

Oracle® Database
2 Day + .NET Developer's Guide
11g Release 1 (11.1)
B28844-01

July 2008

Oracle Database 2 Day + .NET Developer's Guide, 11g Release 1 (11.1)

B28844-01

Copyright © 2006, 2008, Oracle. All rights reserved.

Primary Authors: Janis Greenberg, Roza Leyderman

Contributing Authors: John Paul Cook, Mark Williams

Contributors: Alex Keh, Christian Shay

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface	vii
Audience	vii
Documentation Accessibility	vii
Related Documents	viii
Conventions	viii
1 Introduction	
About This Guide	1-1
What is the Microsoft .NET Framework	1-2
Overview of Oracle Data Provider for .NET	1-2
Overview of Oracle Developer Tools for Visual Studio	1-2
Overview of .NET Stored Procedures	1-2
Overview of Oracle Providers for ASP.NET	1-3
2 Installing .NET Products	
What You Need	2-1
Oracle Database	2-1
Sample Data	2-1
Oracle Data Access Components	2-1
Oracle Database Extensions for .NET	2-2
Visual Studio 2008	2-2
Installing .NET Products	2-2
Configuring a NET Connect Alias	2-8
3 Building a Simple .NET Application Using ODP.NET	
Creating a New Project	3-1
Adding a Reference	3-4
Adding Namespace Directives	3-6
Designing the User Interface	3-7
Writing the Connection Code	3-11
Compiling and Running the Application	3-14
Error Handling	3-16
Using Try-Catch-Finally Block Structure	3-17
Handling General Errors	3-17
Handling Common Oracle Errors	3-18

4	Retrieving and Updating with Oracle Data Provider for .NET	
	Using the Command Object	4-1
	Retrieving Data: a Simple Query	4-2
	Retrieving Data: Bind Variables	4-4
	Retrieving Data: Multiple Values	4-7
	Using the DataSet Class with Oracle Data Provider for .NET	4-8
	Enabling Updates to the Database	4-11
	Inserting, Deleting, and Updating Data	4-13
5	Using Oracle Developer Tools for Visual Studio	
	Using Oracle Developer Tools	5-1
	Connecting to the Oracle Database	5-1
	Creating a Table and Its Columns	5-5
	Creating a Table Index	5-8
	Adding Table Constraints	5-10
	Adding Data to a Table	5-13
	Generating Code Automatically to Display and Update Data	5-15
6	Using PL/SQL Stored Procedures and REF CURSORS	
	Introduction to PL/SQL Stored Procedures	6-1
	Introduction to PL/SQL Packages and Package Bodies	6-1
	Introduction to REF CURSORS	6-2
	Creating a PL/SQL Stored Procedure that Uses REF CURSORS	6-2
	Modifying an ODP.NET Application to Run Stored Procedures	6-9
	Running a PL/SQL Stored Procedure Using an ODP.NET Application	6-10
7	Developing and Deploying .NET Stored Procedures	
	Overview of .NET Stored Procedures	7-1
	Starting the Common Language Runtime Service	7-1
	Creating a Connection as SYSDBA	7-2
	Creating an Oracle Project	7-3
	Creating .NET Stored Functions and Procedures	7-4
	Deploying .NET Stored Functions and Procedures	7-7
	Running .NET Stored Functions and Procedures	7-14
	Running .NET Stored Procedure in a Query Window	7-16
8	Including Globalization Support	
	Introduction to Global Applications	8-1
	Developing Global Applications with the .NET Framework	8-1
	Presenting Data in the Correct User Local Convention	8-2
	Connecting to SQL*Plus	8-2
	Using Oracle Date Formats	8-2
	Using Oracle Number Formats	8-5
	Using Oracle Linguistic Sorts	8-6
	Oracle Error Messages	8-7

Synchronizing the .NET and Oracle Database Locale Environments	8-8
Client Globalization Support in Oracle Data Provider for .NET	8-8
Client Globalization Settings	8-8
Using Session Globalization Settings	8-9
Thread-Based Globalization Settings	8-14

A Starting and Stopping an Oracle Database Instance

B Copying a Form

Index

Preface

This document is intended as an introduction to application development on Oracle Database with Oracle technologies for the Microsoft .NET Framework.

