Oracle® Database

ODBC Developer's Guide

23ai
F92980-03
November 2024

Oracle Database ODBC Developer's Guide, 23ai

F92980-03

Primary Author: Jiji Thomas

Contributing Authors: Rhonda Day

Contributors: Veronica Dumitriu, Christopher Jones, Alan Williams

Contents

List of Figures

List of Tables

Preface

This guide provides comprehensive information about the Oracle Database Open Database Connectivity (ODBC) driver, including instructions on how to install and configure the ODBC driver, and how to use the ODBC driver to connect ODBC-compliant applications to an Oracle data source.

Audience

This guide is intended for use by administrators and software programmers who want to use the Oracle Database ODBC driver in their database applications to connect to an Oracle data source.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

These documents in the Oracle Database documentation set provide more information that can assist you when using this document:

- *Oracle Call Interface Programmer's Guide*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Net Services Reference*
- *Oracle Database Client Installation Guide for Linux*
- *Oracle Database Security Guide*
- *Oracle Database JDBC Developer's Guide*
- *Oracle Database Globalization Support Guide*

Conventions

This guide uses these text conventions:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in ODBC Developer's Guide

The following are the changes in ODBC Developer's Guide for Oracle Database 23ai, and earlier releases:

New Features in 23ai

The following are the new features in ODBC Developer's Guide for Oracle Database 23ai.

ODBC Release 23ai

The following new features are added to the Oracle Database ODBC driver, release 23ai.

ODBC Support for SQL BOOLEAN Data Type

The Oracle Database ODBC driver supports a native SQL `BOOLEAN` data type, using `SQL_BIT` to map to the SQL `BOOLEAN` data type, and returning "1" or "0."

ODBC Support for VECTOR Data Type

The Oracle Database ODBC driver supports `VECTOR` data type, using `SQL_CHAR` to map to the `VECTOR` data type.

See Implementation of Data Types (Advanced)

New Features in Earlier Releases

The following are the new features in the earlier releases of the Oracle Database ODBC driver.

ODBC Release 21c, Version 21.1

There are no new features for Oracle Database ODBC Driver Release 21c, Version 21.1.

ODBC Release 19c, Version 19.1.0.0.0

There are no new features of Oracle Database ODBC Driver, Release 19c, Version 19.1.0.0.0 software for the Microsoft Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 7, Windows 8, Windows 8.1, Windows 10, Linux X86-64 (32-bit, 64-bit), Sun Solaris SPARC64 (32-bit, 64-bit), IBM AIX 5L (32-bit, 64-bit), Sun Solaris X64 (32-bit, 64-bit), HPUX IA64 (32-bit, 64-bit), ZLinux (32-bit, 64-bit) operating systems.

ODBC Release 18c, Version 18.1.0.0.0

Features of Oracle Database ODBC Driver Release 18c, Version 18.1.0.0.0 software for the Microsoft Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 7, Windows 8, Windows 8.1, Windows 10, Linux X86-64 (32-bit, 64-bit), Sun Solaris SPARC64 (32-bit, 64-bit), IBM AIX 5L (32-bit, 64-bit), Sun Solaris X64 (32-bit, 64-bit), HPUX IA64 (32-bit, 64-bit), ZLinux (32-bit, 64-bit) operating systems are described as follows:

- unixODBC ODBC Driver Manager is upgraded from unixODBC–2.3.2 to unixODBC–2.3.4.

ODBC 12.2.0.1.0

Features of Oracle Database ODBC Driver Release 12.2.0.1.0 software for the Microsoft Windows Server 2008, Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows 7, Windows 8, Windows 8.1, Windows 10, Linux X86-64 (32-bit, 64-bit), Sun Solaris SPARC64 (32-bit, 64-bit), IBM AIX 5L (32-bit, 64-bit), Sun Solaris X64 (32-bit, 64-bit), HPUX IA64 (32-bit, 64-bit), ZLinux (32-bit, 64-bit) operating systems are described as follows:

- Support is added for long identifiers up to 128 bytes.
- Support is added for time stamp with time zone and time stamp with local time zone.

  This feature does not require changes to the existing ODBC application where ODBC `TIMESTAMP` data type is used. If an existing application uses ODBC `TIMESTAMP` data type and the database column is `TIMESTAMP`, the current behavior is preserved.

  For database column `TIMESTAMP WITH TIMEZONE` or `TIMESTAMP WITH LOCAL TIMEZONE`, the time component in the ODBC `TIMESTAMP_STRUCT` is in the user's session time zone. This behavior is transparent to the user's application, requiring no change to the ODBC application.

Deprecated Features

The following is the deprecated feature in *ODBC Developer's Guide* for Oracle Database Release 23ai.

Oracle OLAP

Analytic workspaces, the OLAP DML programming language, financial reporting, and the OLAP Java API continue to be deprecated in Oracle Database 23ai.

Be aware that OLAP will not be supported beyond the term of the current release (Oracle Database 23ai) premier support. Oracle strongly recommends that you do not start new projects using OLAP and begin migrating applications using OLAP to alternatives now. If your application requires an in-database dimensional model, then consider using Oracle Analytic

Views. Analytic views provide a dimensional semantic model, calculations, and query semantics using data in Oracle Database. When used with columnar tables, analytic views provide query performance similar to the OLAP Option. If your application requires support for advanced dimensional analytics, what-if analysis, or forecasting, then consider Oracle Essbase. Oracle Essbase is a multidimensional database management system with support for complex dimensional business analytics.

1

Introduction to the Oracle Database ODBC Driver

This chapter introduces you to the Oracle Database ODBC driver.

Topics:

1.1 About the ODBC Driver

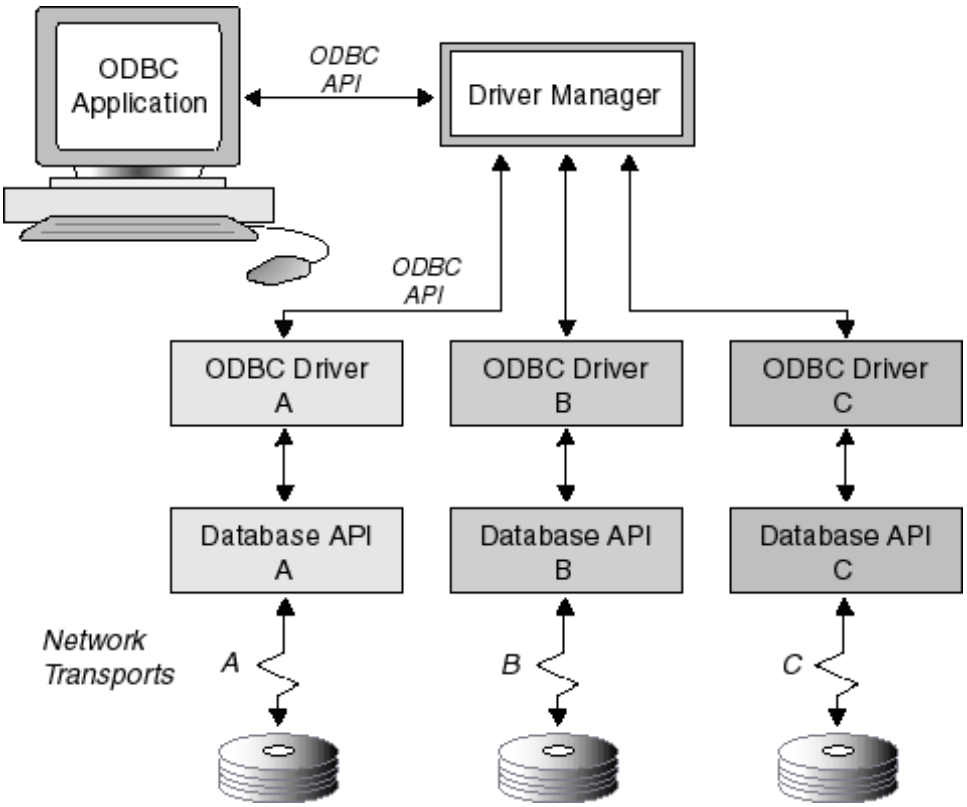
Open Database Connectivity (ODBC) provides a standard interface that allows one application to access many different data sources. The application's source code does not have to be recompiled for each data source. A database driver links the application to a specific data source. A database driver is a dynamic-link library that an application can invoke on demand to gain access to a particular data source. Therefore, the application can access any data source for which a database driver exists.

The ODBC interface defines the following:

- A library of ODBC function calls that allows an application to connect to a data source, execute structured query language (SQL) statements, and retrieve results.
- SQL syntax based on the SQL-99 specification.
- A standard set of error codes.
- A standard way to connect to and log in to a data source.
- A standard representation for data types.

The following figure shows the components of the ODBC model. The model begins with an ODBC application making a call to the Driver Manager through the ODBC application program interface (API). The Driver Manager can be either the Microsoft Driver Manager or the unixODBC Driver Manager. While using the ODBC API, the Driver Manager makes a call to the ODBC driver. The ODBC driver accesses the database over a network communications link using the database API. This figure shows an ODBC application accessing three separate databases.

Figure 1-1 Components of the ODBC Model



Related Topic

What Is the Oracle Database ODBC Driver

1.2 What Is the Oracle Database ODBC Driver

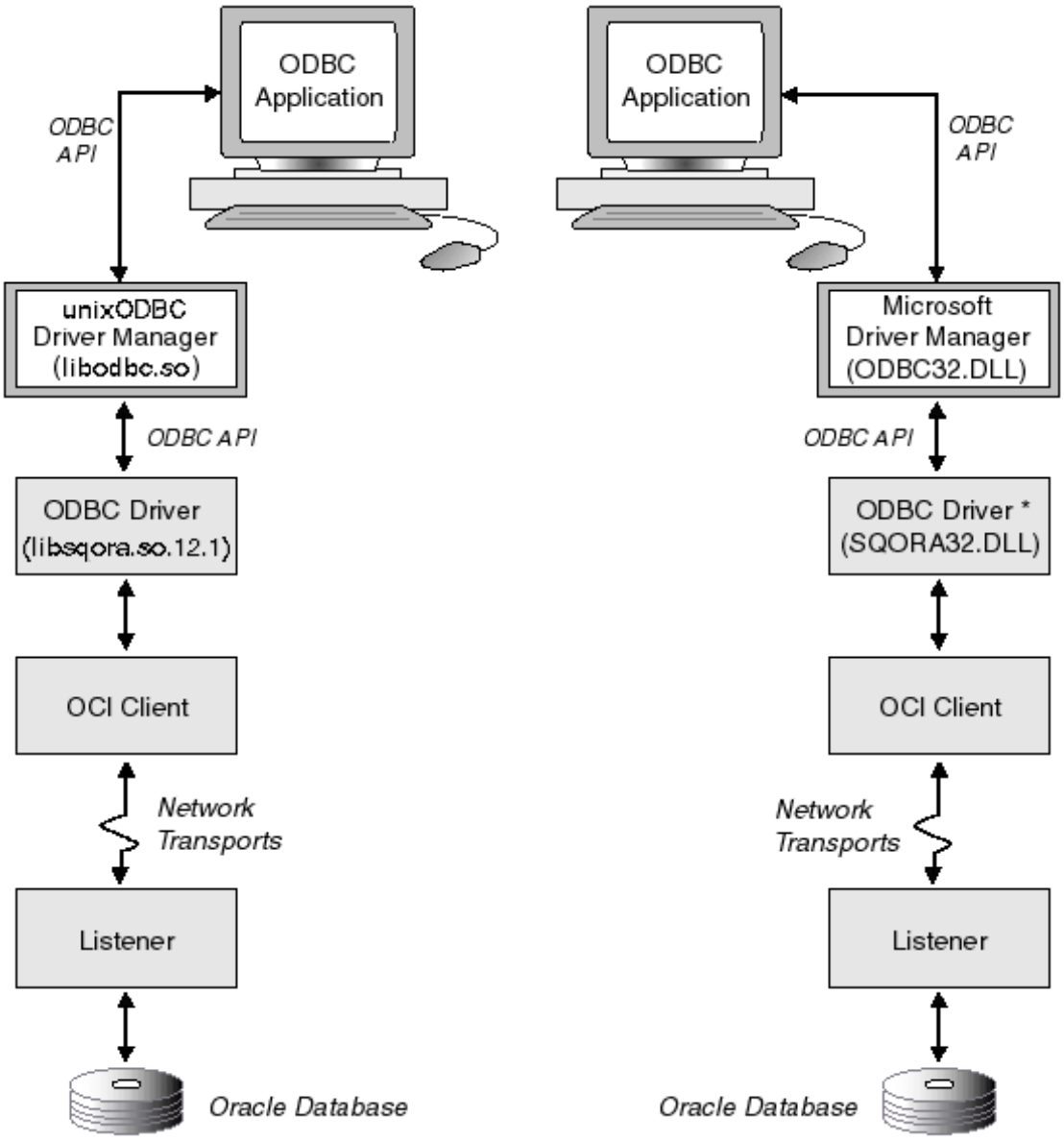
The Oracle Database ODBC driver enables ODBC applications on Microsoft Windows, and on UNIX platforms such as Linux, Solaris, and IBM AIX to have read and write access to Oracle® Databases through the ODBC interface using the Oracle Net Services software.

The Oracle Database ODBC driver uses the Oracle Call Interface (OCI) client and server software to submit requests to and receive responses from a data source. The Oracle Net Services communications protocol is used for communications between the OCI client and the Oracle server.

The Oracle Database ODBC driver translates the ODBC SQL syntax into the syntax that can be used to access a data source. When the results are returned from the data source, the Oracle Database ODBC driver translates them back to the ODBC SQL syntax.

The following figure shows the Oracle Database ODBC driver architecture as described in the preceding paragraphs.

Figure 1-2 Oracle Database ODBC Driver Architecture



* The Oracle ODBC Resource data definition language (DLL) file (`sqres`*`xx`*`.dll`), where *`xx`* represents the language abbreviation, contains all pertinent language information; the default resource file used is `sqresus.dll`.

The Oracle Database ODBC driver complies with ODBC version 3.52 specifications. For UNIX platforms, the ODBC driver is certified with unixODBC Driver Manager version 2.3.11.

Related Topics

Configuring the Data Source (Windows)

Configuration for UNIX Platforms

Connecting to a Data Source

Driver Conformance Levels

New and Changed Features

Files Created by the Installation

1.3 Certifications for Oracle Database ODBC Driver on Windows

The following table summarizes the Windows operating system versions on which the Oracle Database ODBC driver is certified.

Table 1-1 Oracle Database ODBC Driver Is Certified on Windows Operating Systems

Driver Version	Database Version	Operating Systems
ODBC Release 23ai, Version 23.3.0.0	As Supported by OCI	See Software Required.
ODBC Release 21c, Version 21.1	As Supported by OCI	See Software Required.
ODBC Release 19c, Version 19.1.0.0.0	As Supported by OCI	See Software Required.
ODBC Release 18c, Version 18.1.0.0.0	As Supported by OCI	See Software Required.
ODBC 12.2.0.1.0	As Supported by OCI	See Software Required.

1.4 Certifications for Oracle ODBC Driver on UNIX Platforms

Oracle has certified the Oracle Database ODBC driver for release 23.3 against Driver Manager (DM) 2.3.11 on the following listed UNIX platforms.

These UNIX platforms are shown in Table 1-2. On 64-bit UNIX platforms, DM 2.3.11 is built with the `-DBUILD_REAL_64_BIT_MODE -DSIZEOF_LONG=8 -fshort-wchar` flags and then certified.

Table 1-2 Certification Matrix for Oracle Database ODBC Driver on UNIX Platforms

Platform	32-bit/64-bit	UnixODBC DM version
Linux x86-64	32-bit, 64-bit	2.3.11
Solaris SPARC64	32-bit, 64-bit	2.3.11
AIX5L	32-bit, 64-bit	2.3.11
Solaris x64	32-bit, 64-bit	2.3.11
HPUX.IA64	32-bit, 64-bit	2.3.11
ZLinux	32-bit, 64-bit	2.3.11

To learn more about each operating system and Oracle Client software requirements, see the Installation guide of each platform.

1.5 Driver Conformance Levels

ODBC defines the conformance levels for drivers in two areas:

- ODBC application programming interface (API)
- ODBC SQL-99 syntax

The Oracle Database ODBC driver supports all core API functionality and a limited set of Level 1 and Level 2 functionalities.

The Oracle Database ODBC driver is broadly compatible with the SQL-99 Core specification, which is a superset of the SQL-92 Entry Level specification. Applications must call `SQLGetInfo` with the appropriate information type to retrieve a list of SQL-99 supported features.

> **See Also:**
>
> API Conformance for more information about the core API functionality support

2
Getting Started

This chapter guides you through the procedures required to install and configure the Oracle Database ODBC driver.

The Oracle Database ODBC driver enables applications to connect to Oracle Database from a Windows client as well as a UNIX client that use Microsoft Open Database Connectivity (ODBC) API to read from and write to Oracle Databases.

The Oracle Database ODBC driver distribution kit consists of Dynamic Link Libraries and shared libraries (for UNIX platforms), help file (on Windows and UNIX platforms), a copy of the license, and this product description. To use an ODBC-enabled application, the following software is required in addition to the Oracle Database ODBC driver:

- Oracle Client, such as full client install or Oracle Instant Client
- Oracle Database Server

> **See Also:**
>
> - The OCI documentation in *Oracle Call Interface Programmer's Guide* for more information about the OCI client and server software

Topics:

2.1 Summary of Steps

The summary of steps for installing and configuring the Oracle Database ODBC driver is as follows.

Linux and UNIX

1. Confirm that the system requirements have been met for the installation.

2. Download and install the Oracle Instant Client Basic or Light package.
3. Download and install the `unixODBC` Driver Manager.
4. Download and extract the Instant Client ODBC package into the Instant Client directory.
5. Configure the path for the driver's shared library and other environment variables.
6. Configure the Oracle net services (TNS service name) using the Oracle Net Configuration Assistant (NETCA) tool.
7. Run `odbc_update_ini.sh` from the Instant Client directory to configure the data source.
8. Set globalization variables, if required for your locale.

Windows

1. Confirm that the system requirements have been met for the installation.
2. Download and install the Oracle Instant Client Basic or Light package.
3. Download and extract the Instant Client ODBC package into the Instant Client directory.
4. Run `odbc_install.exe` from the Instant Client directory.
5. Configure the path for the driver's shared library and other environment variables.
6. Configure the Oracle net services (TNS service name) using the Oracle NETCA tool.
7. Run ODBC Data Source Administrator to configure the data source.

2.2 Installation

This section guides you through the procedures required to install the Oracle Database ODBC driver.

Oracle's Instant Client ODBC software is a standalone package that offers the full functionality of the Oracle Database ODBC driver (except the Oracle service for Microsoft Transaction Server) with a simple installation procedure.

The ODBC driver has Oracle's standard client-server version interoperability (see Support Doc ID 207303.1). For example, Instant Client ODBC 19c can connect to Oracle Database 11.2, or later.

Topics:

2.2.1 System Requirements

Before installing the ODBC driver, verify that the hardware, the target operating system and server versions are compatible for use with the ODBC driver.

Topics:

- Hardware Required
  The requirements for the Oracle Database ODBC driver system configuration for Windows and UNIX platforms.

2.2.1.1 Software Required

The Oracle Database ODBC driver was certified against the currently supported Windows and UNIX operating system versions, the most current release of Oracle Net Client and Oracle Universal Installer shipping with Oracle Database.

The Oracle Database ODBC driver was certified against the following versions of software:

- Windows operating system versions: Windows Server 2008, Windows Server 2008 R2, Windows 7, Windows 8, and Windows Server 2012
- UNIX operating system versions: 32-bit and 64-bit ports of Linux X86-64, AIX5L, Solaris.Sparc64, Solaris X64, HPUX.IA64, and ZLinux
- Oracle Net Client 12.2
- Oracle Universal Installer shipping with Oracle Database 12.2

2.2.1.2 Server Software Requirements

Oracle Database Server 12.2, or later, is the server software required to support ODBC-enabled applications that use the Oracle Database ODBC driver.

2.2.1.3 Hardware Required

The requirements for the Oracle Database ODBC driver system configuration for Windows and UNIX platforms.

The Oracle Database ODBC driver requires a system configuration that the certified Windows platforms as mentioned in Software Required supports, and on a few UNIX platforms, the hardware requirements are as documented in the Oracle Databse ODBC driver for UNIX Platforms Readme.

2.2.2 Installing Oracle Instant Client ODBC (Linux and UNIX)

1. Install the unixODBC Driver Manager from `unixODBC.org`. To install the ODBC Driver Manager, first download `.tar` file from `http://www.unixodbc.org/`.

   > **See Also:**
   >
   > Recommended unixODBC Driver Manager Versions for Linux and UNIX for more information about the recommended unixODBC Driver Manager versions

2. Navigate to package from OTN to download Oracle Instant Client Basic or Basic Lite.
3. Download the Instant Client ODBC package.
4. Unzip the Instant Client package to the desired location, for example: `/opt/oracle/instantclient_xx_yy`, or use `yum` to install the RPM packages on Linux.

> **See Also:**
>
> Installing Oracle Instant Client for more information about installing Oracle Instant Client

5. Unzip the ODBC package into the Instant Client folder, for example: `/opt/oracle/instantclient_xx_yy`, which is the same directory as the Basic or Basic Light package. Alternatively, if using the RPM package on Linux, install it with `yum`.

