

In the old driver, the columns UPDATE_RULE and DELETE_RULE are returned as empty strings. In the new driver, these columns are instead returned as NULL.

SQLGetConnectAttr

The following table lists the results of the new and old drivers.

Function	Old Driver Returns	New Driver Returns
SQL_ATTR_ASYNC_ENABLE	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 0, *pcbErrorMsg = 50,`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = SQL_ASYNC_ENABLE_OFF = 0, *StringLengthPtr = 4`

Function	Old Driver Returns	New Driver Returns
	`*ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Invalid Attribute"`	
SQL_ATTR_DISCONNECT_BEHAVIOR	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 0, *pcbErrorMsg = 50, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Invalid Attribute"`	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 10210, *pcbErrorMsg = 75, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute identifier invalid or not supported: 114"`
SQL_ATTR_ENLIST_IN_DTC	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 0, *pcbErrorMsg = 50, *ColumnNumber = -1, *RowNumber = -1 MessageText =`	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HY092", *pfNativeError = 10210, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute`

Function	Old Driver Returns	New Driver Returns
	`"[Teradata][ODBC Teradata Driver] Invalid Attribute"`	`identifier invalid or not supported: 1207"`
SQL_ATTR_PACKET_SIZE	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 4096, *StringLengthPtr = 4`
SQL_ATTR_TRANSLATE_LIB	`Return: SQL_SUCCESS=0 Out: *ValuePtr = "쨀쫊쫊쫊쫊쫊쫊쫊쫊쫊쫊쫊쫊쫊쫊쫊쫊...", *StringLengthPtr = 0`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = "", *StringLengthPtr = 0`

SQLGetDiagField

At this time, the new driver does not support setting the following:

- SQL_DIAG_CURSOR_ROW_COUNT
- SQL_DIAG_ROW_COUNT

SQLGetInfo

The following table lists the results of the new and old drivers.

Function	Old Driver Returns	New Driver Returns

SQL_SQL_CONFORMANCE	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = SQL_SC_SQL92_ENTRY = 1, *StringLengthPtr = 4`

SQL_CATALOG_NAME_SEPARATOR	`Return: SQL_SUCCESS=0`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = 0`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = ".", *StringLengthPtr = 2`

SQL_CREATE_TABLE	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver]`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`

Function	Old Driver Returns	New Driver Returns
	`Driver does not support specified fInfoType"`	

SQL_DROP_TABLE	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`

SQL_DROP_VIEW	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`

SQL_GETDATA_EXTENSIONS	`Return: SQL_SUCCESS=0`
	`Out: *InfoValuePtr`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x0000000F =`

Function	Old Driver Returns	New Driver Returns
	= 0x0000000B = SQL_GD_ANY_COLUMN | SQL_GD_ANY_ORDER | SQL_GD_BOUND, *StringLengthPtr = 4	SQL_GD_ANY_COLUMN | SQL_GD_ANY_ORDER | SQL_GD_BLOCK | SQL_GD_BOUND, *StringLengthPtr = 4
SQL_INDEX_KEYWORDS	Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4
SQL_LOCK_TYPES	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000001 = SQL_LCK_NO_CHANGE, *StringLengthPtr = 4	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000002 = SQL_LCK_EXCLUSIVE, *StringLengthPtr = 4
SQL_MAX_ASYNC_CONCURRENT_STATEMENTS	Return: SQL_ERROR=-1 Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified> dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1 MessageText =	Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0, *StringLengthPtr = 4