Audience

We assume that users of this book have already read the *Oracle Database 2 Day DBA* and the *Oracle Database 2 Day Developer's Guide*, are familiar with basics of SQL and PL/SQL, and know how to use Microsoft Visual Studio.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at <http://www.oracle.com/accessibility/>.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Related Documents

For more information, see the following documents in Oracle Database documentation set:

- *Oracle Data Provider for .NET Developer's Guide*
- *Oracle Database Extensions for .NET Developer's Guide*
- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day Developer's Guide*
- *Oracle Developer Tools for Visual Studio Dynamic Help*
- *Oracle Net Services Administrator's Guide*
- *Oracle Database Express Edition Installation Guide for Microsoft Windows*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	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.

Introduction

This chapter contains:

- [About This Guide](#)
- [What is the Microsoft .NET Framework](#)
- [Overview of Oracle Data Provider for .NET](#)
- [Overview of Oracle Developer Tools for Visual Studio](#)
- [Overview of .NET Stored Procedures](#)
- [Overview of Oracle Providers for ASP.NET](#)

About This Guide

This guide serves as a quick start guide, which describes Oracle technologies for the Microsoft .NET Framework, including the key features of Oracle Data Provider for .NET and Oracle Developer Tools for Visual Studio. It leads you through installation and configuration, shows how to build basic applications using Oracle .NET products, and how to create and use both PL/SQL and .NET stored procedures.

Note: This guide was created using Microsoft Visual Studio 2008. If you are using Microsoft Visual Studio 2005, you may notice differences in screen shots, shortcuts, menu options, and generated code, but generally the differences should be minor and not cause problems.

After working through this book, you will be ready to continue with more extensive information available in the Oracle Database documentation library.

See Also:

- Visual Studio Dynamic help
- *Oracle Data Provider for .NET Developer's Guide*
- *Oracle Database Extensions for .NET Developer's Guide*
- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day Developer's Guide*

What is the Microsoft .NET Framework

The Microsoft .NET Framework is a multi-language environment for building, deploying, and running applications and XML Web services. Its main components are:

Common Language Runtime

The Common Language Runtime, or CLR, is a language-neutral development and run-time environment that provides services that help manage running applications

Framework Class Libraries

The Framework Class Libraries, or FCL, provide a consistent, object-oriented library of prepackaged functionality.

Overview of Oracle Data Provider for .NET

Oracle Data Provider for .NET (ODP.NET) provides fast and efficient ADO.NET data access from .NET client applications to Oracle databases and access to other Oracle Database features.

ODP.NET allows developers to take advantage of advanced Oracle database functionality, including Real Application Clusters, XML DB, and advanced security.

Overview of Oracle Developer Tools for Visual Studio

Oracle Developer Tools for Visual Studio (ODT) is a set of application tools that integrate with the Visual Studio environment. These tools provide graphical user interface access to Oracle functionality, enable the user to perform a wide range of application development tasks, and improve development productivity and ease of use. Oracle Developer Tools supports the programming and implementation of .NET stored procedures using Visual Basic, C#, and other .NET languages.

These are some of the Oracle Developer Tools features:

- Integration with Server Explorer for browsing the Oracle schema.
- Designers and wizards to create and alter schema objects.
- The ability to drag and drop schema objects onto a .NET form to automatically generate code.
- A PL/SQL editor and debugger with integrated context-sensitive dynamic help.
- An Oracle Data Window for performing routine database tasks such as inserting and updating data or testing stored procedures in the Visual Studio environment.
- An Oracle Query Window for executing SQL statements or PL/SQL scripts.
- An Oracle Deployment Wizard for .NET described "[Integration with Microsoft Visual Studio](#)" on page 1-3.

Overview of .NET Stored Procedures

Oracle Database Extensions for .NET is a database option for Oracle Database on Windows. It makes it possible to build and run .NET stored procedures or functions with Oracle Database for Microsoft Windows using Visual Basic .NET or Visual C#.

See Also: *Oracle Database Extensions for .NET Developer's Guide*

Integration with Microsoft Visual Studio

After building .NET procedures and functions into a .NET assembly, you can deploy them in Oracle Database using the Oracle Deployment Wizard for .NET, a component of the Oracle Developer Tools for Visual Studio.