   - After the installation, set the environment variables, configure net services, and run `odbc_update_ini.sh` (from the Instant Client directory) to configure data sources.

     > **See Also:**
     >
     > Configuration for more post-installation configuration steps.

   - Set any Oracle globalization variables required for your locale. For example, on Linux, you could set `export NLS_LANG=JAPANESE_JAPAN.JA16EUC` to work in the JA16EUC character in Japanese.

     > **See Also:**
     >
     > Oracle Database Globalization Support Guide for more information.

- Recommended unixODBC Driver Manager Versions for Linux and UNIX

2.2.2.1 Recommended unixODBC Driver Manager Versions for Linux and UNIX

For Instant Client 23ai:

Platform	unixODBC Driver Manager Version
Linux 32bit, 64bit	2.3.11
Solaris SPARC64 32bit, 64bit	2.3.11
Solaris 32bit, 64bit	2.3.11
AIX 5L 32bit, 64bit	2.3.11
HP IA64 32bit, 64bit	2.3.11
z/Linux 31bit, 64bit	2.3.11

For Instant Client 21c:

Platform	unixODBC Driver Manager Version
Linux 32bit, 64bit	2.3.4
Solaris SPARC64 32bit, 64bit	2.3.4
Solaris 32bit, 64bit	2.3.4
AIX 5L 32bit, 64bit	2.3.4

Platform	unixODBC Driver Manager Version
HP IA64 32bit, 64bit	2.3.4
z/Linux 31bit, 64bit	2.3.4

For Instant Client 18c and 19c:

Platform	unixODBC Driver Manager Version
Linux 32bit, 64bit	2.3.4
Solaris SPARC64 32bit, 64bit	2.3.4
Solaris 32bit, 64bit	2.3.4
AIX 5L 32bit, 64bit	2.3.4
HP IA64 32bit, 64bit	2.3.4
z/Linux 31bit, 64bit	2.3.4

For Instant Client 12.1 and 12.2

Platform	unixODBC Driver Manager Version
Linux 32bit, 64bit	2.3.4
Solaris SPARC64 32bit, 64bit	2.3.4
Solaris 32bit, 64bit	2.3.4
AIX 5L 32bit, 64bit	2.3.4
HP IA64 32bit, 64bit	2.3.4
z/Linux 31bit, 64bit	2.3.4

2.2.3 Installing Oracle Instant Client ODBC (Windows)

1. Navigate to https://www.oracle.com/database/technologies/instant-client/microsoft-windows-32-downloads.html or https://www.oracle.com/database/technologies/instant-client/winx64-64-downloads.html
2. Download and install the Instant Client Basic or Basic Light package.
3. Download the Instant Client ODBC package.
4. Unzip the Instant Client folder (`instantclient_xx_yy`) to the desired location. For example: `C:\Users\app\`.
5. Unzip the ODBC package and put the contents of the zip file: `instantclient_xx_yy` into the Instant Client folder (in the same directory as your Instant Client Basic or Basic Light package).
6. Run `odbc_install.exe` from the Instant Client directory.

   This registers the ODBC driver with the ODBC Data Sources GUI.
7. To install with Japanese language support, execute the command `odbc_install.exe JA`.

   After the installation, set the environment variables, configure net services, and configure the data sources.

> **See Also:**
>
> Configuration for more post-installation configuration steps.

2.2.4 Content of the Oracle Instant Client ODBC Package

Description	Linux and UNIX	Windows
Oracle Database ODBC driver shared library	`libsqora.so.XX.Y.` For example `libsqora.so.23.1`	`sqora32.dll`
Installation file	`odbc_update_ini.sh`	`odbc_install.exe`, `odbc_uninstall.exe`
Oracle Database ODBC driver configuration dialog box (GUI)	Not available	`sqoras32.dll`, `sqresus.dll`, `sqresja.dll`
Help System	`help/`	`help/`

2.2.5 Files Created by the Installation

The following table describes the files that are installed by the Oracle Database ODBC driver kit.

Table 2-1 Files Installed by the Oracle Database ODBC driver Kit

Description	File Name for Windows Installation	File Name for UNIX Installation
Oracle ODBC Database Access DLL	`sqora32.dll`	libsqora.so.*nn.n* (where *nn.n* reflects a version number; for example, libsqora.so.23.1)
Oracle Database ODBC driver Setup DLL	`sqoras32.dll`	None
Oracle ODBC Resource DLL	`sqresus.dll`	None
Oracle ODBC Resource DLL for Japanese	`sqresja.dll`	None
Oracle Database ODBC driver message file	`oraodbcus.msb`	`oraodbcus.msb`
Oracle Database ODBC driver message file for Japanese	`oraodbcja.msb`	`oraodbcja.msb`

Table 2-1 (Cont.) Files Installed by the Oracle Database ODBC driver Kit

Description	File Name for Windows Installation	File Name for UNIX Installation
Oracle Database ODBC driver release notes	*Oracle Database ODBC Driver Release Notes*	*Oracle Database ODBC Driver Release Notes*
Oracle Database ODBC driver Instant Client Release Notes	`ODBC_IC_Readme_Win.html`	`ODBC_IC_Readme_Unix.html`
Oracle Database ODBC driver help file	`sqora.htm`	`sqora.htm`
Oracle Database ODBC driver help file for Japanese	`sqora.htm`	`sqora.htm`
Oracle Database ODBC driver Instant Client install script	`odbc_install.exe`	`odbc_update_ini.sh`
Oracle Database ODBC driver Instant Client uninstall script	`odbc_uninstall.exe`	None

Microsoft Driver Manager and Administrator Files

See the Microsoft ODBC 3.52 Software Development Kit and Programmer's Reference for the list of files that are installed with Microsoft's ODBC 3.52 Components.

The Microsoft ODBC components are packages in the Microsoft Data Access Component (MDAC) kit. The Oracle Database ODBC driver on Windows has been tested using MDAC version 2.8.

unixODBC Driver Manager and Administrator Files

See the unixODBC readme and INSTALL files for the list of files that are installed with unixODBC Driver Manager.

See Also:

- MDAC Kit to download MDAC kit
- Unix ODBC Driver Manager to download unixODBC Driver

2.3 Configuration

This section guides you through the procedures required to configure the Oracle Database ODBC driver.

Post installation, you must set environment variables for ODBC applications, configure network database services, and configure the data sources on Windows and UNIX platforms.

Use the Microsoft ODBC Administrator to configure your Oracle Database ODBC driver data sources on Windows. For more information, see the information about configuring the data source in Configuring the Data Source.

To configure the Oracle Database ODBC driver data source on a UNIX Client, see Configuration for UNIX Platforms.

Topics:

- Environment Setup for ODBC Applications
- Configuring Oracle Net Services
- Configuration for UNIX Platforms
  Complete these post-installation configuration tasks for the Oracle Database ODBC driver on UNIX platforms.
- Configuration for Windows
  Complete these post-installation configuration tasks for the Oracle Database ODBC driver on Windows.
- Reducing Lock Timeout

2.3.1 Environment Setup for ODBC Applications

An ODBC application must load the Oracle Instant Client ODBC driver's shared library file to connect to Oracle Database. On Linux/Unix, the directory path of the shared library file `libsqora.so.XX.Y` (for example `libsqora.so.19.1`) should be set in the `LD_LIBRARY_PATH` environment variable, or in a platform equivalent. It can also be configured in `/etc/ld.so.conf`. On Windows, the directory path of the shared library file should be set in the `PATH` environment variable.

See Also:

- Content of the Oracle Instant Client ODBC Package for more information about the ODBC driver's shared library
- Environment Variables for Oracle Instant Client for more information about related environment variables, such as `TNS_ADMIN`, `TWO_TASK`, and `LOCAL`
- Database Connection Strings in *Oracle Call Interface Programmer's Guide* for information about setting up a database connection string

2.3.2 Configuring Oracle Net Services

Before configuring the data source, you must configure network database services to ensure that there is an entry for each Transparent Network Substrate (TNS) Service Name. To do this, use the Oracle Net Configuration Assistant (NETCA) tool.

TNS service name is the location of the Oracle database from which the ODBC driver retrieves data.

Using NETCA, you can create an entry in the `tnsnames.ora` file for each TNS Service Name.

> **Note:**
>
> NETCA is installed when you install Oracle Net Services.

> **See Also:**
>
> Using Oracle Net Configuration Assistant to Configure Network Components in *Oracle Database Net Services Administrator's Guide* for more information about using the NETCA tool.

2.3.3 Configuration for UNIX Platforms

Complete these post-installation configuration tasks for the Oracle Database ODBC driver on UNIX platforms.

1. Run *install-home*`/odbc/utl/odbc_update_ini.sh` to configure the Oracle Database ODBC driver on UNIX.

   The utility `odbc_update_ini.sh` takes four command-line arguments:

   - *arg-1*: Complete path where `unixODBC DM` has been installed.
   - *arg-2*: Complete path of driver install location (optional); if this argument is not passed, the driver path is set to the directory from where the utility is run.
   - *arg-3*: Driver name (optional); if this argument is not passed, the driver name is set to the downloaded version.
   - *arg-4*: Data Source Name or DSN (optional); if no value is passed, the DSN is set to the downloaded version.

   > **See Also:**
   >
   > Usage for detailed information about how to use `odbc_update_ini.sh`.

2. Update and verify values of environment variables such as: `PATH`, `LD_LIBRARY_PATH`, `LIBPATH`, and `TNS_ADMIN`.

- Usage

2.3.3.1 Usage

```
odbc_update_ini.sh <ODBCDM_Home> [<Install_Location> <Driver_Name> <DSN>
<ODBCINI>]
```

Table 2-2 Parameter Descriptions

Parameter	Required/ Optional	Description
`ODBCDM_Home`	Required	unixODBC Driver Manager home directory path.
`Install_Location`	Optional	Oracle Instant Client directory path. The default path is the current directory.
`Driver_Name`	Optional	Driver name to identify the Oracle Database ODBC driver residing in current Oracle Instant Client home. The default name is like "Oracle 23ai ODBC driver."
`DSN`	Optional	Sets ODBC Data Source Name (DSN). The default name is "OracleODBC-23ai."
`ODBCINI`	Optional	Directory path of the `.odbc.ini` file. The default path is the user's home directory, for example `~/.odbc.ini`.

2.3.4 Configuration for Windows

Complete these post-installation configuration tasks for the Oracle Database ODBC driver on Windows.

Topics:

- Configuring the Data Source
- Oracle Database ODBC Driver Configuration Dialog Box

2.3.4.1 Configuring the Data Source

> **Note:**
>
> The following configuration steps are for Windows users. Unix users must use the `odbc_update_ini.sh` file to create a Data Source Name (DSN).

After installing the Oracle Database ODBC driver and Configuring Oracle Net Services, and before using the Oracle Database ODBC driver, you must configure the data source.

Before an application can communicate with the data source, you must provide configuration information. The configuration information informs the Oracle Database ODBC driver as to which information you want to access.

The data source consists of the data that you want to access, its associated operating system, database management system, and network platform used to access the database

management system. The data source for requests submitted by the Oracle Database ODBC driver is an Oracle Database and supports transports available under Oracle Net Services.

To configure or add an Oracle data source:

After you have installed the Oracle Database ODBC driver, use the ODBC Data Source Administrator to configure or add an Oracle data source for each of your Oracle Databases. the Oracle Database ODBC driver uses the information you enter when you add the data source to access the data. Follow these steps:

1. From the **start** menu, select **Programs, Administrative Tools, Data Sources (ODBC)**.

   In the ODBC Data Source Administrator dialog box, in the **Drivers** tab, a list of installed drivers is displayed. Ensure that the **Drivers** tab displays the Oracle Database ODBC driver that you just installed.
2. On the **System DSN** tab, click **Add** to display the Create New Data Source dialog box.
3. In the Create New Data Source dialog box, from the list of installed drivers, select the Oracle Database ODBC driver for which you want to set up a data source.
4. Click **Finish**.

   The Oracle ODBC Driver Configuration dialog box is displayed. You must enter the DSN and TNS Service Name. You can provide the other information requested in the dialog box, or you can leave the fields blank and provide the information when you run the application.
5. After you have entered the data, click **OK** or click **Return**.

You can change or delete a data source at any time. The following subtopics explain how to add, change, or delete a data source.

To modify an Oracle data source:

1. From the **start** menu, select **Programs, Administrative Tools, Data Sources(ODBC)**.
2. In the ODBC Data Source Administrator dialog box, select the data source from the **Data Sources** list and click **Configure**.

   The Oracle ODBC Driver Configuration dialog box is displayed.
3. In the Oracle ODBC Driver Configuration dialog box, modify the option values as necessary and click **OK**.

To delete an Oracle data source:

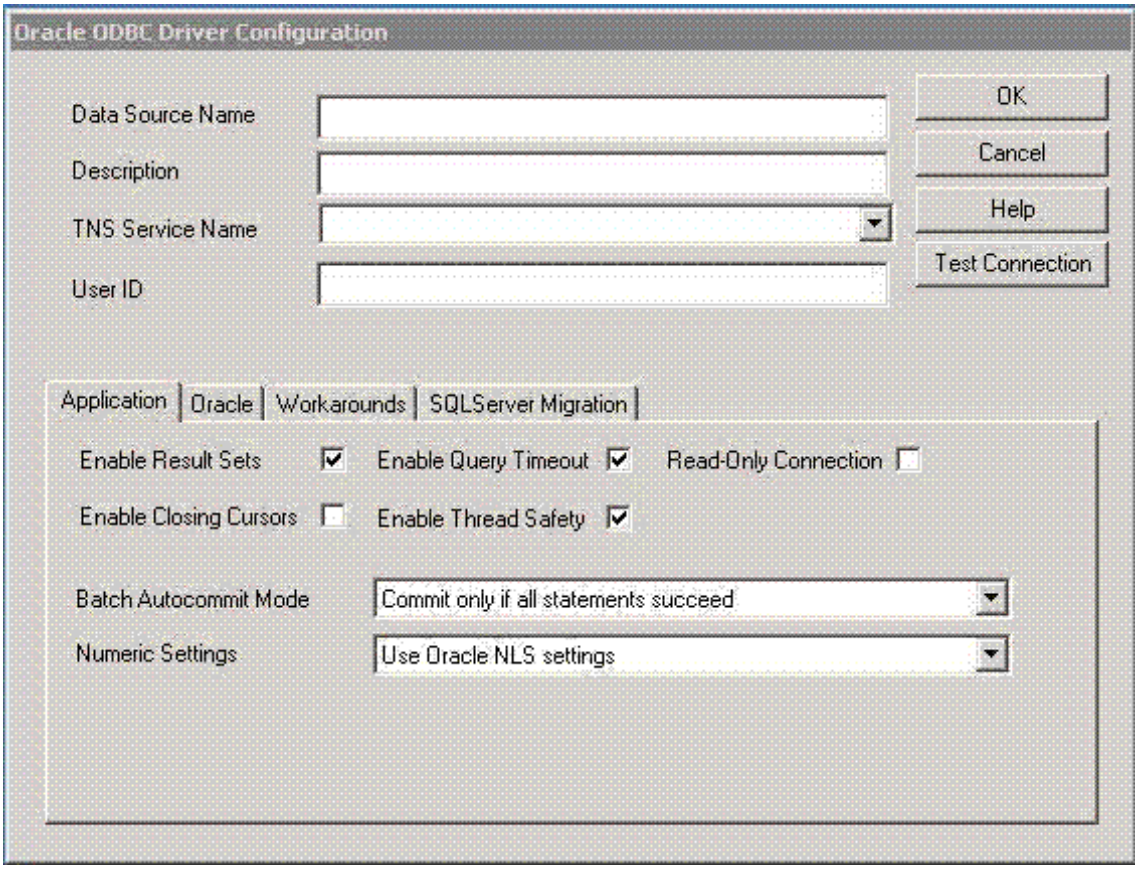
1. From the **start** menu, select **Programs, Administrative Tools, Data Sources(ODBC)**.
2. In the ODBC Data Source Administrator dialog box, select the data source you want to delete from the **Data Sources** list.
3. Click **Remove**, and then click **Yes** to confirm the deletion.

2.3.4.2 Oracle Database ODBC Driver Configuration Dialog Box

> **Note:**
>
> The Oracle Database ODBC Driver Configuration dialog box is available only to Microsoft Windows users.

The following screenshot shows an example of the Oracle Database ODBC Driver Configuration dialog box.

Figure 2-1 Oracle ODBC Driver Configuration Dialog Box



The following list is an explanation of the main setup options and fields found on the Oracle Database ODBC Driver Configuration dialog box shown in the preceding graphic. The tabs found on the lower half of this dialog box are described in the subsequent topics.

- **Data Source Name (DSN)** - The name used to identify the data source to ODBC. For example, "odbc-pc". You must enter a DSN.
- **Description** - A description or comment about the data in the data source. For example, "Hire date, salary history, and current review of all employees." The Description field is optional.
- **TNS Service Name** - The location of the Oracle database from which the ODBC driver will retrieve data. This is the same name entered in Configuring Oracle Net Services using the Oracle Net Configuration Assistant (NETCA). For more information, see the NETCA documentation and About Using Oracle Database ODBC Driver for the First Time. The TNS Service Name can be selected from a pull-down list of available TNS names. For example, "ODBC-PC." You must enter a TNS Service Name.
- **User ID** - The user name of the account on the server used to access the data. For example, "scott." The User ID field is optional.

You must enter the DSN and the TNS Service Name. You can provide the other information requested in the dialog box or you can leave the fields blank and provide the information when you run the application.

In addition to the main setup options previously described, there is a **Test Connection** option available. The **Test Connection** option verifies whether the ODBC environment is configured properly, by connecting to the database specified by the DSN definition. When you click **Test Connection**, you are prompted for the username and password.

For an explanation of the options tabs found on the lower half of the Oracle Database ODBC Driver Configuration dialog box, click any of these links:

Application Options

Oracle Options

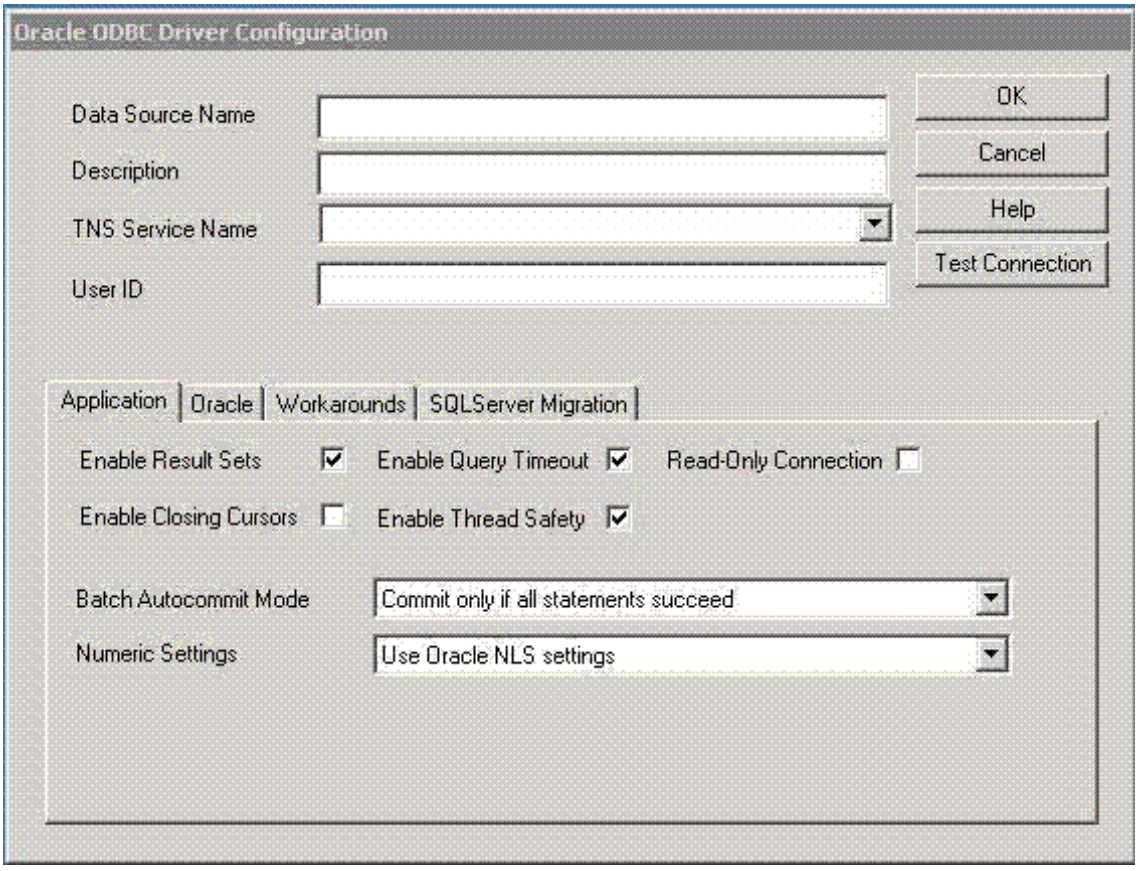
Workarounds Options

SQL Server Migration Options

Application Options

The following screenshot shows an example of the **Application** options tab found on the Oracle ODBC Driver Configuration dialog box.