<table>
<tr><th>Function</th><th>Old Driver Returns</th><th>New Driver Returns</th></tr>
<tr><td></td><td>"[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"</td><td></td></tr>
<tr><td>SQL_NUMERIC_FUNCTIONS</td><td>Return: SQL_SUCCESS=0 Out:*InfoValuePtr = 0x00014D01 = SQL_FN_NUM_ABS | SQL_FN_NUM_EXP | SQL_FN_NUM_LOG | SQL_FN_NUM_MOD | SQL_FN_NUM_SQRT | SQL_FN_NUM_PI, StringLengthPtr = 4</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00194D01 = SQL_FN_NUM_ABS | SQL_FN_NUM_EXP | SQL_FN_NUM_LOG | SQL_FN_NUM_MOD | SQL_FN_NUM_SQRT | SQL_FN_NUM_PI | SQL_FN_NUM_LOG10 | SQL_FN_NUM_POWER, *StringLengthPtr = 4</td></tr>
<tr><td>SQL_ODBC_SAG_CLI_CONFORMANCE</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = SQL_OSCC_COMPLIANT = 1, *StringLengthPtr = 2</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = SQL_OSCC_NOT_COMPLIANT = 0, *StringLengthPtr = 2</td></tr>
<tr><td>SQL_POS_OPERATIONS</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000001 = SQL_POS_POSITION, *StringLengthPtr = 4</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4</td></tr>
<tr><td>SQL_QUALIFIER_NAME_SEPARATOR</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = 0</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = ".", *StringLengthPtr = 2</td></tr>
<tr><td>SQL_SCROLL_CONCURRENCY</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000001 =</td><td>Return: SQL_SUCCESS=0 Out: *InfoValuePtr = 0x00000003 = SQL_SCCO_READ_ONLY |</td></tr>
</table>

Function	Old Driver Returns	New Driver Returns
	`SQL_SCCO_READ_ONLY, *StringLengthPtr = 4`	`SQL_SCCO_LOCK, *StringLengthPtr = 4`

SQL_SQL92_GRANT	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000010 = SQL_SG_WITH_GRANT_OPTION, *StringLengthPtr = 4`

SQL_SQL92_PREDICATES	`Return: SQL_ERROR=-1Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00003F07 = SQL_SP_EXISTS | SQL_SP_ISNOTNULL | SQL_SP_ISNULL | SQL_SP_UNIQUE | SQL_SP_LIKE | SQL_SP_IN | SQL_SP_BETWEEN | SQL_SP_COMPARISON | SQL_SP_QUANTIFIED_COMPARISON, *StringLengthPtr = 4`

Function	Old Driver Returns	New Driver Returns

SQL_SQL92_RELATIONAL_JOIN_OPERATORS	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x0000015A = SQL_SRJO_CROSS_JOIN | SQL_SRJO_FULL_OUTER_JOIN | SQL_SRJO_INNER_JOIN | SQL_SRJO_LEFT_OUTER_JOIN | SQL_SRJO_RIGHT_OUTER_JOIN, *StringLengthPtr = 4`

SQL_SQL92_STRING_FUNCTIONS	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000006 = SQL_SSF_LOWER | SQL_SSF_UPPER, *StringLengthPtr = 4`

SQL_STATIC_SENSITIVITY	`Return: SQL_SUCCESS=0`
	`Out: *InfoValuePtr = 0x00000000, *StringLengthPtr = 4`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = 0x00000003 = SQL_SS_ADDITIONS | SQL_SS_DELETIONS, *StringLengthPtr = 4`

Function	Old Driver Returns	New Driver Returns

SQL_XOPEN_CLI_YEAR	`Return: SQL_ERROR=-1`
	`Out: *InfoValuePtr = <unmodified>, *StringLengthPtr = <unmodified>`
	`dbc: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 76, *ColumnNumber = -1, *RowNumber = -1`
	`MessageText = "[Teradata][ODBC Teradata Driver] Driver does not support specified fInfoType"`
		`Return: SQL_SUCCESS=0`
		`Out: *InfoValuePtr = "1995", *StringLengthPtr = 8`

SQLGetStmtAttr

The following table lists the results of the new and old drivers.

Function	Old Driver Returns	New Driver Returns
SQL_ATTR_CURSOR_SCROLLABLE	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = 4`

Function	Old Driver Returns	New Driver Returns
SQL_ATTR_CURSOR_SENSITIVITY	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = 4`
SQL_ATTR_KEYSET_SIZE	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = <unmodified>`	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HY092", *pfNativeError = 10210, *pcbErrorMsg = 73, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute identifier invalid or not supported: 8"`
SQL_ATTR_RETRIEVE_DATA	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr =`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = SQL_RD_ON = 1,`