Overview of Oracle Providers for ASP.NET

Oracle Providers for ASP.NET offer ASP.NET developers an easy to use method to store state common to web applications within an Oracle database. These providers are modeled on existing Microsoft ASP.NET providers, sharing similar schema and programming interfaces to provide .NET developers a familiar interface. Oracle supports the Membership, Profile, Role and other providers.

Oracle Providers for ASP.NET classes, their use, installation, and requirements are described in *Oracle Providers for ASP.NET Developer's Guide*, which is also provided as dynamic help.

Installing .NET Products

This chapter contains:

- [What You Need](#)
- [Installing .NET Products](#)
- [Configuring a NET Connect Alias](#)

What You Need

This section lists the products and database schemas you need to run the examples provided in this guide.

Oracle Database

You must have Oracle Database installed, either locally or on a remote computer.

Note: This samples in this guide all require Oracle Database 11g client. However, you may use any Oracle Database 9i Release 2 or higher as they are supported with this client.

You can administer the database with the user interface, Enterprise Manager, which can run scripts and queries, and more.

See Also: *Oracle Database Express Edition Installation Guide for Microsoft Windows* if you do not have the Oracle Database installed and configured

Sample Data

The sample data used in this book is contained in the HR schema, one of the Oracle Sample Schemas. The Sample Schemas are included as part of the Oracle Database installation.

See Also: *Oracle Database Sample Schemas* for the HR data model and table descriptions

Oracle Data Access Components

Oracle Data Access Components (ODAC) is a collection of tools that include:

- Oracle Developer Tools for Visual Studio

- Oracle Data Provider for .NET
- Oracle Providers for ASP.NET
- Oracle Provider for OLE DB
- Oracle Objects for OLE
- Oracle ODBC Driver
- Oracle Services for Microsoft Transaction Server
- Oracle SQL*Plus
- Oracle Instant Client

Oracle Database Extensions for .NET

Oracle Database Extensions for .NET is installed as part of the Oracle Database installation on Windows. The ODAC installation provides an upgrade to Oracle Database Extensions for .NET, as indicated in Step # 8 of the section "[Installing .NET Products](#)" on page 2-2.

Visual Studio 2008

Before proceeding with instructions in this book, you must install Visual Studio 2008.

Installing .NET Products

These steps demonstrate how to install Oracle Developer Tools for Visual Studio (ODT) and Oracle Data Provider for .NET and other ODAC products once Visual Studio is installed.

Note: Please note that as new versions of Oracle .NET products are released, the install process may change slightly from what is shown in this guide. The screenshots are based on Oracle Data Access Components (ODAC) version 11.1.0.6.21.

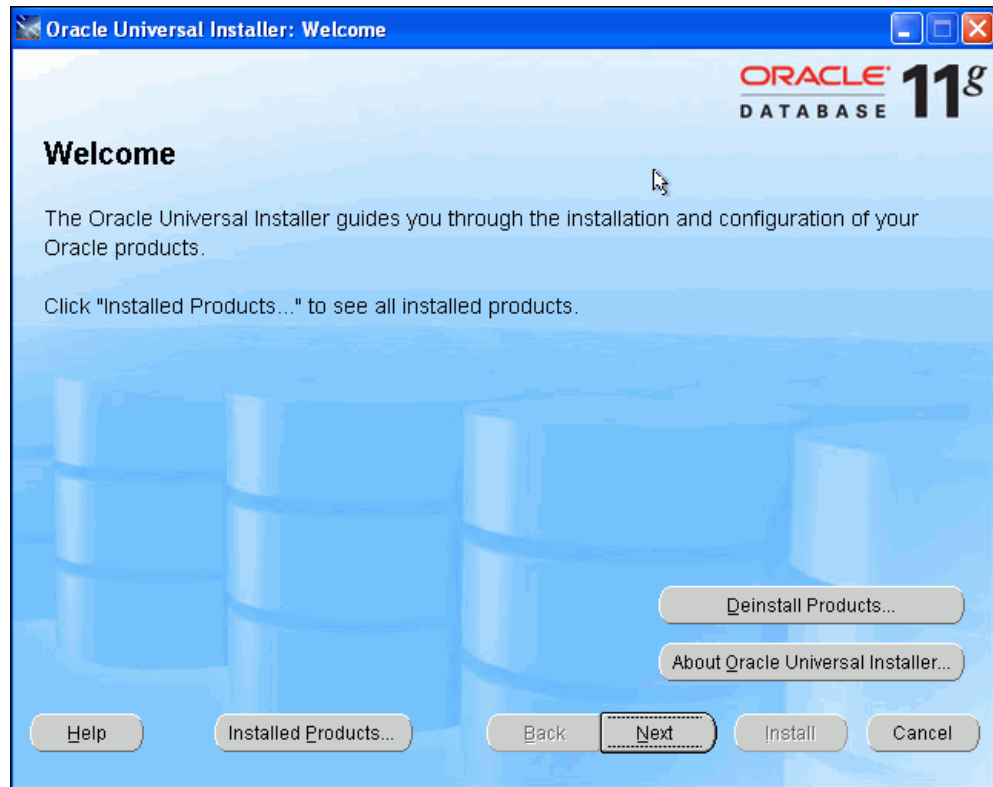
To install:

1. In your Internet browser, navigate to the following location, and download ODAC with Oracle Developer Tools for Visual Studio:

<http://www.oracle.com/technology/software/tech/windows/odpnet/index.html>

2. Extract all the files from the zip file to a folder in your file system.
3. Double-click **Setup.exe**.

Oracle Installer launches. A screen appears briefly to detect required dependencies and then the Oracle Universal Installer (OUI) Welcome screen appears.



4. Click Next.

The Select a Product to Install screen appears

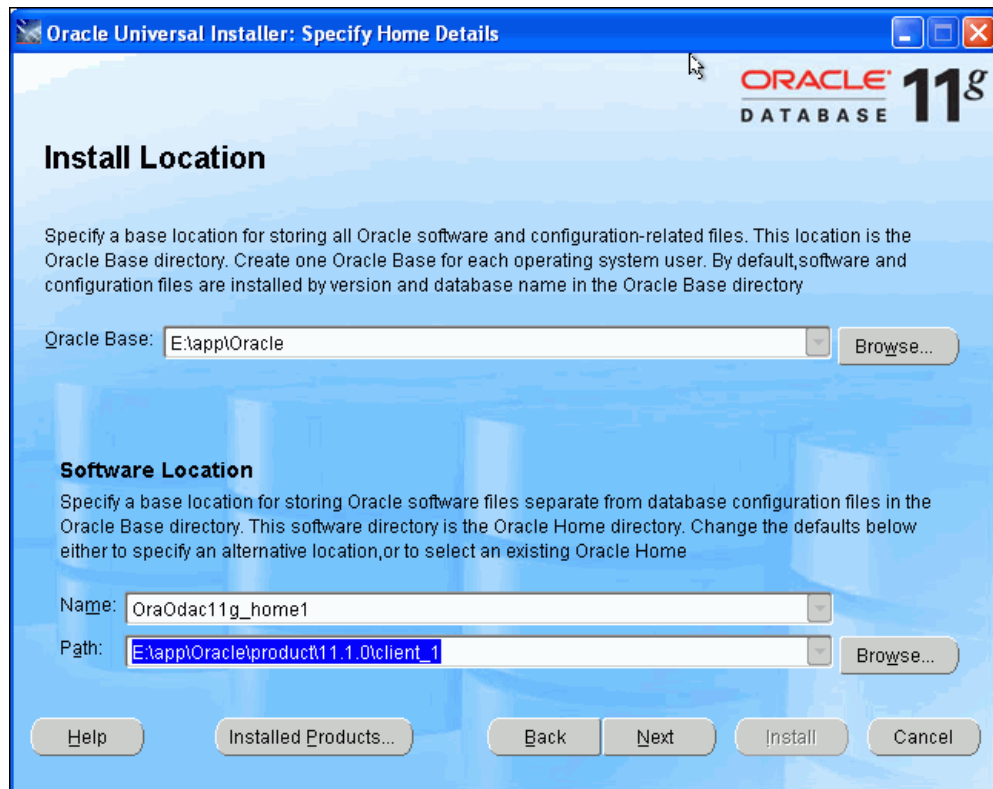


5. Select the first option.

This option, ODAC for Oracle Client, installs only products that are used in a client Oracle home. The second option, ODAC for Oracle Server, allows you to install directly into an Oracle home that contains an Oracle database.