Figure 2-2 The Application Options Tab of the Oracle ODBC Driver Configuration Dialog Box



The following list is an explanation of the fields found on the **Application** options tab shown in the preceding graphic:

- **Enable Result Sets** enables the processing of Oracle Result Sets. If Result Sets are not required for your application, Result Set support can be disabled. There is a small

performance penalty for procedures called from packages not containing Result Sets. Result Sets are enabled by default.

- **Enable Query Timeout** enables query timeout for SQL queries. By default, the Oracle Database ODBC driver supports the `SQL_ATTR_QUERY_TIMEOUT` attribute for the SQLSetStmtAttr function. If this box is not checked, the Oracle Database ODBC driver responds with a "not capable" message. Query Timeout is enabled by default.
- **Read-Only Connection** - Check this box to force read-only access. The default is write access.
- **Enable Closing Cursors** enables closing cursors. By default, closing cursors is disabled (the field is empty), meaning a call to close a cursor does not force the closing of OCI cursors when this behavior is not desired because it can cause an unnecessary performance hit. Enable closing cursors when you want to force the closing of OCI cursors upon a call to close a cursor.

> **Note:**
>
> There is an impact on performance each time a cursor is closed.

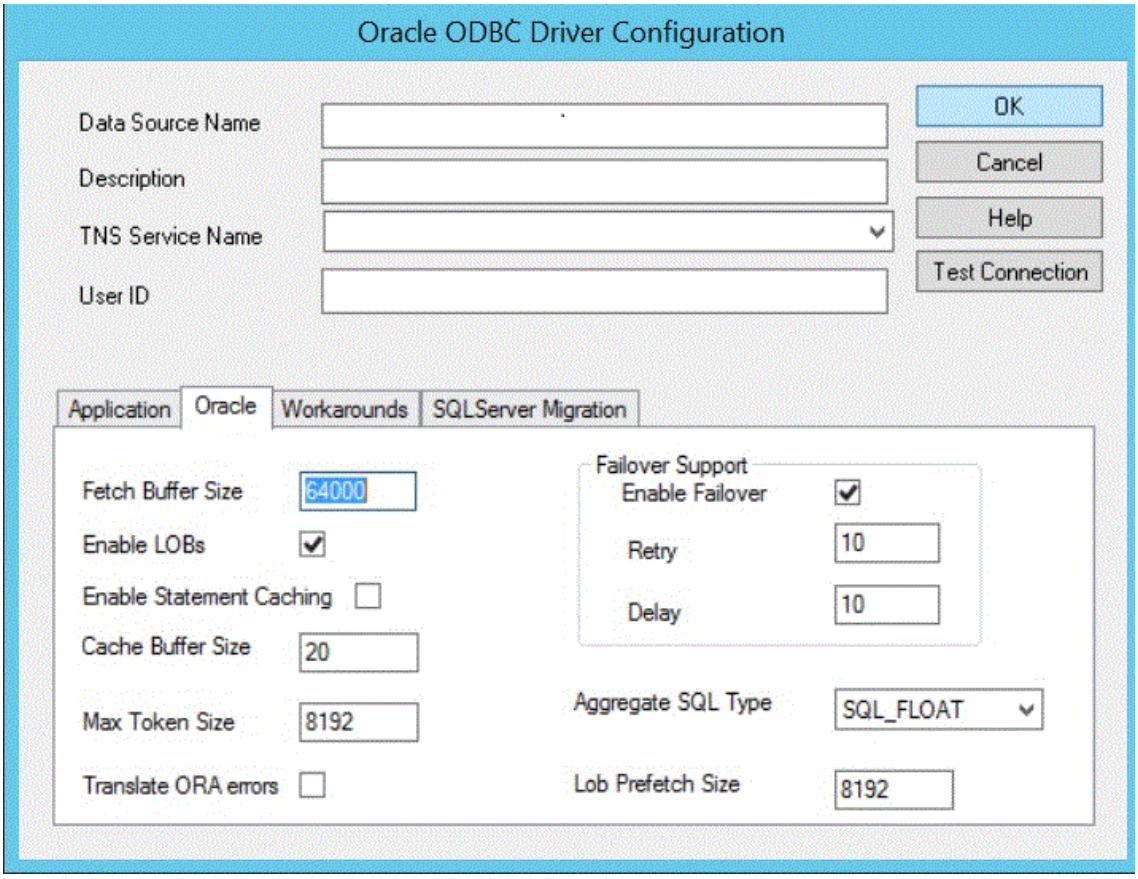
- **Enable Thread Safety** - Thread safety can be disabled for a data source. If thread safety is not required, disabling this option eliminates the overhead of using thread safety. By default, thread safety is enabled.
- **Batch Autocommit Mode** - By default, commit is executed only if all statements succeed.
- **Numeric Settings** allows you to choose the numeric settings that determine the decimal and group separator characters when receiving and returning numeric data that is bound as strings. This option allows you to choose Oracle NLS settings (the default setting), Microsoft default regional settings (to provide a way to mirror the Oracle OLE DB driver's behavior for greater interoperability), or US numeric settings (which are necessary when using MS Access or DAO (Database Access Objects) in non-US environments).

> **See Also:**
>
> Oracle ODBC Driver Configuration Dialog Box for the main configuration setup options

Oracle Options

The following screenshot shows an example of the **Oracle** options tab found on the Oracle Database ODBC Driver Configuration dialog box.

Figure 2-3 The Oracle Options Tab of the Oracle ODBC Driver Configuration Dialog Box



The following list is an explanation of the fields found on the **Oracle** options tab shown in the preceding graphic:

- **Fetch Buffer Size** - The amount of memory used to determine how many rows of data the ODBC driver prefetches at a time from an Oracle database regardless of the number of rows the application program requests in a single query. However, the number of prefetched rows depends on the width and number of columns specified in a single query. Applications that typically fetch fewer than 20 rows of data at a time improve their response time, particularly over slow network connections or on heavily loaded servers. Setting the Fetch Buffer Size too high can make response time worse or consume large amounts of memory.

> **Note:**
>
> When `LONG` and LOB data types are present, the number of rows prefetched by the ODBC driver is not determined by the Fetch Buffer Size. The inclusion of the `LONG` and LOB data types minimizes the performance improvement and could result in excessive memory use. The ODBC driver disregards the Fetch Buffer Size and prefetches a set number of rows only in the presence of the `LONG` and LOB data types.

- **Enable LOBs** - Enables the writing of Oracle LOBs. If writing Oracle LOBs is not required for your application, LOB support can be disabled. There is a small performance penalty for insert and update statements when LOBs are enabled. LOB writing is enabled by default but disabled for Oracle databases that do not support the LOB data type.
- **Enable Statement Caching** - Enables statement caching feature, which increases the performance of parsing the query, in case the user has to parse the same text of query and related parameters multiple times. The default is disabled.
- **Cache Buffer Size** - The statement cache has a maximum size (number of statements) that can be modified by an attribute on the service context, `OCI_ATTR_STMTCACHESIZE`. The default cache buffer size is 20 that are used only if statement caching option is enabled. Setting cache buffer size to 0 disables statement caching feature.
- **Max Token Size** - Sets the token size to the nearest multiple of 1 KB (1024 bytes) beginning at 4 KB (4096 bytes). The default size is 8 KB (8192 bytes). The maximum value that can be set is 128 KB (131068 bytes).
- **Translate ORA errors** - Any migrated third party ODBC application, which is using the SQL Translation Framework feature, expects that the errors returned by the server need to be in their native database format, then users can enable this option to receive native errors based on the error translation registered with SQL Translation Profile.
- The **Failover** area of the **Oracle** options tab contains the following fields:
  - **Enable Failover** - Enables Oracle Fail Safe and Oracle Parallel Server failover retry. This option in an enhancement to the failover capabilities of Oracle Fail Safe and Oracle Parallel Server. Enable this option to configure additional failover retries. The default is enabled.
  - **Retry** - The number of times the connection failover is attempted. The default is 10 attempts.
  - **Delay** - The number of seconds to delay between failover attempts. The default is 10 seconds.
- **Aggregate SQL Type** - Specifies the number type return for aggregate functions: `SQL_FLOAT`, `SQL_DOUBLE`, or `SQL_DECIMAL`.
- **Lob Prefetch Size** - Sets the amount of LOB data (in bytes) to prefetch from the database at one time. The default size is 8192.

> **Note:**
>
> Oracle Fail Safe is deprecated and it can be desupported and unavailable in a future release. Oracle recommends that you evaluate other single-node failover options, such as Oracle RAC One Node.

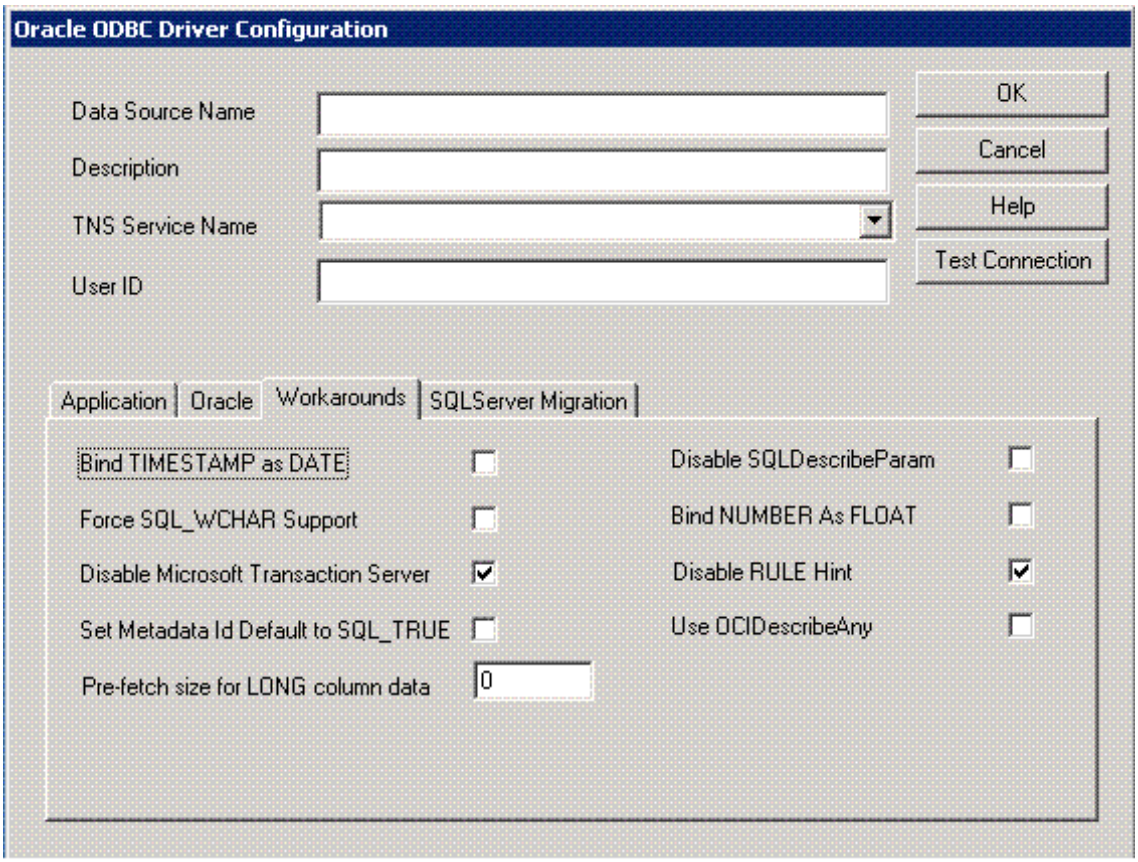
> **Note:**
>
> See the Oracle Fail Safe and Oracle Parallel Server documentation on how to set up and use both of these products.

> **See Also:**
>
> Oracle ODBC Driver Configuration Dialog Box for the main configuration setup options

Workarounds Options

The following screenshot shows an example of the **Workarounds** options tab found on the Oracle Database ODBC Driver Configuration dialog box.

Figure 2-4 The Workarounds Options Tab of the Oracle ODBC Driver Configuration Dialog Box



The following list is an explanation of the fields found on the **Workarounds** options tab shown in the preceding graphic:

- **Bind TIMESTAMP as DATE** - Check this box to force the Oracle Database ODBC driver to bind `SQL_TIMESTAMP` parameters as the Oracle `DATE` type instead of as the Oracle `TIMESTAMP` type (the default).
- **Force SQL_WCHAR Support** - Check this box to enable SQLDescribeCol, SQLColumns, and SQLProcedureColumns to unconditionally return the data type of `SQL_WCHAR` for `SQL_CHAR` columns; `SQL_WVARCHAR` for `SQL_VARCHAR` columns; and `SQL_WLONGVARCHAR` for `SQL_LONGVARCHAR` columns. This feature enables Unicode support in applications that rely on the results of these ODBC calls (for example, ADO). This support is disabled by default.

- **Disable Microsoft Transaction Server** - Clear the check in this box to enable Microsoft Transaction Server (MTS) support. By default, MTS support is disabled.
- **Set Metadata Id Default to SQL_TRUE** - Check this box to change the default value of the `SQL_ATTR_METADATA_ID` connection and statement attribute at connection time to `SQL_TRUE`. Under normal circumstances, `SQL_ATTR_METADATA_ID` would default to `SQL_FALSE`. ODBC calls made by the application to specifically change the value of the attribute after connection time are unaffected by this option and complete their functions as expected. By default, this option is off.
- **Prefetch size for LONG column data** - Set this value to prefetch `LONG` or `LONG RAW` data to improve performance of ODBC applications. This enhancement improves the performance of Oracle ODBC driver up to 10 times, depending on the prefetch size set by the user. The default value is 0. The maximum value that you can set is 64 KB (65536 bytes).

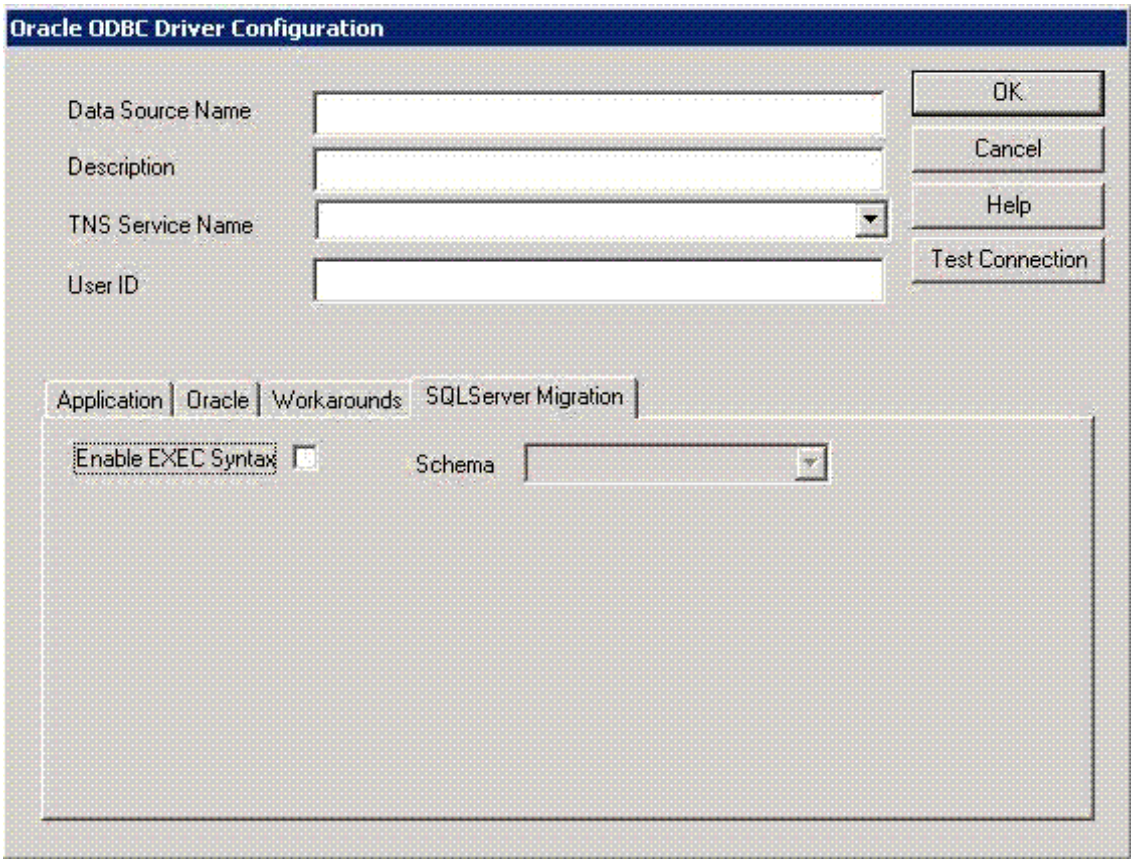
  If the value of prefetch size is greater than 65536, the data fetched is only 65536 bytes. If you have `LONG` or `LONG RAW` data in the database that is greater than 65536 bytes, then set the prefetch size to 0 (the default value), which causes single-row fetch and fetches complete `LONG` data. If you pass a buffer size less than the prefetch size in nonpolling mode, a data truncation error occurs if the `LONG` data size in the database is greater than the buffer size.
- **Disable SQLDescribeParam** - If the `SQLDescribeParam` function is enabled, the `SQL_VARCHAR` data type is returned for all parameters. If the Force SQL_WCHAR Support function is also enabled, the `SQL_WVARCHAR` data type is returned for all parameters. By default, this function is enabled.
- **Bind NUMBER as FLOAT** - Check this box to force the Oracle Database ODBC driver to bind `NUMBER` column containing `FLOAT` data as Float instead of as the Binary Float (the default).
- **Disable RULE Hint** - Clear the check in this box to enable `RULE Hint` specified with catalogue queries. By default, `RULE Hint` option is disabled.
- **Use OCIDescribeAny** - Check this box to gain a performance improvement by forcing the driver to use `OCIDescribeAny()` when an application makes heavy calls to small packaged procedures that return `REF CURSORS`.

> **See Also:**
>
> - Implementation of Data Types (Advanced) for more information about **DATE and TIMESTAMP**
> - Implementation of ODBC API Functions for more information about the `SQL_ATTR_METADATA_ID` attribute
> - Oracle ODBC Driver Configuration Dialog Box for the main configuration setup options
> - About Using OCIDescribeAny() for more information about `OCIDescribeAny()`

SQL Server Migration Options

The following screenshot shows an example of the **SQL Server Migration** options tab found on the Oracle ODBC Driver Configuration dialog box.

Figure 2-5 The SQL Server Migration Options Tab of the Oracle ODBC Driver Configuration Dialog Box



The fields of the **SQL Server Migration** options tab in the preceding graphic are:

- **EXEC Syntax Enabled** enables support for SQL Server EXEC syntax. A subprogram call specified in an EXEC statement is translated to its equivalent Oracle subprogram call before being processed by an Oracle database server. By default this option is disabled.
- **Schema** is the translated Oracle subprogram assumed to be defined in the user's default schema. However, if all subprograms from the same SQL Server database are migrated to the same Oracle schema with their database name as the schema name, then set this field to database. If all subprograms owned by the same SQL Server user are defined in the same Oracle schema, then set this field to owner. This field is empty by default.

> **See Also:**
>
> Oracle ODBC Driver Configuration Dialog Box for the main configuration setup options

2.3.5 Reducing Lock Timeout

An Oracle server waits indefinitely for lock conflicts between transactions to be resolved. You can limit the amount of time that an Oracle server waits for locks to be resolved by setting the Oracle Database ODBC driver's `LockTimeOut` entry in the `oraodbc.ini` file. The value you

enter for the `LockTimeOut` parameter is the number of seconds after which an Oracle server times out if it cannot obtain the requested locks. In the following example, the Oracle server times out after 60 seconds:

```
[Oracle ODBC Driver Common]
LockTimeOut=60
```

2.4 Patching Oracle Instant Client ODBC

This section guides you through the procedures required to patch Oracle Instant Client ODBC.

> **Note:**
>
> - Back up the Oracle Database ODBC driver shared library and other files before patching them.
> - You must rebuild your Oracle Instant Client packages and libraries as part of the patching process.

Topics:

- Patching Oracle Instant Client ODBC on Linux and UNIX Method 1
- Patching Oracle Instant Client ODBC on Linux and UNIX Method 2
- Patching on Windows

2.4.1 Patching Oracle Instant Client ODBC on Linux and UNIX Method 1

Patching the Instant Client ODBC driver on Linux/UNIX can be done by generating the Instant Client ODBC package and Basic or Basic Light package in a patched ORACLE_HOME. These new packages should then be unzipped into the Instant Client directory that needs to be patched. This method of patching is recommended.

> **See Also:**
>
> Patching Oracle Instant Client explains the procedure for patching and generating Instant Client Basic and Basic Light packages and Instant Client ODBC.