Function	Old Driver Returns	New Driver Returns
	`<unmodified> stmt: szSqlState = "HYC00", *pfNativeError = 0, *pcbErrorMsg = 44, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC Teradata Driver] Unsupported"`	`*StringLengthPtr = 4`
SQL_ATTR_ROW_NUMBER ***NOTICE*** The old driver always returns SQL_ROW_NUMBER_UNKNOWN; the new driver returns the actual number of the current row in the entire result set.	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 0, *StringLengthPtr = <unmodified>`	`Return: SQL_SUCCESS=0 Out: *ValuePtr = 1, *StringLengthPtr = 4`
SQL_ATTR_SIMULATE_CURSOR	`Return: SQL_SUCCESS=0 Out: *ValuePtr = SQL_SC_NON_UNIQUE = 0, *StringLengthPtr = <unmodified>`	`Return: SQL_ERROR=-1 Out: *ValuePtr = <unmodified>, *StringLengthPtr = <unmodified> stmt: szSqlState = "HY092", *pfNativeError = 10210, *pcbErrorMsg = 74, *ColumnNumber = -1, *RowNumber = -1 MessageText = "[Teradata][ODBC] (10210) Attribute identifier invalid or not supported: 10"`

SQLGetTypeInfo

The new driver returns an additional custom column "USER_DATA_TYPE" at index 20.

New driver columns:

```
1, TYPE_NAME, 9, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
2, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
3, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
4, LITERAL_PREFIX, 14, SQL_VARCHAR=12, 32, 0, SQL_NULLABLE=1
5, LITERAL_SUFFIX, 14, SQL_VARCHAR=12, 32, 0, SQL_NULLABLE=1
6, CREATE_PARAMS, 13, SQL_VARCHAR=12, 32, 0, SQL_NULLABLE=1
7, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
8, CASE_SENSITIVE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
9, SEARCHABLE, 10, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
10, UNSIGNED_ATTRIBUTE, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
11, FIXED_PREC_SCALE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
12, AUTO_UNIQUE_VALUE, 17, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
13, LOCAL_TYPE_NAME, 15, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
14, MINIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
15, MAXIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
16, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
17, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
18, NUM_PREC_RADIX, 14, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
19, INTERVAL_PRECISION, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
20, USER_DATA_TYPE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
21, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

Old driver columns:

```
1, TYPE_NAME, 9, SQL_VARCHAR=12, 39, 0, SQL_NO_NULLS=0
2, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
3, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
4, LITERAL_PREFIX, 14, SQL_VARCHAR=12, 11, 0, SQL_NULLABLE=1
5, LITERAL_SUFFIX, 14, SQL_VARCHAR=12, 18, 0, SQL_NULLABLE=1
6, CREATE_PARAMS, 13, SQL_VARCHAR=12, 18, 0, SQL_NULLABLE=1
7, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
8, CASE_SENSITIVE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
9, SEARCHABLE, 10, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
10, UNSIGNED_ATTRIBUTE, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
11, FIXED_PREC_SCALE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
12, AUTO_UNIQUE_VALUE, 17, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
13, LOCAL_TYPE_NAME, 15, SQL_VARCHAR=12, 39, 0, SQL_NULLABLE=1
14, MINIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
15, MAXIMUM_SCALE, 13, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
```

```
16, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
17, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
18, NUM_PREC_RADIX, 14, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
19, INTERVAL_PRECISION, 18, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
20, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

SQLProcedureColumns

The old driver uses SQL_VARCHAR for string columns. The new driver uses SQL_WVARCHAR for string columns and returns two additional custom columns, at index 20 and 21.

New driver columns:

```
1, PROCEDURE_CAT, 13, SQL_VARCHAR=12, 1024, 0, SQL_NULLABLE=1
2, PROCEDURE_SCHEM, 15, SQL_VARCHAR=12, 30, 0, SQL_NULLABLE=1
3, PROCEDURE_NAME, 14, SQL_VARCHAR=12, 30, 0, SQL_NO_NULLS=0
4, COLUMN_NAME, 11, SQL_VARCHAR=12, 30, 0, SQL_NO_NULLS=0
5, COLUMN_TYPE, 11, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
6, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
7, TYPE_NAME, 9, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
8, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
9, BUFFER_LENGTH, 13, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
10, DECIMAL_DIGITS, 14, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
11, NUM_PREC_RADIX, 14, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
12, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
13, REMARKS, 7, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
14, COLUMN_DEF, 10, SQL_VARCHAR=12, 4000, 0, SQL_NULLABLE=1
15, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
16, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
17, CHAR_OCTET_LENGTH, 17, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
18, ORDINAL_POSITION, 16, SQL_INTEGER=4, 10, 0, SQL_NO_NULLS=0
19, IS_NULLABLE, 11, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
20, IS RESULT SET COLUMN, 20, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
21, USER_DATA_TYPE, 14, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
22, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

Old driver columns:

```
1, PROCEDURE_CAT, 13, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
2, PROCEDURE_SCHEM, 15, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
3, PROCEDURE_NAME, 14, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
4, COLUMN_NAME, 11, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
5, COLUMN_TYPE, 11, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
6, DATA_TYPE, 9, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
7, TYPE_NAME, 9, SQL_VARCHAR=12, 128, 0, SQL_NO_NULLS=0
```

```
8, COLUMN_SIZE, 11, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
9, BUFFER_LENGTH, 13, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
10, DECIMAL_DIGITS, 14, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
11, NUM_PREC_RADIX, 14, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
12, NULLABLE, 8, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
13, REMARKS, 7, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
14, COLUMN_DEF, 10, SQL_VARCHAR=12, 60, 0, SQL_NULLABLE=1
15, SQL_DATA_TYPE, 13, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
16, SQL_DATETIME_SUB, 16, SQL_SMALLINT=5, 5, 0, SQL_NULLABLE=1
17, CHAR_OCTET_LENGTH, 17, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
18, ORDINAL_POSITION, 16, SQL_INTEGER=4, 10, 0, SQL_NO_NULLS=0
19, IS_NULLABLE, 11, SQL_VARCHAR=12, 3, 0, SQL_NULLABLE=1
20, TDODBC_DATA_TYPE, 16, SQL_SMALLINT=5, 5, 0, SQL_NO_NULLS=0
```

SQLTables

When using pattern matching with a wildcard character (%), the default catalog metadata is null, so the new driver returns SQL_INTEGER as the SQL Type for some columns.

For example, for the following call:

```
SQLTables(<empty string>, %, <empty string>, <null pointer>)
```

New driver returns:

```
icol, szColName, *pcbColName, *pfSqlType, *pcbColDef, *pibScale, *pfNullable
1, TABLE_CAT, 9, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
2, TABLE_SCHEM, 11, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
3, TABLE_NAME, 10, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
4, TABLE_TYPE, 10, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
5, REMARKS, 7, SQL_INTEGER=4, 10, 0, SQL_NULLABLE=1
```

Old driver returns:

```
icol, szColName, *pcbColName, *pfSqlType, *pcbColDef, *pibScale, *pfNullable
1, TABLE_CAT, 9, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
2, TABLE_SCHEM, 11, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
3, TABLE_NAME, 10, SQL_VARCHAR=12, 128, 0, SQL_NULLABLE=1
4, TABLE_TYPE, 10, SQL_VARCHAR=12, 17, 0, SQL_NULLABLE=1
5, REMARKS, 7, SQL_VARCHAR=12, 254, 0, SQL_NULLABLE=1
```

Column Attributes

For all column attribute information retrieved using SQLColAttribute, the old driver returns values that vary based on the data. In the new driver, the following column attributes are based on another attribute or hard-coded to a specific value:

- SQL_DESC_DATETIME_INTERVAL_PRECISION

  For all column types, in the new driver, SQL_DESC_DATETIME_INTERVAL_PRECISION = SQL_DESC_PRECISION.
- SQL_DESC_PRECISION

  For all column types except for NUMERIC and DECIMAL, in the new driver, SQL_DESC_PRECISION = SQL_DESC_LENGTH.

Additionally, the following table lists by column type the column attributes the new Teradata ODBC Driver returns.

Column Type	Column Attributes Returned by New Driver
BIGINT	Octet length is 20.
DATE	Display size is returned as a number of characters.
DOUBLE, FLOAT, or REAL	Display size is 24.
Interval types	• Octet length is 34 • Display size is SQL_DESC_LENGTH + 1 • SQL_DESC_SCALE = SQL_DESC_PRECISION
JSON	Octet length is SQL_DESC_LENGTH * character size.
TIME	Display size is returned as a number of characters.
TIMESTAMP	Display size is returned as a number of characters.

ODBC API Conformance

For the following functions, the new driver adheres to a different level of ODBC API conformance than the old driver:

- SQLExecute and SQLExecDirect, when no rows are affected by the statement.

  When a statement affects 0 rows, the old driver returns `SQL_SUCCESS`. The new driver returns `SQL_SUCCESS` when using ODBC 2.x, and returns `SQL_NO_DATA` when using ODBC 3.x.

  For more information, see "SQL_NO_DATA" in the *ODBC Programmers' Reference*: .
- SQLPrepare, SQLExecute, and SQLExecDirect, when an empty statement is passed in.

  When an empty SQL statement is passed into SQLPrepare, SQLExecute, or SQLExecDirect, the old driver returns `SQL_SUCCESS`. The new driver returns `SQL_ERROR` with `SQLState 42000`.

Converting Data between SQL Types and C Types

When converting data between a SQL type and a C Interval type, the new driver returns some results differently than the old driver because the new driver supports more conversions. The old driver does not support the following conversions and returns `SQLState 07006`; the new driver converts the data successfully as shown in the example below:

- Numeric types (BIGINT, BIT, DOUBLE, FLOAT, INTEGER, REAL, SMALLINT, TINYINT) to:
    - INTERVAL DAY TO HOUR
    - INTERVAL DAY TO MINUTE
    - INTERVAL DAY TO SECOND
    - INTERVAL HOUR TO MINUTE
    - INTERVAL HOUR TO SECOND
    - INTERVAL MINUTE TO SECOND
- INTERVAL DAY to INTERVAL SECOND
- INTERVAL HOUR to INTERVAL SECOND
- INTERVAL DAY TO HOUR to INTERVAL SECOND
- INTERVAL DAY TO SECOND to INTERVAL SECOND
- INTERVAL HOUR TO MINUTE to INTERVAL SECOND
- INTERVAL MONTH to INTERVAL YEAR
- INTERVAL HOUR to INTERVAL DAY
- INTERVAL MINUTE to INTERVAL DAY
- INTERVAL MINUTE to INTERVAL HOUR
- INTERVAL MINUTE to INTERVAL DAY TO HOUR
- INTERVAL SECOND to INTERVAL DAY
- INTERVAL SECOND to INTERVAL HOUR
- INTERVAL SECOND to INTERVAL MINUTE
- INTERVAL SECOND to INTERVAL DAY TO HOUR
- INTERVAL SECOND to INTERVAL DAY TO MINUTE
- INTERVAL SECOND to INTERVAL HOUR TO MINUTE
- INTERVAL YEAR TO MONTH to INTERVAL YEAR
- INTERVAL DAY TO HOUR to INTERVAL DAY
- INTERVAL DAY TO MONTH to INTERVAL DAY
- INTERVAL DAY TO MONTH to INTERVAL HOUR
- INTERVAL DAY TO MONTH to INTERVAL DAY TO HOUR
- INTERVAL DAY TO SECOND to INTERVAL DAY
- INTERVAL DAY TO SECOND to INTERVAL HOUR
- INTERVAL DAY TO SECOND to INTERVAL MINUTE
- INTERVAL DAY TO SECOND to INTERVAL DAY TO MINUTE

- INTERVAL DAY TO SECOND to INTERVAL HOUR TO MINUTE
- INTERVAL HOUR TO MINUTE to INTERVAL DAY
- INTERVAL HOUR TO MINUTE to INTERVAL HOUR
- INTERVAL HOUR TO MINUTE to INTERVAL DAY TO HOUR
- INTERVAL HOUR TO SECOND to INTERVAL DAY
- INTERVAL HOUR TO SECOND to INTERVAL HOUR
- INTERVAL HOUR TO SECOND to INTERVAL DAY TO HOUR
- INTERVAL MINUTE TO SECOND to INTERVAL DAY