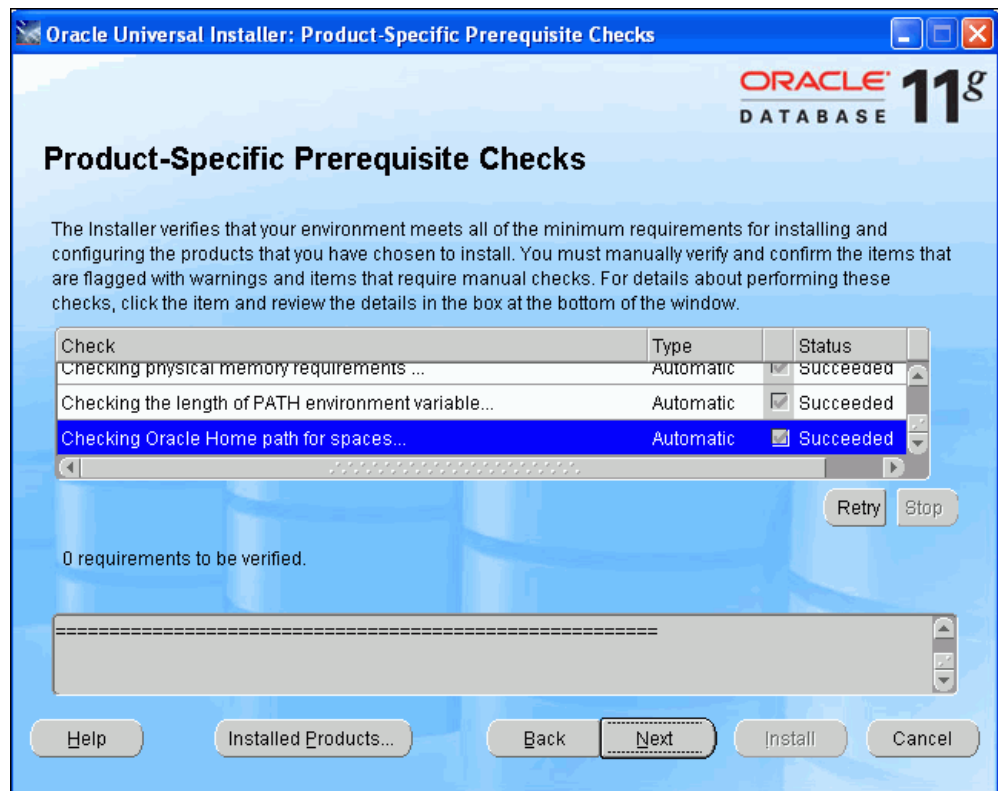
6. Click **Next**.

The Install Location window appears, allowing you to choose the installation location. By default, a new client Oracle home is created. For the purposes of this guide, accept the default which will create a new Oracle home.



7. Click **Next**.

The installer performs prerequisite check. The status for each should be succeeded.