2.4.2 Patching Oracle Instant Client ODBC on Linux and UNIX Method 2

Alternatively, to patch Oracle Instant Client ODBC Driver, copy the following files from a patched ORACLE_HOME:

- ODBC driver shared library file:
  - For 23ai: libsqora.so.23.1
  - For 21c: libsqora.so.21.1
  - For 19c: libsqora.so.19.1
  - For 18c: libsqora.so.18.1

  - For 12c: libsqora.so.12.1
- Required additional files when using Oracle Instant Client Basic:
  - For 23ai: libociei.so, libclntshcore.so.23.1, libclntsh.so.23.1, libnnz23.so, libons.so
  - For 21c: libociei.so, libclntshcore.so.21.1, libclntsh.so.21.1, libnnz21.so, libons.so
  - For 19c: libociei.so, libclntshcore.so.19.1, libclntsh.so.19.1, libnnz19.so, libons.so
  - For 18c: libociei.so, libclntshcore.so.18.1, libclntsh.so.18.1, libnnz18.so, libons.so
  - For 12c: libociei.so, libclntshcore.so.12.1, libclntsh.so.12.1, libnnz12.so, libons.so
- Required additional files when using Oracle Instant Client Basic Light:
  - For 23ai: libclntsh.so.23.1, libclntshcore.so.23.1, libociicus.so, libnnz23.so, libons.so
  - For 21c: libclntsh.so.21.1, libclntshcore.so.21.1, libociicus.so, libnnz21.so, libons.so
  - For 19c: libclntsh.so.19.1, libclntshcore.so.19.1, libociicus.so, libnnz19.so, libons.so
  - For 18c: libclntsh.so.18.1, libclntshcore.so.18.1, libociicus.so, libnnz18.so, libons.so
  - For 12c: libclntsh.so.12.1, libclntshcore.so.12.1, libociicus.so, libnnz12.so, libons.so

2.4.3 Patching on Windows

You can patch Instant Client ODBC Driver on Windows only manually by copying the ODBC driver shared library files and supporting library files from a patched ORACLE_HOME or from an unpacked Oracle Database Bundle patch. These should be copied into the Instant Client directory. Generating an Instant Client ODBC package is not available on Windows.

The files that must be copied to the Instant Client directory:

- ODBC driver shared library files: sqora32.dll, sqoras32.dll, sqresus.dll, sqresja.dll
- Required additional files when using Oracle Basic Instant Client:
  - For 23ai: oraociei23.dll, orannzsbb23.dll, oci.dll, oraons.dll, ociw32.dll, oraociei23.sym, orannzsbb23.sym, oci.sym, ociw32.sym
  - For 21c: oraociei21.dll, orannzsbb21.dll, oci.dll, oraons.dll, ociw32.dll, oraociei21.sym, orannzsbb21.sym, oci.sym, ociw32.sym
  - For 19c: oraociei19.dll, orannzsbb19.dll, oci.dll, oraons.dll, ociw32.dll, oraociei19.sym, orannzsbb19.sym, oci.sym, ociw32.sym
  - For 18c: oraociei18.dll, orannzsbb18.dll, oci.dll, oraons.dll, ociw32.dll, oraociei18.sym, orannzsbb18.sym, oci.sym, ociw32.sym
  - For 12c: oraociei12.dll, orannzsbb12.dll, oci.dll, oraons.dll, ociw32.dll, oraociei12.sym, orannzsbb12.sym, oci.sym, ociw32.sym
- Required additional files when using Oracle Basic Light Instant Client:
  - For 23ai: oraociicus23.dll, orannzsbb23.dll, oci.dll, oraons.dll, ociw32.dll, oraociicus23.sym, orannzsbb23.sym, oci.sym, ociw32.sym
  - For 21c: oraociicus21.dll, orannzsbb21.dll, oci.dll, oraons.dll, ociw32.dll, oraociicus21.sym, orannzsbb21.sym, oci.sym, ociw32.sym
  - For 19c: oraociicus19.dll, orannzsbb19.dll, oci.dll, oraons.dll, ociw32.dll, oraociicus19.sym, orannzsbb19.sym, oci.sym, ociw32.sym
  - For 18c: oraociicus18.dll, orannzsbb18.dll, oci.dll, oraons.dll, ociw32.dll, oraociicus18.sym, orannzsbb18.sym, oci.sym, ociw32.sym

– For 12c: oraociicus12.dll, orannzsbb12.dll, oci.dll, oraons.dll, ociw32.dll, oraociicus12.sym, orannzsbb12.sym, oci.sym, ociw32.sym

> **Note:**
>
> While copying from the Oracle Database Bundle patch, some of the aforementioned files may be missing. This implies that those files are unchanged and do not need to be patched.

2.5 Uninstallation

This section takes you through the steps required to uninstall the Oracle Database ODBC driver.

Topics:

- Uninstalling Oracle Instant Client ODBC on Linux and UNIX
  Use the following procedure to uninstall Oracle Instant Client ODBC on Linux and UNIX.
- Uninstalling Oracle Instant Client ODBC on Windows
  Use the following procedure to uninstall Instant Client ODBC on Windows.

2.5.1 Uninstalling Oracle Instant Client ODBC on Linux and UNIX

Use the following procedure to uninstall Oracle Instant Client ODBC on Linux and UNIX.

1. Remove the Oracle Database ODBC driver entry from the `odbcinst.ini` file of the unixODBC Driver Manager.

   The default name of this entry is like: `[Oracle 19c ODBC driver]`.
2. Remove the DSN entry of the Oracle Database ODBC driver from `odbc.ini`.

   The default name of the DSN entry is like `[OracleODBC-19c]`.
3. Delete all files and directories in the Instant Client ODBC directory.

2.5.2 Uninstalling Oracle Instant Client ODBC on Windows

Use the following procedure to uninstall Instant Client ODBC on Windows.

1. Remove the DSN associated with the Oracle Database ODBC driver in the ODBC Data Source Administrator (`odbcad32`) console.
2. Execute the `odbc_uninstall.exe` file from the Instant Client ODBC directory.
3. Delete all files and directories in the Instant Client ODBC directory.

3
Basic Connection Steps

This chapter guides you through the steps required to connect your ODBC application to an Oracle data source.

Topics:

3.1 Connecting to an Oracle Data Source

To connect to a data source, the Oracle Database ODBC driver requires that the OCI client software be installed on your computer and the corresponding listener be running on the Oracle server. Oracle Net Services for Windows is a Dynamic Linked Library (DLL) based application.

> **See Also:**
>
> *Oracle Database Net Services Administrator's Guide* and *Oracle Database Net Services Reference* for more information about Oracle Net Services.

As part of the connection process, an application can prompt you for information. If an application prompts you for information about an Oracle data source, do the following:

1. In the **TNS Service Name** box, enter the name of the TNS service.
2. In the **User Name** box, enter the name you use to access an Oracle Database.
3. In the **Password** box, enter the password you use to access an Oracle Database.
4. Click **OK**.

An application must connect to a data source to access the data in it. Different applications connect to data sources at different times. For example, an application might connect to a data source only at your request, or it might connect automatically when it starts. For information about when an application connects to a data source, see the documentation for that application.

For additional information, click any of these links:

- For all users:
  - Configuring the Data Source
- For programmers:
  - SQLDriverConnect Implementation
  - Data Source Configuration Options

3.2 Troubleshooting

Topics:

3.2.1 About Using Oracle Database ODBC Driver for the First Time

Describes useful information about using the Oracle Database ODBC driver for the first time.

See the Oracle Database ODBC driver developer home: ODBC Developer Center, where you can find additional information about the Oracle Database ODBC driver features, resources, such as where to find Oracle Instant Client ODBC installation information, the Oracle Instant Client ODBC download site, the Oracle ODBC discussion forum, and information about some related technologies.

3.2.2 Expired Password

This section contains information about expired passwords.

Expired Password Behavior

If you try to connect to the database and your password has expired, you are prompted to change your password. Upon making a successful password change, you are connected to the database. However, if you try to connect to the database using a `SQLDriverConnect` call with a `SQL_DRIVER_NOPROMPT` parameter value, the Oracle Database ODBC driver does not prompt you for the password change. Instead, an error condition results, producing an error message and number that indicates that the password has expired.

4
Using the Oracle Database ODBC Driver

This chapter is intended to provide the Oracle Database ODBC driver users with information about configuring and using the Oracle Database ODBC driver.

Topics:

4.1 Connecting to Oracle Database Using TLS (Preconfigured for Azure AD)

This example demonstrates how to connect to Oracle Database from Microsoft Excel with the TLS preconfigured to use Azure AD.

4.1.1 Overview

You can use your Microsoft Entra ID (was Azure AD) SSO credentials to access an Oracle Database from Microsoft Excel and other tools, when using the Oracle Database ODBC driver. The following example shows you how you can access an Oracle Database from Microsoft Excel.

4.1.2 Prerequisite Steps to Using Oracle ODBC with Excel

> **Note:**
>
> Only Oracle Database 19.16, and higher, and Oracle Database 23ai (not including Oracle Database 21c) support the Entra ID integration.

- Configure the database for Entra ID integration.

  > **See Also:**
  >
  > Authenticating and Authorizing Microsoft Entra ID (Azure AD) Users for Oracle Databases in *Oracle Database Security Guide* for more information about Entra ID integration with Oracle Database.

- Get a valid OAuth 2 token for your database before you start your configuration and put the token into the location specified by the `TOKEN_LOCATION` parameter in your connect string.

  > **Note:**
  >
  > The token is only valid for about an hour. You may need to request a new token if the time taken to complete the configurations exceeds an hour.

  > **See Also:**
  >
  > Local Naming Parameters in the tnsnames.ora File in *Oracle Database Net Services Reference* for more information about the `TOKEN_LOCATION` parameter.

4.1.3 Installing the ODBC Driver

Download and install the appropriate version (19 or 23) of the 32 or 64-bit ODBC driver. Follow these steps:

1. Navigate to https://www.oracle.com/database/technologies/instant-client/microsoft-windows-32-downloads.html or https://www.oracle.com/database/technologies/instant-client/winx64-64-downloads.html
2. Download the Instant Client Basic Package.
3. Download the ODBC package.
4. Unzip the Instant Client folder (`instantclient_xx_yy`) to the desired location. For example: `C:\Oracle\SQLPlus\`.
5. Unzip the ODBC package and put the contents of the zip file: `instantclient_xx_yy` into the Instant Client folder (files include `odbc_install.exe`, `odbc_license`, and `odbc_readme`).

6. Run `odbc_install`.

   This registers the ODBC driver with the ODBC Data Sources GUI.

4.1.4 Configuring tnsnames.ora, TNS_ADMIN, and PATH

1. Set the `TNS_ADMIN` environment variable to point to your `tnsnames.ora` folder.
2. Add the path to your ODBC driver in the `PATH` environment variable.
   - The name of the ODBC driver is `odbcad32.exe` – for both the 32-bit and 64-bit drivers.
   - Paradoxically, on a 64-bit Windows Operating System (OS), the 32-bit `odbcad32.exe` is installed in `C:\Windows\sysWOW64`, and the 64-bit `odbcad32.exe` is installed in `C:\Windows\system32`.
   - If you are on a 64-bit Windows OS, put `C:\Windows\system32` near the beginning of the path – and definitely in front of `sysWOW64` to make sure it sees the 64-bit version first.
3. Add the connect string for the Oracle Database configured for the Entra ID in `tnsnames.ora`.

   The following is an example:

```
azuredb = (description= (retry_count=20)(retry_delay=3)
(address=(protocol=tcps)(port=1521)
(host=adb.us-ashburn-1.oraclecloud.com))
(connect_data=(service_name=xxx123_azuredb_high.adb.oraclecloud.com))
(security=(ssl_server_dn_match=yes)(TOKEN_AUTH=OAUTH)
(TOKEN_LOCATION="C:\USERS\PETERFI\Oracle\azuredb\token")))
```

> **See Also:**
>
> Authenticating and Authorizing Microsoft Entra ID (Azure AD) Users for Oracle Databases in *Oracle Database Security Guide* for more information about Entra ID integration with Oracle Database.

4.1.5 Getting an OAuth 2 Token

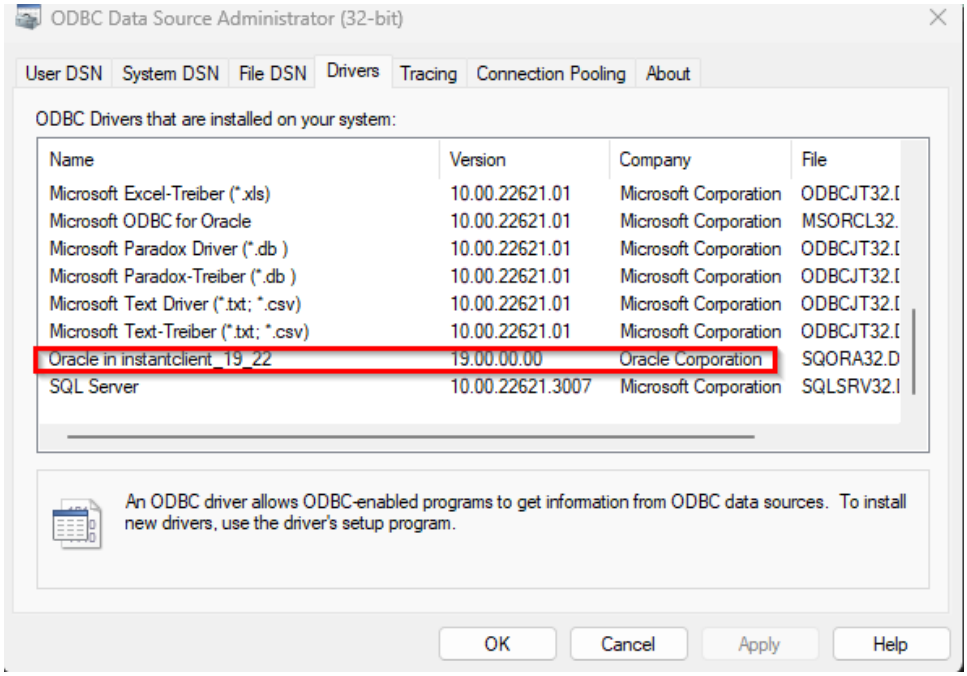
- You need to get an OAuth 2 token (if not already done) at this point because you need the token to test the new Data Source Name (DSN) that you will create in the ODBC Data Source Administrator GUI in the next section.

4.1.6 Configuring DSN

1. Open ODBC Data Source Administrator for the correct bit (32-bit or 64-bit).

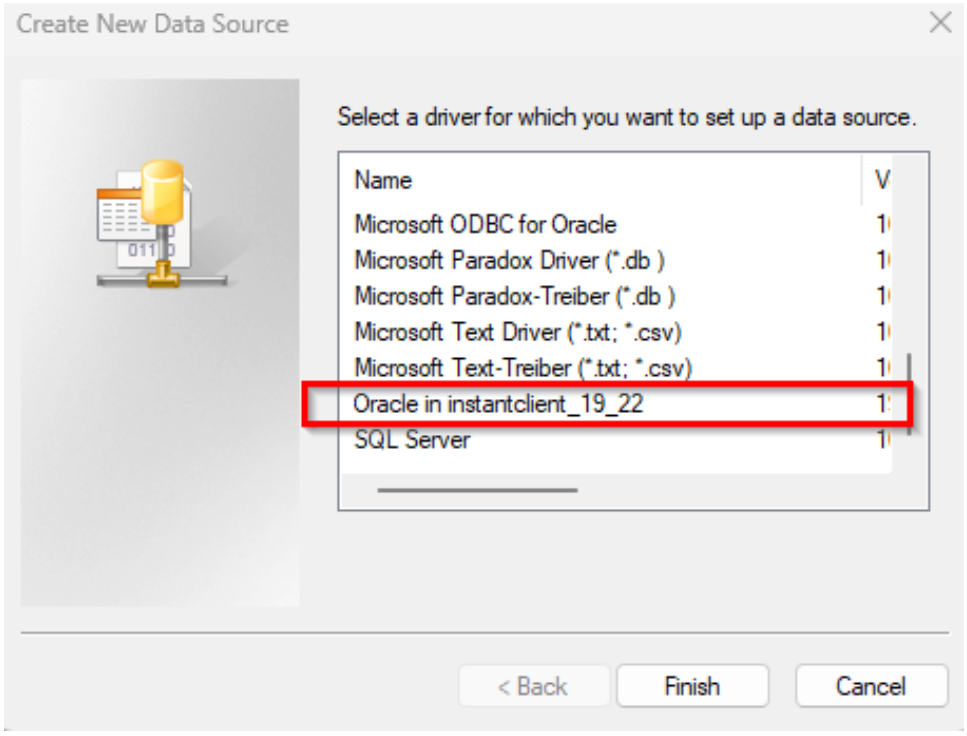
   You can open ODBC Data Source Administrator from the command line or you can use search or open.
2. Ensure that the **Drivers** tab displays the Oracle Database ODBC driver that you just installed.

Figure 4-1 ODBC Data Source Administrator (32-bit)



3. Open the **User DSN** tab.
4. Click **Add**.
5. In the Create New Data Source dialog box, select the appropriate Oracle Database ODBC driver.

Figure 4-2 Create New Data Source

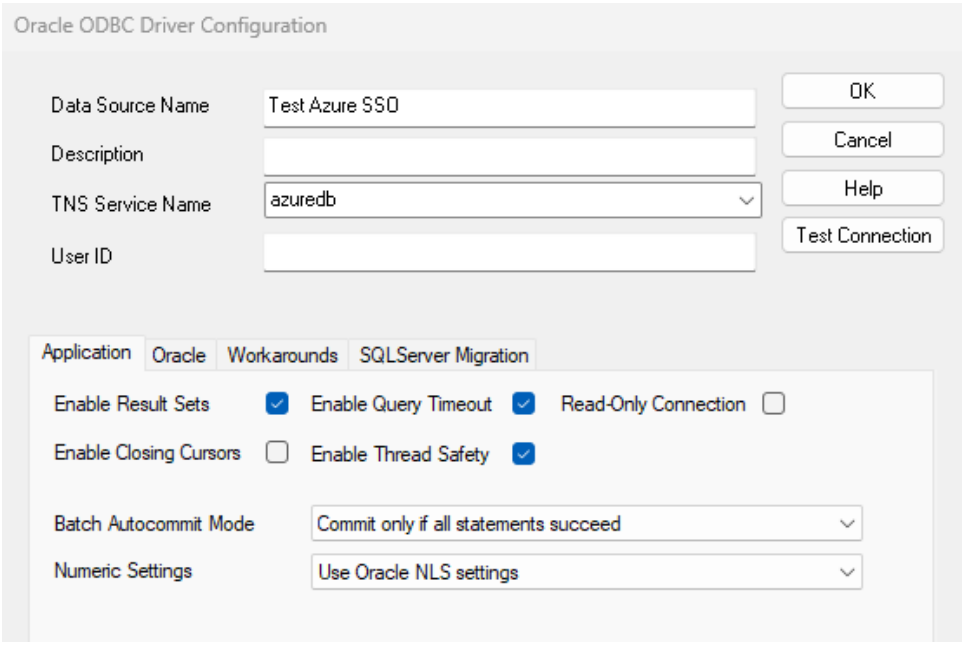


6. Select **Finish**.

   The Oracle ODBC Driver Configuration dialog box is displayed.

7. In the Oracle ODBC Driver Configuration dialog box, do the following:

Figure 4-3 Oracle ODBC Driver Configuration



   a. In **Data Source Name**, fill in a data source name.

   b. In **TNS Service Name**, select the database for the TNS service name.

      The selected name should reflect what is in the `tnsnames.ora` file.

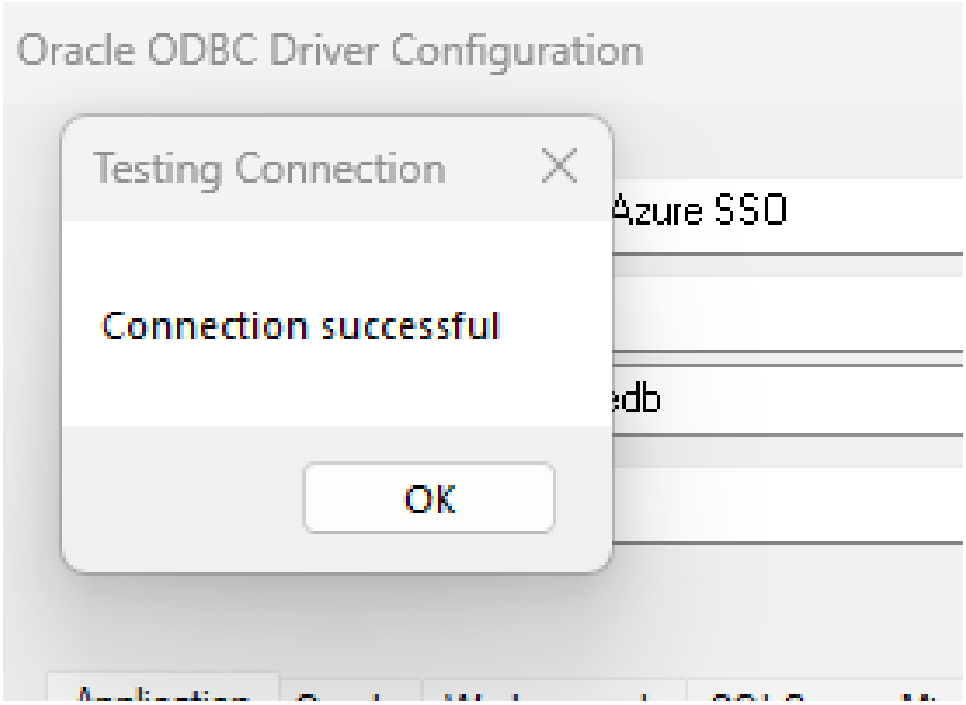
   c. Leave **User ID** blank.

   d. Click the **Test Connection** button.