- INTERVAL MINUTE TO SECOND to INTERVAL HOUR
- INTERVAL MINUTE TO SECOND to INTERVAL MINUTE
- INTERVAL MINUTE TO SECOND to INTERVAL DAY TO HOUR
- INTERVAL MINUTE TO SECOND to INTERVAL DAY TO MINUTE
- INTERVAL MINUTE TO SECOND to INTERVAL HOUR TO MINUTE
- SQL_C_DOUBLE to SQL_INTERVAL_MONTH
- SQL_C_DOUBLE to SQL_INTERVAL_YEAR
- SQL_C_FLOAT to SQL_INTERVAL_MONTH
- SQL_C_FLOAT to SQL_INTERVAL_YEAR

H

Security Considerations

Disabling Password Saving in a DSN

A new registry key `Security` is introduced (for both 32- and 64-bit) in this release which would enable a user with Admin privilege to set a value named `DisableSavePassword` to disable the password saving feature in the DSN configuration.

Note:

> It is important to mention that this is a Windows-only feature. Other platforms are unaffected.

OS type	Registry Key
64-bit	`HKEY_LOCAL_MACHINE\SOFTWARE\Teradata\Client\Security`
32-bit	`HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Teradata\Client\Security`

This key is a DWORD value with the name `DisableSavePassword`. A value of `1` prevents the password from being saved. Any other value allows passwords to be saved. If the Key or Value is missing, then the password is saved.

Changes in ODBC Data Source Administrator

If `DisableSavePassword` is enabled (value is set to `1`), then no password is saved when a user creates a new DSN in the **ODBC Data Source Administrator**. The user can still test the connection by clicking **Test** after providing a **Name or IP Address**, **Username**, and **Password**. If the user clicks **OK**, then the warning dialog box that prompts the user whether or not to save the password no longer appears, and all information entered except the password is saved in the registry.

For an existing DSN that already had the password saved in the registry, if `DisableSavePassword` is enabled, then the user can still use the existing password to try to connect to the database (the **DSN config** window will show the previously saved password as masked) but they cannot save a new password on that DSN. To delete the saved password for an existing DSN, a user must change the existing DSN by modifying the Data Source **Name** and save it (thus creating a new DSN based on the old one).

Additional Information

Audience

This book is intended for use by:

- System and application programmers
- System administrators

Changes and Additions

The following changes were made to this book in support of the current release.

Date and Release	Description
October 2018 16.20.11	• Chapter 3: ◦ *Teradata ODBC Driver Options*: Updated the dialog box and added the **Enable Client Side UDF Upload** and **UDF Upload Path** fields to the table's new **UDF Upload** section. ◦ *Teradata ODBC Driver Advanced Options*: ▪ Updated `AccountString` to specify `AccountStr`. ▪ Updated the **Max Single LOB Bytes** default value to 4000. ▪ Updated the **Max Total LOB Bytes Per Row** value to 65336. • Chapter 4: ◦ *Teradata ODBC Driver Options*: Updated the dialog box and added the **Enable Client Side UDF Upload** and **UDF Upload Path** fields. ◦ *Teradata ODBC Driver Advanced Options*: ▪ Updated `AccountString` to specify `AccountStr`. ▪ Updated the **Max Single LOB Bytes** default value to 4000. ▪ Updated the **Max Total LOB Bytes Per Row** value to 65336. • Chapter 5: ◦ *Teradata DSN Options*: Updated `AccountString` information. ◦ Added `EnableUDFUpload` for the new **Enable UDF Source Upload from Client** field. ◦ Added `UDFUploadPath` for the new **Source Root Directory** field. • Chapter 6, *LOB Retrieval Modes*: ◦ Updated the **Max Single LOB Bytes** default value to 4000. ◦ Updated the **Max Total LOB Bytes Per Row** value to 65336.
April 2018	• Chapter 1:

Date and Release	Description