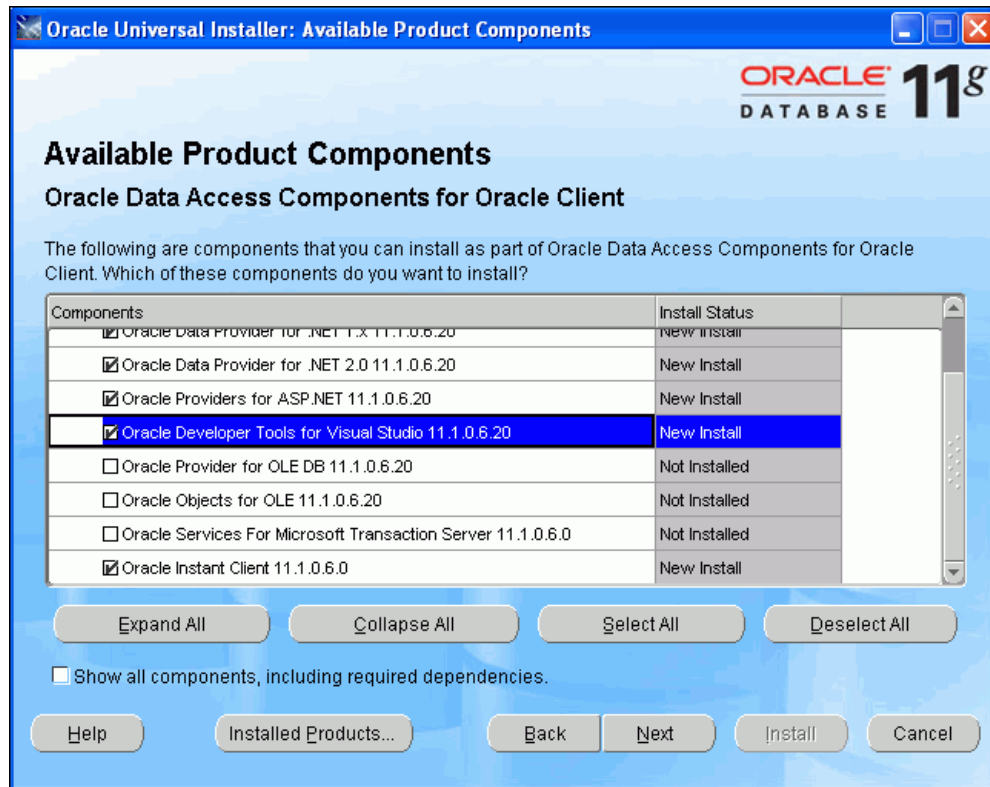


8. Click Next.

The Available Product Components screen appears.

Please be sure that the following are checked:

- Oracle Data Provider for .NET 1.x
- Oracle Data Provider for .NET 2.0
- Oracle Providers for ASP.NET 11.0
- Oracle Developer Tools for Visual Studio 11.0
- Oracle Instant Client 11

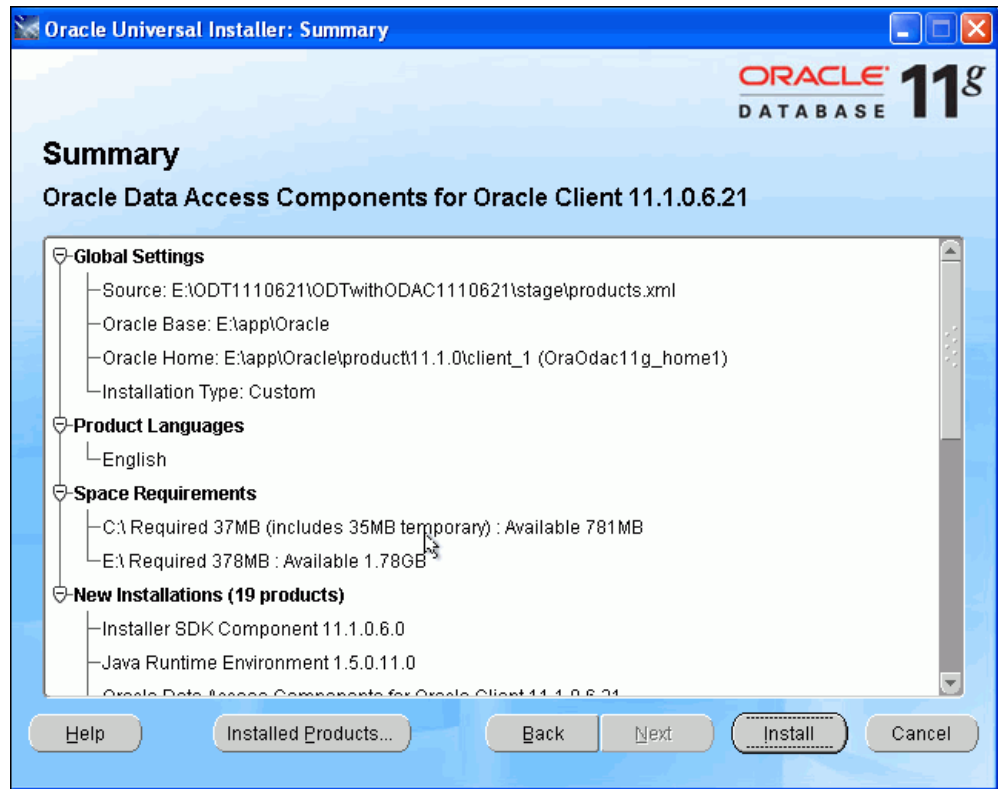


9. Click **Next**.

A screen appears reminding you that you must run the SQL scripts located in `ORACLE_BASE\ORACLE_HOME\client_1\ASP.NET\SQL` if you wish to use Oracle Providers for ASP.NET.