      You should see a connection successful message (make sure you still have a valid token).

      The token that you put into `TOKEN_LOCATION` is sent to the database. If you are authorized to access the database, then you get a connection successful message as follows:

Figure 4-4 Oracle ODBC Driver Configuration - Connection Successful Message

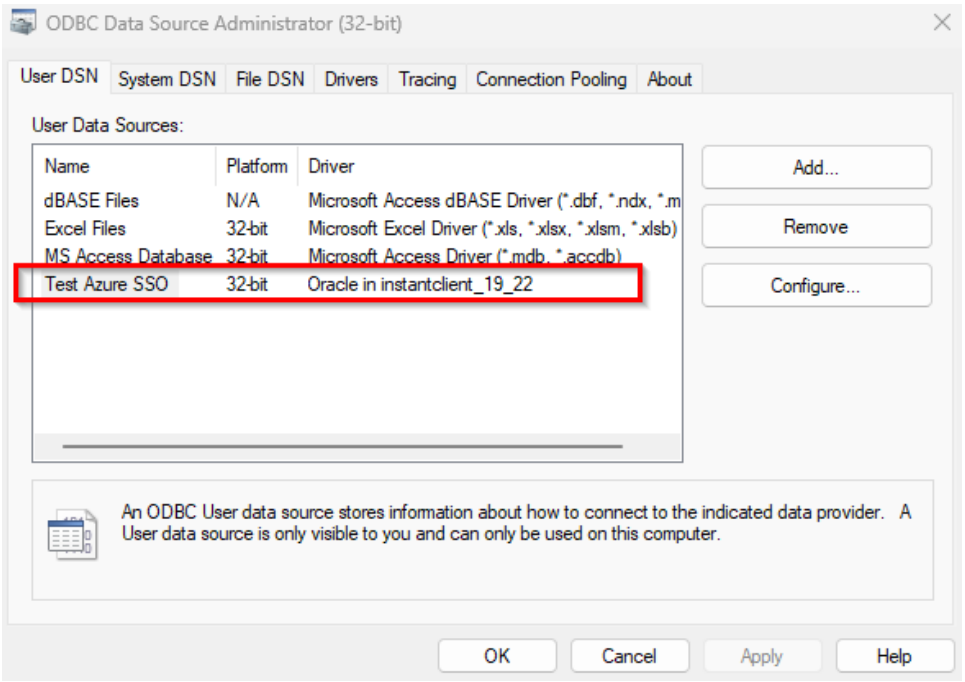


e. Click **OK** to close the message.

8. In the Oracle ODBC Driver Configuration dialog box, click **OK**.

   In the ODBC Data Source Administrator dialog box, in the **User Data Sources** list, you can see your new user data source displayed.

Figure 4-5 ODBC Data Source Administrator (32-bit)

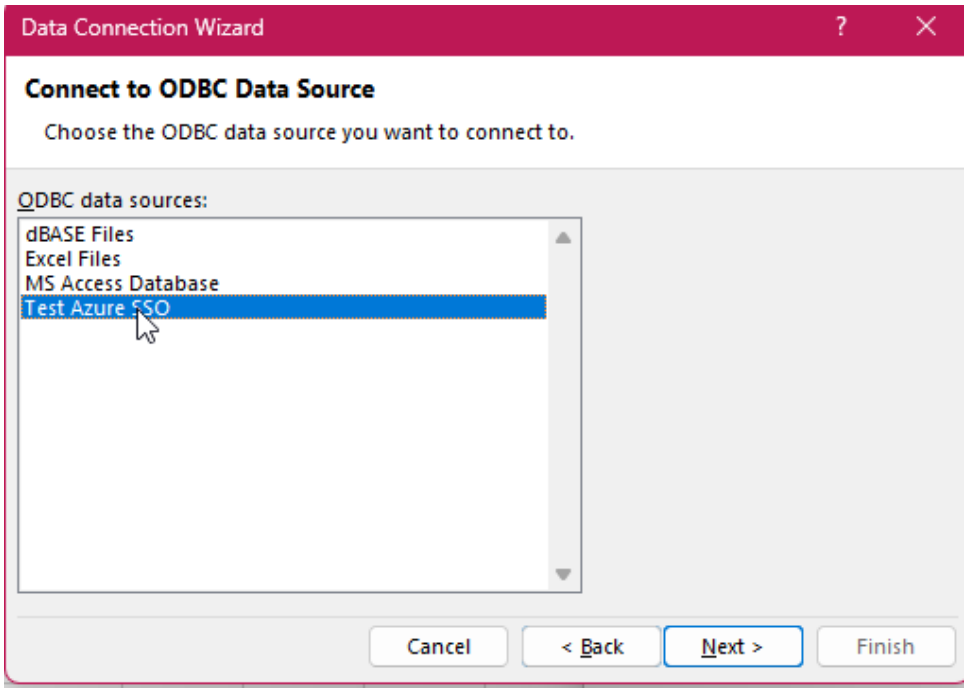


4.1.7 Configuring Excel

1. Open a new Excel sheet.
2. Select the **Data** tab.
3. On the **Data** tab, in the **Get & Transform Data** section, click **Existing Connections**.
4. On the Existing Connections dialog box, click **Browse for More**.
5. On the Select Data Source dialog box, click **New Source**.
6. On the Data Connection Wizard dialog box, select **ODBC DSN** from the data source type list, and click **Next**.

   You should see your new DSN.
7. On the Data Connection Wizard - Connect to ODBC Data Source dialog box, select the new DSN and click **Next**.

Figure 4-6 Data Connection Wizard - Connect to ODBC Data Source

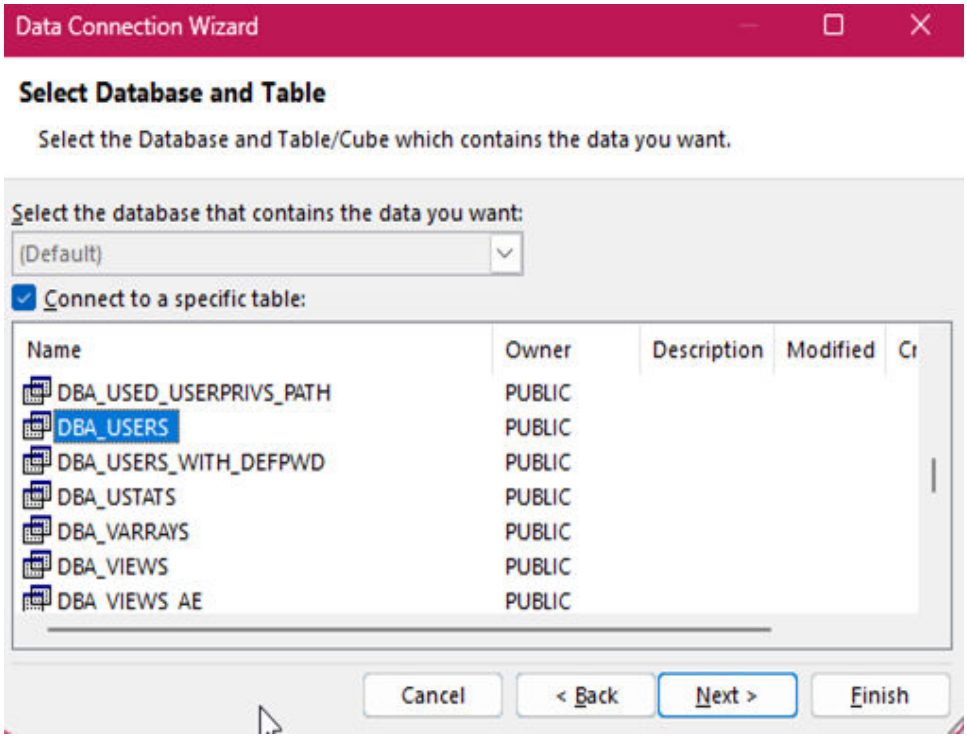


When you click **Next**, Excel accesses the database using your token.

> **Note:**
>
> Ensure that your token is still valid.

The Data Connection Wizard - Select Database and Table dialog box is displayed.

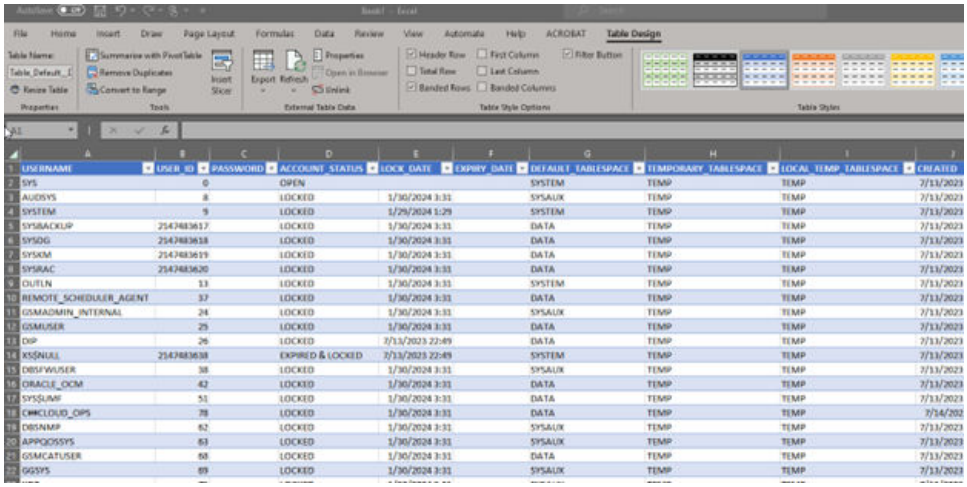
Figure 4-7 Data Connection Wizard - Select Database and Table



8. In the Data Connection Wizard - Select Database and Table dialog box, select the database that contains the data you want to import to Excel, and click **Next**.
9. Fill out the next form with file name and description, and other details as needed, and click **Finish**.
10. From the data import form, select the required data to import.

    You can now see the data in the Excel sheet.

Figure 4-8 Excel Sheet with Imported Data



This concludes the setup of the Oracle Database ODBC driver in Microsoft Excel.

4.2 Creating Oracle Database ODBC Driver TNS Service Names

To create the Oracle Database ODBC driver TNS Service Names with Oracle Net Services, use the Oracle Net Configuration Assistant (NETCA), which is installed when you install Oracle Net Services. NETCA creates the Oracle Database ODBC driver TNS Service Name entries in the `tnsnames.ora` file.

4.3 SQL Statements

The Oracle Database ODBC Driver is broadly compatible with the SQL-99 Core specification, which is a superset of the SQL-92 Entry Level specification. In addition to Oracle's grammar, the vendor-specific escape sequences outlined in Appendix C of the ODBC specifications are also supported. In accordance with the design of ODBC, the Oracle Database ODBC driver passes the native SQL syntax to Oracle Database.

See Also:

- Data Types
- Implementation of the ODBC SQL Syntax for programmers

4.4 Data Types

The Oracle Database ODBC driver maps the Oracle Database data types to the ODBC SQL data types.

Note:

All conversions in Appendix D of the *Microsoft ODBC 3.52 Software Development Kit and Programmer's Reference* are supported for the ODBC SQL data types listed from a call to `SQLGetInfo` with the appropriate information type.

See Also:

- – Implementation of Data Types (Advanced)
  - – SQL Statements
- For programmers:
  - – Implementation of Data Types (Programming)

4.5 Implementation of Data Types (Advanced)

Topics:

BOOLEAN Data Types

Starting Oracle Database 23ai, Oracle Database supports the native `BOOLEAN` data type in compliance with the ISO SQL standard. The native boolean type enables you to define a table column as a SQL boolean data type with the value as true, false, or null.

Using the SQL boolean data type provides clarity, consistency, and speed to coding. With the boolean data type, you can represent a boolean state more clearly, and improve the readability of the code.

Using the native boolean data types support, ODBC-compliant applications can:

- Fetch or modify `BOOLEAN` column data
- Fetch metadata about a `BOOLEAN` column

The boolean data type is represented externally as the `SQLT_BOL` data type. The `SQLT_BOL` data type is used as the SQL type identifier for `BOOLEAN` columns. Bind and define API calls enable the `SQLT_BOL` data type to be associated with host variables in ODBC-based applications.

The ODBC interface represents boolean type with `SQL_C_BIT`, which is the C data type identifier. `SQL_C_BIT` is an unsigned char (`UCHAR`) that represents boolean type in applications. `SQL_C_BIT` only takes a 0 or 1 value, and so, when retrieving boolean data from the database, the data value is represented as 0 or 1.

To bind and fetch (or modify) boolean type data with `BOOLEAN` columns, you can have an application call the bind and define functions, and specify the C data type: `SQL_C_BIT` with the:

- `TargetType` argument in the `SQLBindCol()` and `SQLGetData()` functions.
- `ValueType` argument in the `SQLBindParameter()` function.

The `SQLBindCol()` function binds the `BOOLEAN` column to an application variable before the fetch and the `SQLGetData()` function binds the fetched data to variables after the fetch. The `SQLBindParameter()` function binds parameters in an SQL statement to application variables.

If the `TargetType` argument is a `SQL_C_BIT` data type, the Oracle Database ODBC driver maps `SQLT_BOL` to `SQL_C_BIT` while processing the bind and define parameters. The driver then performs the necessary conversions when fetching (or modifying) and retrieving data from the `BOOLEAN` columns.

To determine if a data source supports boolean data type, you can have an application call the `SQLGetTypeInfo` function.

To retrieve metadata for table columns that are externally defined with the `SQLT_BOL` data type, you can have an application call the `SQLDescribeCol()` function.

For backward compatibility, Oracle Database releases prior to 23ai use internal data type conversions to support boolean values in the Oracle Database ODBC driver.

See Also:

- Boolean Data Type in *Oracle Database SQL Language Reference* for more information about boolean data types
- Microsoft ODBC API specifications for more information about the ODBC bind and define functions and supported data types

DATE and TIMESTAMP

The semantics of Oracle `DATE` and `TIMESTAMP` data types do not correspond exactly with the ODBC data types with the same names. The Oracle `DATE` data type contains both date and time information while the `SQL_DATE` data type contains only date information. The Oracle `TIMESTAMP` data type also contains date and time information, but it has greater precision in fractional seconds. The ODBC driver reports the data types of both Oracle `DATE` and `TIMESTAMP` columns as `SQL_TIMESTAMP` to prevent information loss. Similarly, the ODBC driver binds `SQL_TIMESTAMP` parameters as Oracle `TIMESTAMP` values.

Floating Point Data Types

When connected to a 10.1 or later Oracle server, the ODBC driver maps the Oracle floating point data types `BINARY_FLOAT` and `BINARY_DOUBLE` to the ODBC data types `SQL_REAL` and `SQL_DOUBLE`, respectively. In previous releases, `SQL_REAL` and `SQL_DOUBLE` are mapped to the generic Oracle numeric data type.

See Also:

DATE and TIMESTAMP Data Types

VECTOR Data Type

Starting Oracle Database 23ai, the Oracle Database ODBC driver supports `VECTOR` data type. The driver uses `SQL_CHAR` to map to the `VECTOR` data type.

See Also:

Vector Data Type in *Oracle Database SQL Language Reference* for more information about vector data types

4.6 Error Messages

When an error occurs, the Oracle Database ODBC driver returns the native error number, the `SQLSTATE` (an ODBC error code), and an error message. The driver derives this information both from errors detected by the driver and errors returned by the Oracle server.

Native Error

For errors that occur in the data source, the Oracle Database ODBC driver returns the native error returned to it by the Oracle server. When the Oracle Database ODBC driver or the Driver Manager detects an error, the Oracle Database ODBC driver returns a native error of zero.

SQLSTATE

For errors that occur in the data source, the Oracle Database ODBC driver maps the returned native error to the appropriate `SQLSTATE`. When the Oracle Database ODBC driver detects an error, it generates the appropriate `SQLSTATE`. When the Driver Manager detects an error, it generates the appropriate `SQLSTATE`.

Error Message

For errors that occur in the data source, the Oracle Database ODBC driver returns an error message based on the message returned by the Oracle server. For errors that occur in the Oracle Database ODBC driver or the Driver Manager, the Oracle Database ODBC driver returns an error message based on the text associated with the `SQLSTATE`.

Error messages have the following format:

```
[vendor] [ODBC-component] [data-source] error-message
```

The prefixes in brackets ([ ]) identify the source of the error. The following table shows the values of these prefixes returned by the Oracle Database ODBC driver. When the error occurs in the data source, the [vendor] and [ODBC-component] prefixes identify the vendor and name of the ODBC component that received the error from the data source.

Table 4-1 Error Message Values of Prefixes Returned by the Oracle Database ODBC Driver

Error Source	Prefix	Value
Driver Manager	[vendor][ODBC-component][data-source]	[Microsoft/unixODBC][ODBC Driver Manager]N/A
Oracle ODBC Driver	[vendor][ODBC-component][data-source]	[ORACLE][ODBC Driver]N/A
Oracle server	[vendor][ODBC-component][data-source]	[ORACLE][ODBC Driver]N/A

For example, if the error message does not contain the [Ora] prefix shown in the following format, the error is an Oracle ODBC Driver error and should be self-explanatory.

```
[Oracle][ODBC]Error message text here
```

If the error message contains the [Ora] prefix shown in the following format, it is not an Oracle ODBC Driver error.

> **Note:**
>
> Although the error message contains the [Ora] prefix, the actual error may be originating from one of several sources.

```
[Oracle][ODBC][Ora]Error message text here
```

If the error message text starts with the following prefix, you can obtain more information about the error in the Oracle server documentation.

```
ORA-
```

Oracle Net Services errors and Trace logging are located under the *ORACLE_HOME*\NETWORK directory on Windows systems or the *ORACLE_HOME*/NETWORK directory on UNIX systems where the OCI software is installed and specifically in the log and trace directories respectively. Database logging is located under the *ORACLE_HOME*\RDBMS directory on Windows systems or the *ORACLE_HOME*/rdbms directory on UNIX systems where the Oracle server software is installed.

See the Oracle server documentation for more information about server error messages.

5

Oracle Database ODBC Driver for Programmers

The chapter is intended for programmers who want to develop robust ODBC applications using the Oracle Database ODBC driver.

Topics:

5.1 Format of the Connection String

The following table describes keywords that can be included in the connection string argument of the `SQLDriverConnect` function call. Missing keywords are read from the Administrator entry for the data source. Values specified in the connection string override those contained in the Administrator entry. See the *Microsoft ODBC 3.52 Software Development Kit and Programmer's Reference* for more information about the `SQLDriverConnect` function.

Table 5-1 Keywords that Can Be Included in the Connection String Argument of the SQLDriverConnect Function Call

Keyword	Meaning	Comments
DSN	ODBC Data Source Name	User-supplied name.
DBQ	TNS Service Name	User-supplied name.
UID	User ID or User Name	User-supplied name.
PWD	Password	User-supplied password. Specify PWD=; for an empty password.

Table 5-1 (Cont.) Keywords that Can Be Included in the Connection String Argument of the SQLDriverConnect Function Call

Keyword	Meaning	Comments
DBA	Database Attribute	W=write access. R=read-only access.
APA	Applications Attributes	T=Thread Safety Enabled. F=Thread Safety Disabled.
RST	Result Sets	T=Result Sets Enabled. F=Result Sets Disabled.
QTO	Query Timeout Option	T=Query Timeout Enabled. F=Query Timeout Disabled.
CSR	Close Cursor	T=Close Cursor Enabled. F=Close Cursor Disabled.
BNF	Bind `NUMBER` as `FLOAT`	T=Bind `NUMBER` as `FLOAT`. F=Bind `NUMBER` as `NUMBER`.
DRH	Disable Rule Hint	T=Disable Rule Hint. F=Enable Rule Hint.
BAM	Batch Autocommit Mode	IfAllSuccessful=Commit only if all statements are successful (old behavior). UpToFirstFailure=Commit up to first failing statement (V7 ODBC behavior). AllSuccessful=Commit all successful statements (only when connected to an Oracle database; against other databases, same behavior as V7).
FBS	Fetch Buffer Size	User-supplied numeric value (specify a value in bytes of 0 or greater). The default is 60,000 bytes.
FEN	Failover	T=Failover Enabled. F=Failover Disabled.
FRC	Failover Retry Count	User-supplied numeric value. The default is 10.
FDL	Failover Delay	User-supplied numeric value. The default is 10.
LOB	LOB Writes	T=LOBs Enabled. F=LOBs Disabled.
MTS	Microsoft Transaction Server Support	T=Disabled. F=Enabled.
FWC	Force `SQL_WCHAR` Support	T=Force `SQL_WCHAR` Enabled. F=Force `SQL_WCHAR` Disabled.
EXC	EXEC Syntax	T=EXEC Syntax Enabled. F=EXEC Syntax Disabled.
XSM	Schema Field	Default=Default. Database=Database Name. Owner=Owner Name.