16.20.06	◦ *Key Changes in Product Behavior*: Removed the HPUX reference. ◦ *ODBC Integrated Directories*: Updated the paragraph describing installation options. ◦ *Linux/UNIX and Apple OS X Systems*: Updated `tdodbc` instances to `tdbc1620`. ◦ *ODBC Integrated Directories*: Updated paragraph describing installation options and capitalize the ODBC instances in the file paths in the table (for example, `ODBC_32/`). • Chapter 2: ◦ *ODBC Directories*: Updated the `ErrorMessage` directory reference to specify `ErrorMessages`. ◦ *Coexistence of Different Version Drivers on the Same Machine*: Updated the .env file location path to specify `<prefix>/teradata/client/16.20/etc` and updated the version references in the `bash.env` and `csh.env` files to 1620. • Chapter 3: ◦ *ODBC Driver Setup Parameters*: Updated the table's Mechanism row to remove SPNEGO as it is no longer supported; and added JSON Web Token (JWT) as a new mechanism. ◦ *Teradata ODBC Driver Advanced Options*: Updated State Check Level default value and removed the Note indicating feature deprecation. ◦ *Teradata ODBC Driver Options*: Updated the *Driver Options* figure. ◦ *Windows ODBC Driver Directories*: Updated the following directory and directory options and added a directory: ▪ `tdodbcdsn` is now `tdodbcdsn.vbs` ▪ `libcrypto-1_1-x64.dll` is now `libcrypto-1_1-x64.dll` or `libcrypto-1_1.dll` ▪ `sbicudt53_64.dll` is now `sbicudt53_64.dll` or `sbicudt53_32.dll` ▪ `sbicuin53_64.dll` is now `sbicuin53_64.dll` or `sbicuin53_32.dll` ▪ `sbicuuc53_64.dll` is now `sbicuuc53_64.dll` or `sbicuuc53_32.dll` ▪ `tdataodbc_sb64.dll` is now `tdataodbc_sb64.dll` or `tdataodbc_sb32.dll` ▪ `TeradataODBC64.man` or `TeradataODBC32.man` was added • Chapter 4: ◦ *Configuring a DSN Using ODBC Administrator Tool*: Updated the table's Mechanism row to add JSON Web Token (JWT) as a new mechanism. ◦ *Teradata ODBC Driver Options*: Updated the *Teradata ODBC Driver Options* figure. • Chapter 6:

Date and Release	Description
	◦ *UNIX OS Application Development*: Removed SUSE Linux 390 as a supported operating system. ◦ *Scalar Functions*: Updated the section with new content. ◦ *Teradata Database Connect*: Updated Mechanism row to include JWT as a supported mechanism • Chapter 7: ◦ *Authentication Mechanisms*: Added JSON Web Token. ◦ *Username and Password Requirements*: Deleted the TDO row as TD2 supersedes it, and added a row for JWT. • Chapter 8: ◦ Added a new section titled *SET TRANSFORM GROUP FOR TYPE Statement*. • Appendix A: ◦ *Release-Independent 64-bit odbc.ini File Example*: Updated `tdata.so` to `tdataodbc_sb64.so`. ◦ *Release-Independent odbc.ini File for Apple OS X Example*: Updated `tdata.dylib` to `tdataodbc_sbu.dylib` • Appendix B: ◦ *Release-Independent 64-bit odbcinst.ini File Example*: Updated `tdata.so` to `tdataodbc_sb64.so`. ◦ *Release-Dependent 32-bit odbcinst.ini File Example*: Updated `tdata.so` to `tdataodbc_sb32.so`. ◦ *Release-Dependent 64-bit odbcinst.ini File Example*: Updated `tdata.so` to `tdataodbc_sb64.so`. • Appendix D: ◦ *Data Source Specification Options Example*: Updated `tdata.so` to `tdataodbc_sb64.so`. ◦ *DSN Tracing Options Example*: Updated the file example to reflect the SEN driver. • Appendix H: ◦ *Catalog Functions*: Added section titled *SQLBindParameter and Data Types with Fractional Seconds*.
November 2017 16.20.00	Initial release. This book leverages content from the *ODBC Driver for Teradata User Guide* (B035-2509) to match the functionality of the new Teradata ODBC Driver 16.20.

Teradata Links

Link	Description
https://docs.teradata.com/	Teradata documentation (HTML)
http://www.info.teradata.com	Teradata documentation (PDF)
https://access.teradata.com	Customer portal (one stop source for Teradata services and products)
http://www.teradata.com/products-and-services/TEN	Teradata education network
https://community.teradata.com	Link to Teradata community (also available from the customer portal)

Related Documentation

Documents are located at https://docs.teradata.com/.

Specific books related to Teradata ODBC Driver are as follows:

Title	Publication ID
Security Administration	B035-1100
SQL Data Types and Literals	B035-1143
Teradata Tools and Utilities for Microsoft Windows Installation Guide	B035-2407
Teradata Tools and Utilities for IBM AIX Installation Guide	B035-3125
Teradata Tools and Utilities for Oracle Solaris on AMD Opteron Systems Installation Guide	B035-3126
Teradata Tools and Utilities for Oracle Solaris on SPARC Systems Installation Guide	B035-3127
Teradata Tools and Utilities for IBM z/OS Installation Guide	B035-3128
Teradata Tools and Utilities for Apple OS X Installation Guide	B035-3129
Teradata Tools and Utilities for Linux Installation Guide (CentOS, OEL, RedHat, SLES, Ubuntu)	B035-3160