10. Click **Next**.

The Summary window appears.



11. Click **Install** to complete the installation.

The end of the installation screen appears. It reminds you again to install the ASP.NET scripts. Do this if you plan to use the Oracle Providers for ASP.NET.

12. Click **Exit**.

Configuring a NET Connect Alias

The `tnsnames.ora` file defines database server addresses so that the Oracle client can use a short version of the name to connect to databases. Your DBA may have already provided you with an preconfigured `tnsnames.ora` file.

Otherwise, you need to navigate to the `ORACLE_BASE\ORACLE_HOME\network\admin\sample` directory and copy the `tnsnames.ora` and `sqlnet.ora` files located to the `ORACLE_BASE\ORACLE_HOME\network\admin` directory.

You may use the following connect descriptor in your `tnsnames.ora` file and change the values shown in italics for your specific environment:

Example 2-1 `tnsnames.ora` connect descriptor

```
address name =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(Host = hostname)(Port = port))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = sid)
    )
  )
```

Where:

sid: Is the database service name

hostname: Is the database computer name

port: Is the port to use to communicate to the database

address name: Is a user-defined short name for the connect descriptor. This short name will be used in the connection string to your .NET application.

[Example 2-2](#) shows a sample `tnsnames.ora` file.

Example 2-2 Sample `tnsnames.ora` File

```
ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ORCL)
    )
  )
)
```

See Also: *Oracle Net Services Administrator's Guide*

Building a Simple .NET Application Using ODP.NET

This chapter contains:

- [Creating a New Project](#)
- [Adding a Reference](#)
- [Adding Namespace Directives](#)
- [Designing the User Interface](#)
- [Writing the Connection Code](#)
- [Compiling and Running the Application](#)
- [Error Handling](#)

Creating a New Project

Visual Studio groups all development code that you create into containers known as projects. Simpler projects often contain only one file. In this section, you will learn how to create a new development project.

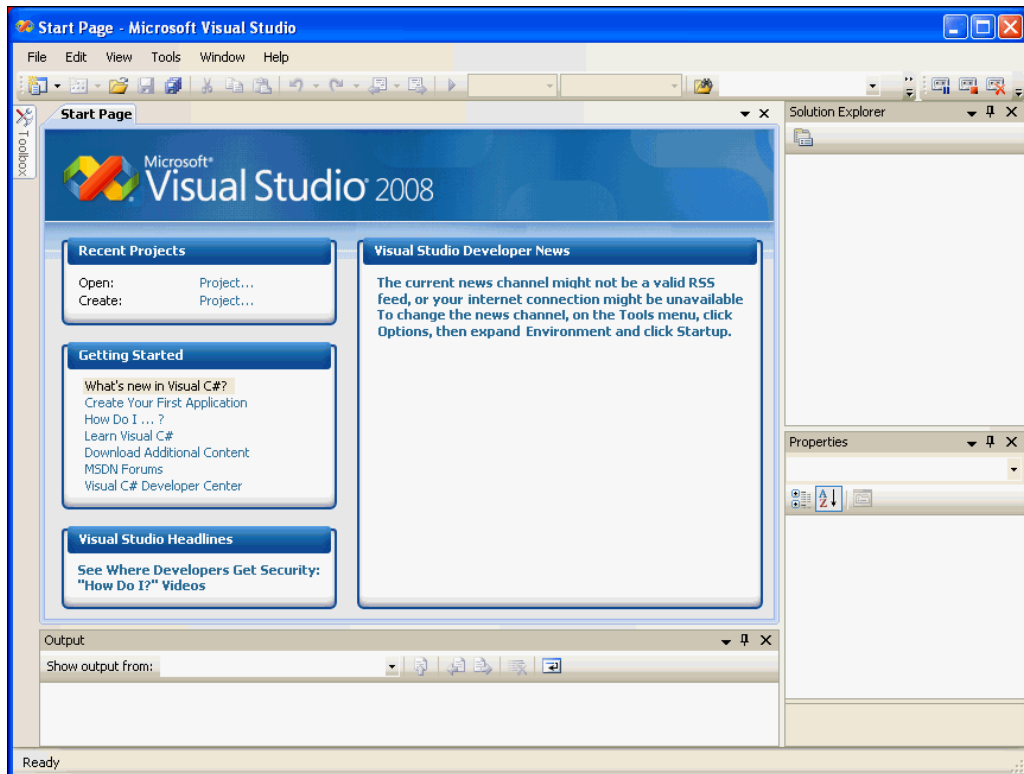
The application you build in this chapter as serves a starting point for work in subsequent chapters, so it is important to follow the order of this guide.