Table 5-1 (Cont.) Keywords that Can Be Included in the Connection String Argument of the SQLDriverConnect Function Call

Keyword	Meaning	Comments
MDI	Set Metadata ID Default	T=`SQL_ATTR_METADATA_ID` defaults to `SQL_TRUE`. F=`SQL_ATTR_METADATA_ID` defaults to `SQL_FALSE`.
DPM	Disable `SQLDescribeParam`	T=`SQLDescribeParam` Disabled. F=`SQLDescribeParam` Enabled.
BTD	Bind `TIMESTAMP` as `DATE`	T=Bind `SQL_TIMESTAMP` as Oracle `DATE` F=Bind `SQL_TIMESTAMP` as Oracle `TIMESTAMP`
NUM	Numeric Settings	NLS=Use Oracle NLS numeric settings (to determine the decimal and group separator). MS=Use Microsoft regional settings. US=Use US settings.
ODA	Use `OCIDescribeAny( )`	T= Use `OCIDescribeAny()` call to gain performance improvement when application makes heavy calls to small packaged procedures that return `REF CURSORS`. F= Do not use `OCIDescribeAny()`. By default, use `OCIDescribeAny()` value is `FALSE`.
STE	SQL Translate ORA Errors Specifies whether the Oracle Database ODBC driver is to translate the Oracle error codes	T=Translate ORA errors. F=Do not translate any ORA error. By default, `SQLTranslateErrors` is `FALSE`.
TSZ	Token Size	User-supplied numeric value. Sets the token size to the nearest multiple of 1 KB (1024 bytes) beginning at 4 KB (4096 bytes). The default size is 8 KB (8192 bytes). The maximum value that can be set is 128 KB (131068 bytes).

If the following keyword is specified in the connection string, the Oracle Database ODBC driver does not read values defined from the Administrator:

```
DRIVER={Oracle ODBC Driver}
```

Examples of valid connection strings are:

```
1) DSN=Personnel;UID=Kotzwinkle;PWD=;2) DRIVER={Oracle ODBC
Driver};UID=Kotzwinkle;PWD=whatever;DBQ=instl_alias;DBA=W;
```

See Also:

- Connecting to an Oracle Data Source for all users
- SQLDriverConnect Implementation for programmers

5.2 SQLDriverConnect Implementation

The following table describes the keywords required by the `SQLDriverConnect` connection string.

Table 5-2 Keywords Required by the SQLDriverConnect Connection String

Keyword	Description
DSN	The name of the data source.
DBQ	The TNS Service Name. See Creating Oracle Database ODBC Driver TNS Service Names. For more information, see the Oracle Net Services documentation.
UID	The user login ID or user name.
PWD	The user-specified password.

5.3 Reducing Lock Timeout in a Program

The Oracle server waits indefinitely for lock conflicts between transactions, to be resolved. You can limit the amount of time that the Oracle server waits for locks to be resolved by calling the ODBC `SQLSetConnectAttr` function before connecting to the data source.

Specify a non-zero value for the `SQL_ATTR_QUERY_TIMEOUT` attribute in the ODBC `SQLSetStmtAttr` function. If you specify a lock timeout value using the ODBC `SQLSetConnectAttr` function, it overrides any value specified in the `oraodbc.ini` file.

> **See Also:**
>
> Reducing Lock Timeout for more information about specifying a value in the `oraodbc.ini` file

5.4 Linking with odbc32.lib (Windows) or libodbc.so (UNIX)

For Windows platforms, when you link your program, you must link it with the import library `odbc32.lib`.

For UNIX platforms, an ODBC application must be linked to `libodbc.so`.

5.5 Information about ROWID

The ODBC `SQLSpecialColumns` function returns information about the columns in a table. When used with the Oracle Database ODBC driver, it returns information about the Oracle `ROWID`s associated with an Oracle table.

5.6 ROWID in a WHERE Clause

ROWIDs can be used in the WHERE clause of an SQL statement. However, the ROWID value must be presented in a parameter marker.

5.7 Enabling Result Sets

Oracle reference cursors (Result Sets) allow an application to retrieve data using stored procedures and stored functions. The following information identifies how to use reference cursors to enable Result Sets through ODBC.

- The ODBC syntax for calling stored procedures must be used. Native PL/SQL is not supported through ODBC. The following identifies how to call the procedure or function without a package and within a package. The package name, in this case, is RSET.

  ```
  Procedure call:
  {CALL Example1(?)}
  {CALL RSET.Example1(?)}
  Function Call:
  {? = CALL Example1(?)}
  {? = CALL RSET.Example1(?)}
  ```

- The PL/SQL reference cursor parameters are omitted when calling the procedure. For example, assume that the procedure Example2 is defined to have four parameters. Parameters 1 and 3 are reference cursor parameters and parameters 2 and 4 are character strings. The call is specified as:

  ```
  {CALL RSET.Example2("Literal 1", "Literal 2")}
  ```

The following example application shows how to return a Result Set using the Oracle Database ODBC driver:

```
/*
 * Sample Application using Oracle reference cursors via ODBC
 *
 * Assumptions:
 *
 * 1) Oracle Sample database is present with data loaded for the EMP table.
 * 2) Two fields are referenced from the EMP table ename and mgr.
 * 3) A data source has been setup to access the sample database.
 *
 * Program Description:
 *
 * Abstract:
 *
 * This program demonstrates how to return result sets using
 * Oracle stored procedures
 *
 * Details:
 *
 * This program:
 * Creates an ODBC connection to the database.
 * Creates a Packaged Procedure containing two result sets.
 * Executes the procedure and retrieves the data from both result sets.
 * Displays the data to the user.
 * Deletes the package then logs the user out of the database.
 *
 *
 * The following is the actual PL/SQL this code generates to
```

```
 * create the stored procedures.
 *
 * DROP PACKAGE ODBCRefCur;
 *
 * CREATE PACKAGE ODBCRefCur AS
 * TYPE ename_cur IS REF CURSOR;
 * TYPE mgr_cur IS REF CURSOR;
 * PROCEDURE EmpCurs(Ename IN OUT ename_cur, Mgr IN OUT mgr_cur, pjob IN VARCHAR2);
 * END;
 *
 * CREATE or REPLACE PACKAGE BODY ODBCRefCur AS
 * PROCEDURE EmpCurs(Ename IN OUT ename_cur, Mgr IN OUT mgr_cur, pjob IN VARCHAR2)
 * AS
 * BEGIN
 * IF NOT Ename%ISOPEN
 * THEN
 * OPEN Ename for SELECT ename from emp;
 * END IF;
 * IF NOT Mgr%ISOPEN
 * THEN
 * OPEN Mgr for SELECT mgr from emp where job = pjob;
 * END IF;
 * END;
 * END;
 *
 */

/* Include Files */
#ifdef WIN32
  #include <windows.h>
#endif
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

/* Defines */
#define JOB_LEN 9
#define DATA_LEN 100
#define SQL_STMT_LEN 500

/* Procedures */
void DisplayError(SWORD HandleType, SQLHANDLE hHandle, char *Module);

/* Main Program */
int main()
{
  SQLHENV hEnv;
  SQLHDBC hDbc;
  SQLHSTMT hStmt;
  SQLRETURN rc;
  char *DefUserName ="scott";
  char *DefPassWord ="tiger";
  SQLCHAR ServerName[DATA_LEN];
  SQLCHAR *pServerName=ServerName;
  SQLCHAR UserName[DATA_LEN];
  SQLCHAR *pUserName=UserName;
  SQLCHAR PassWord[DATA_LEN];
  SQLCHAR *pPassWord=PassWord;
  char Data[DATA_LEN];
  SQLINTEGER DataLen;
  char error[DATA_LEN];
  char *charptr;
```

```
SQLCHAR SqlStmt[SQL_STMT_LEN];
SQLCHAR *pSqlStmt=SqlStmt;
char *pSalesMan = "SALESMAN";
SQLINTEGER sqlnts=SQL_NTS;

/* Allocate the Environment Handle */
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &hEnv );
if (rc != SQL_SUCCESS)
{
  printf( "Cannot Allocate Environment Handle/n");
  printf( "/nHit Return to Exit/n");
  charptr = gets ((char *)error);
  exit(1);
}

/* Set the ODBC Version */
rc = SQLSetEnvAttr(hEnv, SQL_ATTR_ODBC_VERSION, (void *)SQL_OV_ODBC3, 0);
if (rc != SQL_SUCCESS)
{
  printf("Cannot Set ODBC Version/n");
  printf("/nHit Return to Exit/n");
  charptr = gets((char *)error);
  exit(1);
}


/* Allocate the Connection handle */
rc = SQLAllocHandle(SQL_HANDLE_DBC, hEnv, &hDbc);
if (rc != SQL_SUCCESS)
{
  printf("Cannot Allocate Connection Handle/n");
  printf("/nHit Return to Exit/n");
  charptr = gets((char*) error);
  exit(1);
}

/* Get User Information */
lstrcpy((char*) pUserName, DefUserName);
lstrcpy((char*) pPassWord, DefPassWord);

/* Data Source name */
printf( "/nEnter the ODBC Data Source Name/n" );
charptr = gets((char*) ServerName);

/* User Name */
printf("/nEnter User Name Default [%s]/n", pUserName);
charptr = gets((char*) UserName);
if (*charptr == '/0')
{
  lstrcpy((char*) pUserName, (char*) DefUserName);
}

/* Password */
printf ("/nEnter Password Default [%s]/n", pPassWord);
charptr = gets((char*) PassWord);
if (*charptr == '/0')
{
  lstrcpy((char*) pPassWord, (char*) DefPassWord);
}

/* Connection to the database */
rc = SQLConnect(hDbc, pServerName, (SQLSMALLINT) lstrlen((char *)pServerName),
```

```
pUserName,
                  (SQLSMALLINT) lstrlen((char*)pUserName), pPassWord,
                  (SQLSMALLINT) lstrlen((char *)pPassWord));
  if (rc != SQL_SUCCESS)
  {
    DisplayError(SQL_HANDLE_DBC, hDbc, "SQLConnect");
  }

  /* Allocate a Statement */
  rc = SQLAllocHandle(SQL_HANDLE_STMT, hDbc, &hStmt);
  if (rc != SQL_SUCCESS)
  {
    printf( "Cannot Allocate Statement Handle/n");
    printf( "/nHit Return to Exit/n");
    charptr = gets((char *)error);
    exit(1);
  }

  /* Drop the Package */
  lstrcpy((char *) pSqlStmt, "DROP PACKAGE ODBCRefCur");
  rc = SQLExecDirect(hStmt, pSqlStmt, lstrlen((char *)pSqlStmt));

  /* Create the Package Header */
  lstrcpy((char *) pSqlStmt, "CREATE PACKAGE ODBCRefCur AS/n" );
  lstrcat((char *) pSqlStmt, " TYPE ename_cur IS REF CURSOR;/n" );
  lstrcat((char *) pSqlStmt, " TYPE mgr_cur IS REF CURSOR;/n" );
  lstrcat((char *) pSqlStmt, " PROCEDURE EmpCurs (Ename IN OUT ename_cur," );
  lstrcat((char *) pSqlStmt, " Mgr IN OUT mgr_cur,pjob IN VARCHAR2);/n/n");
  lstrcat((char *) pSqlStmt, "END;/n" );

  rc = SQLExecDirect(hStmt, pSqlStmt, lstrlen((char *)pSqlStmt));
  if (rc != SQL_SUCCESS)
  {
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLExecDirect");
  }

  /* Create the Package Body */
  lstrcpy((char *) pSqlStmt, "CREATE PACKAGE BODY ODBCRefCur AS/n" );
  lstrcat((char *) pSqlStmt, " PROCEDURE EmpCurs (Ename IN OUT ename_cur," );
  lstrcat((char *) pSqlStmt, " Mgr IN OUT mgr_cur, pjob IN VARCHAR2)/n" );
  lstrcat((char *) pSqlStmt, " AS/n" );
  lstrcat((char *) pSqlStmt, " BEGIN/n" );
  lstrcat((char *) pSqlStmt, " IF NOT Ename%ISOPEN/n" );
  lstrcat((char *) pSqlStmt, " THEN/n" );
  lstrcat((char *) pSqlStmt, " OPEN Ename for SELECT ename from emp;/n" );
  lstrcat((char *) pSqlStmt, " END IF;/n/n" );
  lstrcat((char *) pSqlStmt, " IF NOT Mgr%ISOPEN/n THEN/n" );
  lstrcat((char *) pSqlStmt, " OPEN Mgr for SELECT mgr from emp where job = pjob;/n");
  lstrcat((char *) pSqlStmt, " END IF;/n" );
  lstrcat((char *) pSqlStmt, " END;/n" );


  lstrcat((char *) pSqlStmt, "END;/n" );

  rc = SQLExecDirect(hStmt, pSqlStmt, lstrlen((char *)pSqlStmt));
  if(rc != SQL_SUCCESS)
    DisplayError(SQL_HANDLE_STMT, hStmt, "SQLExecDirect");

  /* Bind the Parameter */
  rc = SQLBindParameter(hStmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR, JOB_LEN, 0,
pSalesMan, 0, &sqlnts);
```

```
/* Call the Store Procedure which executes the Result Sets */
lstrcpy( (char *) pSqlStmt, "{CALL ODBCRefCur.EmpCurs(?)}");

rc = SQLExecDirect(hStmt, pSqlStmt, lstrlen((char *)pSqlStmt));
if(rc != SQL_SUCCESS)
  DisplayError(SQL_HANDLE_STMT, hStmt, "SQLExecDirect");

/* Bind the Data */
rc = SQLBindCol(hStmt, 1, SQL_C_CHAR, Data, sizeof(Data), &DataLen);
if(rc != SQL_SUCCESS)
  DisplayError(SQL_HANDLE_STMT, hStmt, "SQLBindCol");

/* Get the data for Result Set 1 */
printf("/nEmployee Names/n/n");

while(rc == SQL_SUCCESS)
{
  rc = SQLFetch(hStmt);
  if(rc == SQL_SUCCESS)
    printf("%s/n", Data);
  else
    if(rc != SQL_NO_DATA)
      DisplayError(SQL_HANDLE_STMT, hStmt, "SQLFetch");
}

printf( "/nFirst Result Set - Hit Return to Continue/n");
charptr = gets ((char *)error);

/* Get the Next Result Set */
rc = SQLMoreResults( hStmt );
if(rc != SQL_SUCCESS)
  DisplayError(SQL_HANDLE_STMT, hStmt, "SQLMoreResults");

/* Get the data for Result Set 2 */
printf("/nManagers/n/n");
while (rc == SQL_SUCCESS)
{
  rc = SQLFetch(hStmt);
  if(rc == SQL_SUCCESS)
    printf("%s/n", Data);
  else
    if (rc != SQL_NO_DATA)
      DisplayError(SQL_HANDLE_STMT, hStmt, "SQLFetch");
}

printf("/nSecond Result Set - Hit Return to Continue/n");
charptr = gets((char *)error);

/* Should Be No More Results Sets */
rc = SQLMoreResults( hStmt );
if (rc != SQL_NO_DATA)
  DisplayError(SQL_HANDLE_STMT, hStmt, "SQLMoreResults");

/* Drop the Package */
lstrcpy((char *)pSqlStmt, "DROP PACKAGE ODBCRefCur");
rc = SQLExecDirect(hStmt, pSqlStmt, lstrlen((char *)pSqlStmt));

/* Free handles close connections to the database */
SQLFreeHandle( SQL_HANDLE_STMT, hStmt );
SQLDisconnect( hDbc );
SQLFreeHandle( SQL_HANDLE_DBC, hDbc );
SQLFreeHandle( SQL_HANDLE_ENV, hEnv );
```

```
  printf( "/nAll Done - Hit Return to Exit/n");
  charptr = gets ((char *)error);
  return(0);
}

/* Display Error Messages */
void DisplayError( SWORD HandleType, SQLHANDLE hHandle, char *Module )
{
  SQLCHAR MessageText[255];
  SQLCHAR SQLState[80];
  SQLRETURN rc=SQL_SUCCESS;
  LONG NativeError;
  SWORD RetLen;
  SQLCHAR error[25];
  char *charptr;

  rc = SQLGetDiagRec(HandleType, hHandle, 1, SQLState, &NativeError, MessageText, 255,
&RetLen);
  printf( "Failure Calling %s/n", Module );
  if (rc == SQL_SUCCESS || rc == SQL_SUCCESS_WITH_INFO)
  {
    printf( "/t/t/t State: %s/n", SQLState);
    printf( "/t/t/t Native Error: %d/n", NativeError );
    printf( "/t/t/t Error Message: %s/n", MessageText );
  }

  printf( "/nHit Return to Exit/n");
  charptr = gets ((char *)error);
  exit(1);
}
```

5.8 Enabling EXEC Syntax

If the syntax of your SQL Server `EXEC` statement can be readily translated to an equivalent Oracle procedure call without change, the Oracle Database ODBC driver can translate it, if you enable this option.

The complete name of a SQL Server procedure consists of up to four identifiers:

- server name
- database name
- owner name
- procedure name

The format for the name is:

```
[[[server.][database].][owner_name].]procedure_name
```

During the migration of the SQL Server database to Oracle, the definition of each SQL Server procedure (or function) is converted to its equivalent Oracle syntax and is defined in a schema in Oracle. Migrated procedures are often reorganized (and created in schemas) in one of these ways:

- All procedures are migrated to one schema (the default option).
- All procedures defined in one SQL Server database are migrated to the schema named with that database name.

- All procedures owned by one user are migrated to the schema named with that user's name.

To support these three ways of organizing migrated procedures, you can specify one of these schema name options for translating procedure names. Object names in the translated Oracle procedure call are not case-sensitive.

5.9 Enabling Event Notification for Connection Failures in an Oracle RAC Environment

If the `SQL_ORCLATTR_FAILOVER_CALLBACK` and `SQL_ORCLATTR_FAILOVER_HANDLE` attributes of the `SQLSetConnectAttr` function are set when a connection failure occurs in an Oracle Real Application Clusters (Oracle RAC) Database environment, event notification is enabled. Both attributes are set using the `SQLSetConnectAttr` function. The symbols for the new attributes are defined in the file `sqora.h`.

The `SQL_ORCLATTR_FAILOVER_CALLBACK` attribute specifies the address of a routine to call when a failure event takes place.

The `SQL_ORCLATTR_FAILOVER_HANDLE` attribute specifies a context handle that is passed as a parameter in the callback routine. This attribute is necessary for the ODBC application to determine which connection the failure event is taking place on.

The function prototype for the callback routine is:

```
void failover_callback(void *handle, SQLINTEGER fo_code)
```

The 'handle' parameter is the value that was set by the `SQL_ORCLATTR_FAILOVER_HANDLE` attribute. Null is returned if the attribute has not been set.

The `fo_code` parameter identifies the failure event that is taking place. The failure events map directly to the events defined in the OCI programming interface. The list of possible events is:

- `ODBC_FO_BEGIN`
- `ODBC_FO_ERROR`
- `ODBC_FO_ABORT`
- `ODBC_FO_REAUTH`
- `ODBC_FO_END`

The following is a sample program that demonstrates, using this feature:

```
/*
  NAME
  ODBCCallbackTest

  DESCRIPTION
  Simple program to demonstrate the connection failover callback feature.

  PUBLIC FUNCTION(S)
  main

  PRIVATE FUNCTION(S)

  NOTES

  Command Line: ODBCCallbackTest filename [odbc-driver]
```

```
*/

#include <windows.h>
#include <tchar.h>
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#include <sql.h>
#include <sqlext.h>
#include "sqora.h"

/*
** Function Prototypes
*/
void display_errors(SQLSMALLINT HandleType, SQLHANDLE Handle);
void failover_callback(void *Handle, SQLINTEGER fo_code);

/*
** Macros
*/
#define ODBC_STS_CHECK(sts) \
  if (sts != SQL_SUCCESS) \
{ \
  display_errors(SQL_HANDLE_ENV, hEnv); \
  display_errors(SQL_HANDLE_DBC, hDbc); \
  display_errors(SQL_HANDLE_STMT, hStmt); \
  return FALSE; \
}

/*
** ODBC Handles
*/
SQLHENV *hEnv = NULL; // ODBC Environment Handle
SQLHANDLE *hDbc = NULL; // ODBC Connection Handle
SQLHANDLE *hStmt = NULL; // ODBC Statement Handle

/*
** Connection Information
*/
TCHAR *dsn = _T("odbctest");
TCHAR *uid = _T("scott");
TCHAR *pwd = _T("tiger");
TCHAR *szSelect = _T("select * from emp");

/*
** MAIN Routine
*/
main(int argc, char **argv)
{
  SQLRETURN rc;

  /*
  ** Allocate handles
  */
  rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, (SQLHANDLE *)&hEnv);
  ODBC_STS_CHECK(rc)

  rc = SQLSetEnvAttr(hEnv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)SQL_OV_ODBC3, 0);
  ODBC_STS_CHECK(rc);

  rc = SQLAllocHandle(SQL_HANDLE_DBC, hEnv, (SQLHANDLE *)&hDbc);
  ODBC_STS_CHECK(rc);
```

```
  /*
  ** Connect to the database
  */
  rc = SQLConnect(hDbc, dsn, (SQLSMALLINT)_tcslen(dsn),
  uid, (SQLSMALLINT)_tcslen(uid),
  pwd, (SQLSMALLINT)_tcslen(pwd));
  ODBC_STS_CHECK(rc);

  /*
  ** Set the connection failover attributes
  */
  rc = SQLSetConnectAttr(hDbc, SQL_ORCLATTR_FAILOVER_CALLBACK, &failover_callback, 0);
  ODBC_STS_CHECK(rc);

  rc = SQLSetConnectAttr(hDbc, SQL_ORCLATTR_FAILOVER_HANDLE, hDbc, 0);
  ODBC_STS_CHECK(rc);

  /*
  ** Allocate the statement handle
  */
  rc = SQLAllocHandle(SQL_HANDLE_STMT, hDbc, (SQLHANDLE *)&hStmt);
  ODBC_STS_CHECK(rc);

  /*
  ** Wait for connection failovers
  */
   while (TRUE)
  {
   Sleep(5000);
   rc = SQLExecDirect(hStmt,szSelect, _tcslen(szSelect));
   ODBC_STS_CHECK(rc);

   rc = SQLFreeStmt(hStmt, SQL_CLOSE);
   ODBC_STS_CHECK(rc);
  }

  /*
  ** Free up the handles and close the connection
  */
  rc = SQLFreeHandle(SQL_HANDLE_STMT, hStmt);
  ODBC_STS_CHECK(rc);

  rc = SQLDisconnect(hDbc);
  ODBC_STS_CHECK(rc);

  rc = SQLFreeHandle(SQL_HANDLE_DBC, hDbc);
  ODBC_STS_CHECK(rc);

  rc = SQLFreeHandle(SQL_HANDLE_ENV, hEnv);
  ODBC_STS_CHECK(rc);

  return TRUE;
}

/*
** Failover Callback Routine
*/
void failover_callback(void *Handle, SQLINTEGER fo_code)
{
  switch(fo_code)
  {
```

```
  case ODBC_FO_BEGIN:
    printf("ODBC_FO_BEGIN recevied\n");
    break;

  case ODBC_FO_ERROR:
    printf("ODBC_FO_ERROR recevied\n");
    break;

  case ODBC_FO_ABORT:
    printf("ODBC_FO_ABORT recevied\n");
    break;

  case ODBC_FO_REAUTH:
    printf("ODBC_FO_REAUTH recevied\n");
    break;

  case ODBC_FO_END:
    printf("ODBC_FO_END recevied\n");
    break;

  default:
    printf("Invalid or unknown ODBC failover code recevied\n");
    break;
  }
  return;

}

/*
** Retrieve the errors associated with the handle passed
** and display them.
*/
void display_errors(SQLSMALLINT HandleType, SQLHANDLE Handle)
{
  SQLTCHAR MessageText[256];
  SQLTCHAR SqlState[5+1];
  SQLSMALLINT i=1;
  SQLINTEGER NativeError;
  SQLSMALLINT TextLength;
  SQLRETURN sts = SQL_SUCCESS;

  if (Handle == NULL) return;

  /* Make sure all SQLState text is null terminated */
  SqlState[5] = '\0';

  /*
  ** Fetch and display all diagnostic records that exist for this handle
  */
  while (sts == SQL_SUCCESS)
  {
    NativeError = 0;
    TextLength = 0;

    sts = SQLGetDiagRec(HandleType, Handle, i, SqlState, &NativeError, (SQLTCHAR
*)&MessageText, sizeof(MessageText), &TextLength);
    if (sts == SQL_SUCCESS)
    {
      printf("[%s]%s\n", SqlState, MessageText);
      if (NativeError != 0)
        printf("Native Error Code: %d\n", NativeError);
      i++;
```

```
    }
  }
  return;
}
```

5.10 Using Implicit Results Feature through ODBC

Use this option when you migrate any third party ODBC application to Oracle Database and you want to use implicit results functionality as supported by the previous vendor. The Oracle Database ODBC driver supports implicit results with stored procedures or an anonymous PL/SQL block. For the current release, implicit results are returned only for `SELECT` statements.

The following code example shows an example ODBC test case using an anonymous SQL script for implicit results.

```
const char *query1="declare \
                  c1 sys_refcursor; \
                  c2 sys_refcursor; \
                  begin \
                  open c1 for select empno,ename from emp where rownum<=3; \
                  dbms_sql.return_result(c1); \
                  open c2 for select empno,ename from emp where rownum<=3; \
                  dbms_sql.return_result(c2); end; ";

int main( )
{
  ...
   ...
 //Allocate all required handles and establish a connection to the database.

 //Prepare and execute the above anonymous PL/SQL block
    SQLPrepare (hstmt, (SQLCHAR *) query1, SQL_NTS);
    SQLExecute(hstmt);

 //Bind the columns for the results from the first SELECT statement in an anonymous
block.
    SQLBindCol (hstmt, 1, SQL_C_ULONG, &eno, 0, &jind);
    SQLBindCol (hstmt, 2, SQL_C_CHAR, empname, sizeof (empname),&enind);

 //Fetch implicit results through the SQLFetch( ) call.
    while((retCode = SQLFetch(hstmt)) != SQL_NO_DATA)
    {
 //Do whatever you want to do with the data.
    }

     retCode = SQLMoreResults(hstmt);

     if(retCode == SQL_SUCCESS)
    {
      printf("SQLMoreResults returned with SQL_SUCCESS\n");

 //Bind the columns for the results from the second SELECT statement in an anonymous
block.
    SQLBindCol (hstmt, 1, SQL_C_ULONG, &eno, 0, &jind);
    SQLBindCol (hstmt, 2, SQL_C_CHAR, empname, sizeof (empname),&enind);

 //Fetch implicit results through the SQLFetch( ) call.
    while((retCode = SQLFetch(hstmt)) != SQL_NO_DATA)
    {
 //Do whatever you want to do with data.
    }
```

```
  }
}
```

5.11 About Supporting Oracle TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE Column Type in ODBC

The time zone is dictated by the system variable `ORA_SDTZ`. The system variable can be set to `'OS_TZ'`, `'DB_TZ'`, or a valid time zone value. When `ORA_SDTZ` is set to `'OS_TZ'`, the operating system time zone is used. If it is set to `'DB_TZ'`, the default time zone set in the database is used.

By default, when `ORA_SDTZ` is not set, the operating system time zone is used.

> **Note:**
>
> When setting the `ORA_SDTZ` variable in a Microsoft Windows environment -- in the Registry, among system environment variables, or in a command prompt window -- do not enclose the time zone value in quotes.

> **See Also:**
>
> *Oracle Database Globalization Support Guide* for information about Datetime data types and time zone support

Fetching Data from These Time Zone Columns Using the Variable of ODBC Data Type TIMESTAMP_STRUCT

The following example demonstrates how to fetch data from `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE` column using the variable of ODBC datatype `TIMESTAMP_STRUCT`.

Example 5-1 How to Fetch Data from TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE Columns Using the Variable of ODBC Data Type TIMESTAMP_STRUCT

```
int main()
{
...
...
 /* TSTAB table's DDL statement:
  * ----------------------------
  * CREATE TABLE TSTAB (COL_TSTZ  TIMESTAMP WITH TIME ZONE,
  *                     COL_TSLTZ TIMESTAMP WITH LOCAL TIME ZONE);
  *
  * Insert statement:
  * ----------------
  * Sample #1:
```

```
  * ---------
  * INSERT INTO TSTAB VALUES (TIMESTAMP '2010-03-13 03:47:30.123456 America/
Los_Angeles'
  *                           TIMESTAMP '2010-04-14 04:47:30.123456 America/
Los_Angeles');
  *
  * Sample #2:
  * ---------
  * INSERT INTO TSTAB VALUES ('22-NOV-1963 12:30:00.000000 PM',
  *                           '24-NOV-1974 02:30:00.000000 PM');
  *
  * Refer Oracle Database documentations to know more details about TIMESTAMP

  * WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE columns.
  */
 SQLCHAR sqlSelQuery[] = "SELECT COL_TSTZ, COL_TSLTZ FROM TSTAB";
 TIMESTAMP_STRUCT timestampcol1;
 TIMESTAMP_STRUCT timestampcol2;
...
...
  /* Allocate the ODBC statement handle. */
  SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

  /* Execute the statement sqlSelQuery. */
  SQLExecDirect(hstmt, sqlSelQuery, SQL_NTS);

  /* Bind the variable to read the value from the TIMESTAMP WITH TIME ZONE
column. */
  SQLBindCol(hstmt, 1, SQL_C_TIMESTAMP, &timestampcol1,
sizeof(timestampcol1), NULL);

  /* Bind the variable to read the value from the TIMESTAMP WITH LOCAL TIME
ZONE column. */
  SQLBindCol(hstmt, 2, SQL_C_TIMESTAMP, &timestampcol2,
sizeof(timestampcol2), NULL);
...
...
  /* Fetch data from the TSTAB table. */
  retcode = SQLFetch(hstmt);
  /* Values of column COL_TSTZ and COL_TSLTZ are available in variables
  * timestampcol1 and timestampcol2 respectively. Refer to Microsoft ODBC
  * documentation for more information about data type TIMESTAMP_STRUCT. */

...
...
  /* Close the statement. */
  SQLFreeStmt(hstmt, SQL_CLOSE);
  /* Free the statement handle. */
  SQLFreeHandle(SQL_HANDLE_STMT, hstmt); ... ... }
```

Example 5-2 How to Insert Data into TIMESTAMP WITH TIME ZONE and TIMESTAMP WITH LOCAL TIME ZONE Columns

```
int main()
{
```

```
...
...
 SQLCHAR sqlInsQuery[]   = "INSERT INTO TSTAB VALUES (?, ?)";
 TIMESTAMP_STRUCT timestampcol1;
 TIMESTAMP_STRUCT timestampcol2;
...
...
 /* Input the value for column COL_TSTZ in table TSTAB. */
  timestampcol1.year = 2000;
  timestampcol1.month = 1;
  timestampcol1.day   = 1;
  timestampcol1.hour  = 0;
  timestampcol1.minute = 0;
  timestampcol1.second = 1;
  timestampcol1.fraction = 1000;

 /* Input the value for column COL_TSLTZ in table TSTAB. */
  timestampcol1.year = 2012;
  timestampcol1.month = 2;
  timestampcol1.day   = 5;
  timestampcol1.hour  = 10;
  timestampcol1.minute = 30;
  timestampcol1.second = 10;
  timestampcol1.fraction = 1000;
...
...
  /* Allocate the ODBC statement handle. */
  SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
...
...
  /* Bind the input value for column COL_TSTZ. */
  SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_TIMESTAMP, SQL_TIMESTAMP,

    0, 0, &timestampcol1, sizeof(timestampcol1), NULL);

  /* Bind the input value for column COL_TSLTZ. */
  SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_TIMESTAMP, SQL_TIMESTAMP,

    0, 0, &timestampcol2, sizeof(timestampcol2), NULL);
...
...
  /* Execute the statement sqlInsQuery. */
  SQLExecDirect(hstmt, sqlInsQuery, SQL_NTS);

  /* Close the statement. */
  SQLFreeStmt(hstmt, SQL_CLOSE);
  /* Free the statement handle. */
  SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
...
...
}
```

5.12 About the Effect of Setting ORA_SDTZ in Oracle Clients (OCI, SQL*Plus, Oracle Database ODBC Driver, and Others)

Describes the effect of setting the system variable `ORA_SDTZ` in Oracle Clients.

The time zone is dictated by the system variable `ORA_SDTZ`.

The following sections describe the effects of not setting and setting the system variable `ORA_SDTZ` in Oracle Clients (OCI, SQL*Plus, Oracle Database ODBC Driver, and others). The examples in these sections are run in India (GMT+5:30) time zone.

> **See Also:**
>
> *Oracle Database Globalization Support Guide* for more information about setting the session time zone

Environment Setup

To set up the environment, create the following table with `TSLTZ` (TIMESTAMP WITH LOCAL TIME ZONE) column and insert the value of `01/01/2016 00:00` GMT into the `TSLTZ` column as follows:

Example 5-3 How to Set Up the Environment

The following example sets up the environment for the example sections that follow.

```
SQL> create table timezone_demo(col1 TIMESTAMP WITH LOCAL TIME ZONE);

Table created.

SQL> INSERT INTO TIMEZONE_DEMO VALUES(TIMESTAMP '2016-01-01 00:00:00.000000
ETC/GREENWICH');

1 row created.
```

When ORA_SDTZ Is Not Set in the Environment

When `ORA_SDTZ` is not set in the environment, then the operating system (OS) time zone setting is taken as the default time zone for Oracle Clients. For example:

Example 5-4 What Happens When ORA_SDTZ Is Not Set

```
C:\Users\example.ORADEV>set ORA_SDTZ=

C:\Users\example.ORADEV>sqlplus scott/password@//host01.example.com:1521/
ORCL12C1

SQL*Plus: Release 23.0.0.0.0 - Production on Wed Jun 19 13:14:27 2024 Version
23.4.1.24.05

Copyright (c) 1982, 2024, Oracle.  All rights reserved.
```

```
Connected to:Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 -
Development Version 23.4.0.24.00
With the Partitioning, OLAP, Advanced Analytics, and Real Application Testing
options

SQL> select sessiontimezone from dual;

SESSIONTIMEZONE
--------------------------------------------------------------------------------
+05:30

SQL> select * from timezone_demo;

COL1
--------------------------------------------------------------------------------
01-JAN-24 05.30.00.000000 AM
```

Setting ORA_SDTZ to the Operating System (OS) Timezone in the Environment

When `ORA_SDTZ` is set to the operating system (OS) Time zone, the Oracle Client's user session is set to the OS time zone setting. You can either unset it in the environment or set `ORA_SDTZ` to `OS_TZ`. For example:

Example 5-5 What Happens When ORA_SDTZ Is Set to the Operating System (OS) Timezone

```
C:\Users\example.ORADEV>set ORA_SDTZ=OS_TZ

C:\Users\example.ORADEV>sqlplus scott/password@//host01.example.com:1521/
ORCL12C1
SQL*Plus: Release 23.0.0.0.0 - for Oracle Cloud on Wed Jun 19 13:14:27 2024
Version 23.4.1.24.06

Copyright (c) 1982, 2024, Oracle.  All rights reserved.

Connected to:Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 - for
Oracle Cloud Version 23.4.1.24.06
With the Partitioning, OLAP, Advanced Analytics, and Real Application Testing
options

SQL> select sessiontimezone from dual;

SESSIONTIMEZONE
--------------------------------------------------------------------------------
+05:30

SQL> select * from timezone_demo;

COL1
--------------------------------------------------------------------------------
01-JAN-24 05.30.00.000000 AM
```

Setting ORA_SDTZ to a Specific Time Zone in the Environment

The Oracle Client can be set to retrieve the time stamp value adjusted to a specific time zone (for example, Helsinki Time Zone). To do this, you can set `ORA_SDTZ` to the Oracle Time Zone

region name for the corresponding time zone (Oracle Time Zone Region Name for Helsinki Time Zone is `Europe/Helsinki`). For example:

Example 5-6 What Happens When ORA_SDTZ Is Set to a Specific Time Zone

```
C:\Users\example.ORADEV>set ORA_SDTZ=Europe/Helsinki

C:\Users\example.ORADEV>sqlplus scott/password@//host01.example.com:1521/
ORCL12C1

SQL*Plus: Release 23.0.0.0.0 - for Oracle Cloud on Wed Jun 19 13:14:27 2024
Version 23.4.1.24.06

Copyright (c) 1982, 2024, Oracle.  All rights reserved.

Connected to:Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 - for
Oracle Cloud Version 23.4.1.24.06
With the Partitioning, OLAP, Advanced Analytics, and Real Application Testing
options

SQL> select sessiontimezone from dual;

SESSIONTIMEZONE
--------------------------------------------------------------------------------
Europe/Helsinki

SQL> select * from timezone_demo;

COL1
--------------------------------------------------------------------------------
01-JAN-24 02.00.00.000000 AM
```

6
Supported Functionality

This chapter is intended for use by the programmers and provides information about the additional functionality that the Oracle Database ODBC driver supports.

Topics:

6.1 API Conformance

The Oracle Database ODBC driver release 9.2.0.0.0, and higher, support all Core, Level 2, and Level 1 functions.

Also, the Oracle Database ODBC driver release 9.2.0.0.0, and higher, support translation DLLs.

The following topics describe the ODBC API functions implemented by the Oracle Database ODBC driver.

> **See Also:**
>
> - Error Messages
> - Implementation of ODBC API Functions for programmers

6.2 Implementation of ODBC API Functions

The following table describes how the Oracle Database ODBC driver implements specific functions:

Table 6-1 How the Oracle Database ODBC Driver Implements Specific Functions

Function	Description
`SQLConnect`	SQLConnect requires only a DBQ, user ID, and password.
`SQLDriverConnect`	SQLDriverConnect uses the DSN, DBQ, UID, and PWD keywords.
`SQLMoreResults`	Implements ODBC support for implicit results. This is a new API implemented for Oracle Database 12*c* Release 1 (12.1.0.1). See SQLMoreResults Function for more information.
`SQLSpecialColumns`	If `SQLSpecialColumns` is called with the `SQL_BEST_ROWID` attribute, it returns the rowid column.

Table 6-1 (Cont.) How the Oracle Database ODBC Driver Implements Specific Functions

Function	Description
`SQLProcedures` and`SQLProcedureColumns`	See the information that follows.
All catalog functions	If the `SQL_ATTR_METADATA_ID` statement attribute is `SQL_TRUE`, a string argument is treated as an identifier argument, and its case is not significant. In this case, the underscore ("_") and the percent sign ("%") are treated as the actual character, not as a search pattern character. On the other hand, if this attribute is `SQL_FALSE`, it is either an ordinary argument or a pattern value argument and is treated literally, and its case is significant.

6.3 Implementation of the ODBC SQL Syntax

If a comparison predicate has a parameter marker as the second expression in the comparison and the value of that parameter is `SQL_NULL_DATA` with `SQLBindParameter`, the comparison fails. This is consistent with the null predicate syntax in ODBC SQL.

6.4 Implementation of Data Types (Programming)

For programmers, the noteworthy part of the implementation of the data types concerns the `CHAR`, `VARCHAR`, and `VARCHAR2` data types.

For an `fSqlType` value of `SQL_VARCHAR`, `SQLGetTypeInfo` returns the Oracle database data type `VARCHAR2`. For an `fSqlType` value of `SQL_CHAR`, `SQLGetTypeInfo` returns the Oracle database data type `CHAR`.

7
Unicode Support

This chapter provides information about the Unicode support in the Oracle Database ODBC driver.

Topics:

7.1 Unicode Support within the ODBC Environment

The Microsoft or unixODBC ODBC Driver Manager (Driver Manager) makes all ODBC drivers, regardless of if they support Unicode, appear as if they are Unicode compliant. This allows ODBC applications to be written independent of the Unicode capabilities of underlying ODBC drivers.

The extent to which the Driver Manager can emulate Unicode support for ANSI ODBC drivers is limited by the conversions possible between the Unicode data and the local code page. Data loss is possible when the Driver Manager is converting from Unicode to the local code page. Full Unicode support is not possible unless the underlying ODBC driver supports Unicode. the Oracle Database ODBC driver provides full Unicode support.

7.2 Unicode Support in ODBC API

The ODBC API supports both Unicode and ANSI entry points using the "W" and "A" suffix convention. An ODBC application developer need not explicitly call entry points with the suffix. An ODBC application that is compiled with the UNICODE and _UNICODE preprocessor definitions generates the appropriate calls. For example, a call to `SQLPrepare` is compiled as `SQLPrepareW`.

The C data type, `SQL_C_WCHAR`, was added to the ODBC interface to allow applications to specify that an input parameter is encoded as Unicode or to request column data returned as Unicode. The macro `SQL_C_TCHAR` is useful for applications that must be built as both Unicode and ANSI. The `SQL_C_TCHAR` macro compiles as `SQL_C_WCHAR` for Unicode applications and as `SQL_C_CHAR` for ANSI applications.

The SQL data types: `SQL_WCHAR`, `SQL_WVARCHAR`, and `SQL_WLONGVARCHAR` have been added to the ODBC interface to represent columns defined in a table as Unicode. Potentially, these values are returned from calls to `SQLDescribeCol`, `SQLColAttribute`, `SQLColumns`, and `SQLProcedureColumns`.

Unicode encoding is supported for SQL column types `NCHAR`, `NVARCHAR2`, and `NCLOB`. Additionally, Unicode encoding is supported for SQL column types `CHAR` and `VARCHAR2` if the character semantics are specified in the column definition.

The ODBC driver supports these SQL column types and maps them to ODBC SQL data types.

The following table lists the supported SQL data types and the equivalent ODBC SQL data type.

Table 7-1 Supported SQL Data Types and the Equivalent ODBC SQL Data Type

SQL Data Types	ODBC SQL Data Types
CHAR	SQL_CHAR or SQL_WCHAR [1]
VARCHAR2	SQL_VARCHAR or SQL_WVARCHAR [2]
NCHAR	SQL_WCHAR
NVARCHAR2	SQL_WVARCHAR
NCLOB	SQL_WLONGVARCHAR

1 CHAR maps to SQL_WCHAR if the character semantics were specified in the column definition and if the character set for the database is Unicode.
2 VARCHAR2 maps to SQL_WVARCHAR if the character semantics were specified in the column definition and if the character set for the database is Unicode.

7.3 Unicode Functions in the Driver Manager

The Driver Manager (DM) performs the following functions when it detects that the underlying ODBC driver does not support Unicode:

- The DM converts Unicode function calls to ANSI function calls before calling the ANSI ODBC driver. String arguments are converted from Unicode to the local code page. For example, a call to SQLPrepareW is converted to call SQLPrepare. The text of the SQL statement parameter is converted from Unicode to the local code page.
- The DM converts return parameters that are character data from the local code page to Unicode. For example, returning the column name through SQLColAttribute.
- The DM converts data from the local code page to Unicode for columns bound as SQL_C_WCHAR.
- The DM converts data from Unicode to the local code page for input parameters bound as SQL_C_WCHAR.

7.4 SQLGetData Performance

The SQLGetData function allows an ODBC application to specify the data type to receive a column as after the data has been fetched. OCI requires the Oracle Database ODBC driver to specify the data type before it is fetched. In this case, the Oracle Database ODBC driver uses the knowledge it has about the data type of the column as defined in the database to determine how to best default to fetching the column through OCI.

If a column that contains character data is not bound by SQLBindCol, the Oracle Database ODBC driver must determine if it must fetch the column as Unicode or as the local code page. The driver could default to receiving the column as Unicode, however, this may result in as many as two unnecessary conversions. For example, if the data were encoded in the database as ANSI, there would be an ANSI to Unicode conversion to fetch the data into the Oracle Database ODBC driver. If the ODBC application then requested the data as SQL_C_CHAR, there would be an additional conversion to revert the data back to its original encoding.

The default encoding of the Oracle client is used when fetching data. However, an ODBC application can overwrite this default and fetch the data as Unicode by binding the column or the parameter as the `WCHAR` data type.

7.5 Unicode Samples

As the Oracle Database ODBC driver itself was implemented using TCHAR macros, Oracle recommends that ODBC application programs use TCHAR to take advantage of the driver.

The following links are program examples showing how to use TCHAR, which becomes the `WCHAR` data type in case you compile with `UNICODE` and `_UNICODE`.

- Example 1: Connection to Database
- Example 2: Simple Retrieval
- Example 3: Retrieval Using SQLGetData (Binding After Fetch)
- Example 4: Simple Update
- Example 5: Update and Retrieval for Long Data (CLOB)

Example 1: Connection to Database

No difference other than specifying Unicode literals for `SQLConnect`.

```
SQLHENV envHnd;
SQLHDBC conHnd;
SQLHSTMT stmtHnd;
RETCODE rc;

rc = SQL_SUCCESS;

// ENV is allocated
rc = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &envHnd);
// Connection Handle is allocated
rc = SQLAllocHandle(SQL_HANDLE_DBC, envHnd, &conHnd);
rc = SQLConnect(conHnd, _T("stpc19"), SQL_NTS, _T("scott"), SQL_NTS, _T("tiger"),
 SQL_NTS);
.
.
.
if (conHnd)
{
  SQLDisconnect(conHnd);
  SQLFreeHandle(SQL_HANDLE_DBC, conHnd);
}
if (envHnd)
  SQLFreeHandle(SQL_HANDLE_ENV, envHnd);
```

Example 2: Simple Retrieval

The following example retrieves the employee names and the job titles from the `EMP` table. With the exception that you must specify TCHAR compliant data to every ODBC function, there is no difference to the ANSI case. If the case is a Unicode application, you have to specify the length of the buffer to the `BYTE` length when you call `SQLBindCol` (for example, `sizeof(ename)`).

```
/*
** Execute SQL, bind columns, and Fetch.
** Procedure:
**
```

```
** SQLExecDirect
** SQLBindCol
** SQLFetch
**
 */
static SQLTCHAR *sqlStmt = _T("SELECT ename, job FROM emp");
SQLTCHAR ename[50];
SQLTCHAR job[50];
SQLINTEGER enamelen, joblen;

_tprintf(_T("Retrieve ENAME and JOB using SQLBindCol 1.../n[%s]/n"), sqlStmt);

/* Step 1: Prepare and Execute */
rc = SQLExecDirect(stmtHnd, sqlStmt, SQL_NTS); /* select */
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Bind Columns */
rc = SQLBindCol(stmtHnd, 1, SQL_C_TCHAR, ename, sizeof(ename), &enamelen);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

rc = SQLBindCol(stmtHnd, 2, SQL_C_TCHAR, job, sizeof(job), &joblen);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

do
{
  /* Step 3: Fetch Data */
  rc = SQLFetch(stmtHnd);
  if (rc == SQL_NO_DATA)
    break;
  checkSQLErr(envHnd, conHnd, stmtHnd, rc);
  _tprintf(_T("ENAME = %s, JOB = %s/n"), ename, job);
} while (1);
_tprintf(_T("Finished Retrieval/n/n"));
```

Example 3: Retrieval Using SQLGetData (Binding After Fetch)

This example shows how to use `SQLGetData`. For those who are not familiar with ODBC programming, the fetch is allowed before binding the data using `SQLGetData`, unlike in an OCI program. There is no difference to the ANSI application in terms of Unicode-specific issues.

```
/*
** Execute SQL, bind columns, and Fetch.
** Procedure:
**
** SQLExecDirect
** SQLFetch
** SQLGetData
 */
static SQLTCHAR *sqlStmt = _T("SELECT ename,job FROM emp"); // same as Case 1.
SQLTCHAR ename[50];
SQLTCHAR job[50];

_tprintf(_T("Retrieve ENAME and JOB using SQLGetData.../n[%s]/n"), sqlStmt);
if (rc != SQL_SUCCESS)
{
  _tprintf(_T("Failed to allocate STMT/n"));
  goto exit2;
}

/* Step 1: Prepare and Execute */
rc = SQLExecDirect(stmtHnd, sqlStmt, SQL_NTS); // select
checkSQLErr(envHnd, conHnd, stmtHnd, rc);
```

```
do
{
  /* Step 2: Fetch */
  rc = SQLFetch(stmtHnd);
  if (rc == SQL_NO_DATA)
    break;

  checkSQLErr(envHnd, conHnd, stmtHnd, rc);

  /* Step 3: GetData */
  rc = SQLGetData(stmtHnd, 1, SQL_C_TCHAR, (SQLPOINTER)ename, sizeof(ename), NULL);
  checkSQLErr(envHnd, conHnd, stmtHnd, rc);

  rc = SQLGetData(stmtHnd, 2, SQL_C_TCHAR, (SQLPOINTER)job, sizeof(job), NULL);
  checkSQLErr(envHnd, conHnd, stmtHnd, rc);

  _tprintf(_T("ENAME = %s, JOB = %s/n"), ename, job);

} while (1);

_tprintf(_T("Finished Retrieval/n/n"));
```

Example 4: Simple Update

This example shows how to update data. Likewise, the length of data for `SQLBindParameter` has to be specified with the `BYTE` length, even in the case of a Unicode application.

```
/
*
** Execute SQL, bind columns, and Fetch.
** Procedure:
**
** SQLPrepare
** SQLBindParameter
** SQLExecute
*/

static SQLTCHAR *sqlStmt = _T("INSERT INTO emp(empno,ename,job) VALUES(?,?,?)");
static SQLTCHAR *empno = _T("9876"); // Emp No
static SQLTCHAR *ename = _T("ORACLE"); // Name
static SQLTCHAR *job = _T("PRESIDENT"); // Job

_tprintf(_T("Insert User ORACLE using SQLBindParameter.../n[%s]/n"), sqlStmt);

/* Step 1: Prepare */

rc = SQLPrepare(stmtHnd, sqlStmt, SQL_NTS);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Bind Parameter */

rc = SQLBindParameter(stmtHnd, 1, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_DECIMAL,4, 0,
(SQLPOINTER)empno, 0, NULL);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

rc = SQLBindParameter(stmtHnd, 2, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_CHAR,
lstrlen(ename)*sizeof(TCHAR), 0, (SQLPOINTER)ename, lstrlen(ename)*sizeof(TCHAR), NULL);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

rc = SQLBindParameter(stmtHnd, 3, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_CHAR,
lstrlen(job)*sizeof(TCHAR), 0, (SQLPOINTER)job, lstrlen(job)*sizeof(TCHAR), NULL);
```

```
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 3: Execute */

rc = SQLExecute(stmtHnd);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);
```

Example 5: Update and Retrieval for Long Data (CLOB)

This example may be the most complicated case to update and retrieve data for long data, like `CLOB`, in Oracle. Because the length of data must be the `BYTE` length, `lstrlen(TCHAR data)*sizeof(TCHAR)` is needed to derive the `BYTE` length.

```
/*
** Execute SQL, bind columns, and Fetch.
** Procedure:
**
** SQLPrepare
** SQLBindParameter
** SQLExecute
** SQLParamData
** SQLPutData
**
** SQLExecDirect
** SQLFetch
** SQLGetData
 */

static SQLTCHAR *sqlStmt1 = _T("INSERT INTO clobtbl(clob1) VALUES(?)");
static SQLTCHAR *sqlStmt2 = _T("SELECT clob1 FROM clobtbl");
SQLTCHAR clobdata[1001];
SQLTCHAR resultdata[1001];
SQLINTEGER ind = SQL_DATA_AT_EXEC;
SQLTCHAR *bufp;
SQLTCHAR ch;
int clobdatalen, chunksize, dtsize, retchklen, i, len;

_tprintf(_T("Insert CLOB1 using SQLPutData.../n[%s]/n"), sqlStmt1);

/* Set CLOB Data *

for (i=0, ch=_T('A'); i< sizeof(clobdata)/sizeof(SQLTCHAR); ++i, ++ch)
{
  if (ch > _T('Z'))
  ch = _T('A');
  clobdata[i] = ch;
}

clobdata[sizeof(clobdata)/sizeof(SQLTCHAR)-1] = _T('/0');
clobdatalen = lstrlen(clobdata);
chunksize = clobdatalen / 7;

/* Step 1: Prepare */
rc = SQLPrepare(stmtHnd, sqlStmt1, SQL_NTS);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Bind Parameter with SQL_DATA_AT_EXEC */
rc = SQLBindParameter(stmtHnd, 1, SQL_PARAM_INPUT, SQL_C_TCHAR, SQL_LONGVARCHAR,
clobdatalen*sizeof(TCHAR), 0, (SQLPOINTER)clobdata, clobdatalen*sizeof(TCHAR), &ind);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 3: Execute */
```

```
rc = SQLExecute(stmtHnd);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);
sdhamoth: Continuation:


/* Step 4: ParamData (initiation) */
rc = SQLParamData(stmtHnd, (SQLPOINTER*)&bufp);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

for (dtsize=0, bufp = clobdata; dtsize < clobdatalen; dtsize += chunksize, bufp +=
chunksize)
{
  if (dtsize+chunksize<clobdatalen)
    len = chunksize;
  else
    len = clobdatalen-dtsize;

  /* Step 5: PutData */
  rc = SQLPutData(stmtHnd, (SQLPOINTER)bufp, len*sizeof(TCHAR));
  checkSQLErr(envHnd, conHnd, stmtHnd, rc);
}

/* Step 6: ParamData (termination) */
rc = SQLParamData(stmtHnd, (SQLPOINTER*)&bufp);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

rc = SQLFreeStmt(stmtHnd, SQL_CLOSE);
_tprintf(_T("Finished Update/n/n"));

rc = SQLAllocStmt(conHnd, &stmtHnd);
if (rc != SQL_SUCCESS)
{
  _tprintf(_T("Failed to allocate STMT/n"));
  goto exit2;
}

/* Clear Result Data */
memset(resultdata, 0, sizeof(resultdata));
chunksize = clobdatalen / 15; /* 15 times to put */

/* Step 1: Prepare */
rc = SQLExecDirect(stmtHnd, sqlStmt2, SQL_NTS); /* select */
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

/* Step 2: Fetch */
rc = SQLFetch(stmtHnd);
checkSQLErr(envHnd, conHnd, stmtHnd, rc);

for(dtsize=0, bufp = resultdata; dtsize < sizeof(resultdata)/sizeof(TCHAR) && rc !=
SQL_NO_DATA; dtsize += chunksize-1, bufp += chunksize-1)
{
  if (dtsize+chunksize<sizeof(resultdata)/sizeof(TCHAR))
    len = chunksize;
  else
    len = sizeof(resultdata)/sizeof(TCHAR)-dtsize;

  /* Step 3: GetData */
  rc = SQLGetData(stmtHnd, 1, SQL_C_TCHAR, (SQLPOINTER)bufp, len*sizeof(TCHAR),
&retchklen);
}

if (!_tcscmp(resultdata, clobdata))
```

```
{
  _tprintf(_T("Succeeded!!/n/n"));
}
else
{
  _tprintf(_T("Failed!!/n/n"));
}
```

8
Performance and Tuning

This chapter provides necessary information for programmers to manage the performance and tuning of the Oracle Database ODBC driver.

Topics:

8.1 General ODBC Programming Tips

This section describes some general programming tips to improve the performance of an ODBC application.

- Enable connection pooling if the application frequently connects to and disconnects from a data source. Reusing pooled connections is extremely efficient compared to reestablishing a connection.
- Minimize the number of times a statement must be prepared. Where possible, use bind parameters to make a statement reusable for different parameter values. Preparing a statement once and executing it several times is much more efficient than preparing a statement for every `SQLExecute`.
- Do not include columns in a `SELECT` statement if you know the application will not retrieve them; especially `LONG` columns. Due to the nature of the database server protocols, the ODBC driver must fetch the entire contents of a `LONG` column if it is included in the `SELECT` statement, regardless of if the application binds the column or does a `SQLGetData`.
- If you are performing transactions that do not update the data source, set the `SQL_ATTR_ACCESS_MODE` attribute of the ODBC `SQLSetConnectAttr` function to `SQL_MODE_READ_ONLY`.
- If you are not using ODBC escape clauses, set the `SQL_ATTR_NOSCAN` attribute of the ODBC `SQLSetConnectAttr` function or the ODBC `SQLSetStmtAttr` function to true.
- Use the ODBC `SQLFetchScroll` function instead of the ODBC `SQLFetch` function for retrieving data from tables that have a large number of rows.
- Enable OCI statement caching when the same SQL statements are used multiple times (`StatementCache=T`).
- Binding `NUMBER` columns as `FLOAT` speeds up query execution (`BindAsFLOAT=T`).
- While fetching `LONG` or `LONG RAW` set `MaxLargeData=<value>` for optimum performance.
- Setting `UseOCIDescribeAny=T` for applications that make heavy calls to small packaged procedures with return `Ref Cursor` improves performance.

8.2 Data Source Configuration Options

This topic discusses performance implications of the following ODBC data source configuration options:

Topics:

Enable Result Sets

This option enables the support of returning result sets (for example, `RefCursor`) from procedure calls. The default is enabling the returning of result sets.

The ODBC driver must query the database server to determine the set of parameters for a procedure and their data types to determine if there are any `RefCursor` parameters. This query incurs an additional network round trip the first time any procedure is prepared and executed.

Enable LOBs

This option enables the support of inserting and updating LOBs. By default, it is enabled.

The ODBC driver must query the database server to determine the data types of each parameter in an `INSERT` or `UPDATE` statement to determine if there are any LOB parameters. This query incurs an additional network round trip the first time any `INSERT` or `UPDATE` is prepared and executed.

Bind TIMESTAMP as DATE

Binds `SQL_TIMESTAMP` parameters as the appropriate Oracle data type. If this option is `TRUE`, `SQL_TIMESTAMP` binds as the Oracle `DATE` data type. If this option is `FALSE`, `SQL_TIMESTAMP` binds as the Oracle `TIMESTAMP` data type (which is the default).

Enable Closing Cursors

The `SQL_CLOSE` option of the ODBC function, `SQLFreeStmt`, is supposed to close associated cursors with a statement and discard all pending results. The application can reopen the cursor by executing the statement again without doing a `SQLPrepare` again. A typical scenario for this is an application that is idle for a while but reuses the same SQL statement. While the application is idle, it might free up associated server resources.

The Oracle Call Interface (OCI), on which the Oracle Database ODBC driver is layered, does not support the functionality of closing cursors. So, by default, the `SQL_CLOSE` option has no effect in the Oracle Database ODBC driver. The cursor and associated resources remain open on the database server.

Enabling this option causes the associated cursor to be closed on the database server. However, this results in the parse context of the SQL statement being lost. The ODBC application can execute the statement again without calling `SQLPrepare`. However, internally

the ODBC driver must prepare and execute the statement all over. Enabling this option severely impacts performance of applications that prepare a statement once and execute it repeatedly.

Enable this option only if freeing the resources on the server is absolutely necessary.

Enable Thread Safety

If an application is single-threaded, this option can be disabled. By default, the ODBC driver ensures that access to all internal structures (environment, connection, statement) are thread-safe. Single-threaded applications can eliminate some of the thread safety overhead by disabling this option. Disabling this option typically shows a minor performance improvement.

Fetch Buffer Size

Set the Fetch Buffer Size in the Oracle Options tab of the Oracle ODBC Driver Configuration Dialog Box to a value specified in bytes. This value determines how many rows of data at a time the ODBC driver prefetches from an Oracle Database to the client's cache, regardless of the number of rows the application program requests in a single query, thus improving performance.

Applications that typically fetch fewer than 20 rows of data at a time improve their response time, particularly over slow network connections or on heavily loaded servers. Setting this too high can worsen response time or consume large amounts of memory. The default is 64,000 bytes. Choose a value that works best for your application.

> **Note:**
>
> When `LONG` and LOB data types are present, the number of rows prefetched by the ODBC driver is not determined by the Fetch Buffer Size. The inclusion of the `LONG` and LOB data types minimizes the performance improvement and could result in excessive memory use. The ODBC driver disregards the Fetch Buffer Size and prefetches a set number of rows in the presence of the `LONG` and LOB data types.

8.3 DATE and TIMESTAMP Data Types

If a `DATE` column in the database is used in a `WHERE` clause and the column has an index, there can be an impact on performance. For example:

```
SELECT * FROM EMP WHERE HIREDATE = ?
```

In this example, an index on the `HIREDATE` column could be used to make the query execute quickly. But, because `HIREDATE` is actually a `DATE` value and the ODBC driver supplies the parameter value as `TIMESTAMP`, the Oracle server's query optimizer must apply a conversion function. To prevent incorrect results (as might happen if the parameter value had non-zero fractional seconds), the optimizer applies the conversion to the `HIREDATE` column resulting in the following statement:

```
SELECT * FROM EMP WHERE TO_TIMESTAMP(HIREDATE) = ?
```

Unfortunately, this has the effect of disabling the use of the index on the `HIREDATE` column and, instead, the server performs a sequential scan of the table. If the table has many rows, this can take a long time. As a workaround for this situation, the ODBC driver has the connection option to `Bind TIMESTAMP as DATE`. When this option is enabled, the ODBC driver binds

SQL_TIMESTAMP parameters as the Oracle DATE data type instead of the Oracle TIMESTAMP data type. This allows the query optimizer to use any index on the DATE columns.

Note:

This option is intended for use only with Microsoft Access or other similar programs that bind DATE columns as TIMESTAMP columns. Do not use this option when there are actual TIMESTAMP columns present or when data loss may occur. Microsoft Access executes such queries using whatever columns are selected as the primary key.

See Also:

Implementation of Data Types (Advanced)

A.1 Appendix: Unsupported Features and Known Issues

Topics:

A.1.1 Unsupported Features

The Oracle Database ODBC driver does not support the following ODBC 3.0 features:

- Interval data types
- Shared connections
- Shared environments

The Oracle Database ODBC driver does not support the following SQL string functions:

- `BIT_LENGTH`
- `CHAR_LENGTH`
- `CHARACTER_LENGTH`
- `DIFFERENCE`
- `OCTET_LENGTH`
- `POSITION`

The Oracle Database ODBC driver does not support the following SQL numeric functions:

- `ACOS`
- `ASIN`
- `ATAN`
- `ATAN2`
- `COT`
- `DEGREES`
- `RADIANS`
- `RAND`
- `ROUND`

The Oracle Database ODBC driver does not support the following SQL time, date, and interval functions:

- `CURRENT_DATE`
- `CURRENT_TIME`
- `CURRENT_TIMESTAMP`

- `EXTRACT`
- `TIMESTAMPDIFF`

A.1.2 Known Limitations

The Oracle Database ODBC driver does not support the following:

- ODBC ASYNC interface
- Control-C to cancel execution in an application

A.1.3 Known Software Issues

The following are the known software issues and unsupported usage in the Oracle Database ODBC driver:

- The `SQLSetStmtOption SQL_QUERY_TIMEOUT` does not work if the database server is running on Windows NT. As a workaround, setting `BREAK_POLL_SKIP=1` in the server's `sqlnet.ora` file solves the problem. By default, this is set to 100, and the database would not check for a time-out set by the ODBC application.
- `SQLBindParameter`, when used to bind a buffer as `SQL_PARAM_INPUT_OUTPUT` and having a PL/SQL procedure with `IN OUT` parameter, and if the parameter is not changed in the procedure, then the driver does not return `SQL_NULL_DATA` in `StrLen_or_IndPtr`.
- The Oracle Database ODBC driver does not support the usage of `Keyset` cursors with the `CASE` clause in a SQL `SELECT` query.

Index

T

U