NOTE: When necessary, instructions specify **Visual C#** or **Visual Basic**.

To start a new project:

1. Start Visual Studio.

Open the **Start** menu, select **All Programs**, and then select **Microsoft Visual Studio 2008**.



The Microsoft Visual Studio IDE environment appears.

- In the Start Page, under the Recent Projects heading, click **Create: Project**.

Alternatively, from the **File** menu, select **New**, and then select **Project**.

A **New Project** dialog box appears.

- In Project Types, select the type of project you are creating:

Visual C#:

Visual C#: Windows

Visual Basic:

Other Languages: Visual Basic: Windows

- In Templates, select **Windows Forms Application**.

- In the Name field, enter the appropriate name.

Visual C#:

HR_Connect_CS

Visual Basic:

HR_Connect_VB

The abbreviation CS indicates C# projects and VB indicates Visual Basic projects.

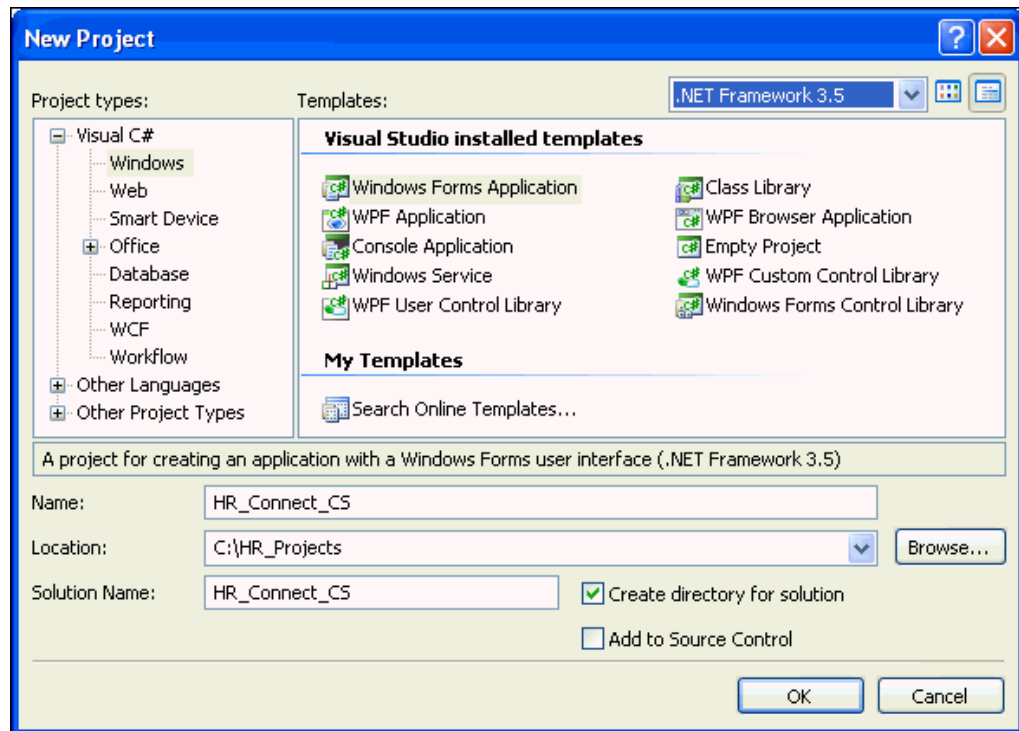
- In Location, enter the directory where you want to save the files.

For this guide, enter this directory `C:\HR_Projects`.

- In Solution Name, the appropriate name, `HR_Connect_CS` or `HR_Connect_VB` should appear.

A solution can contain several projects; when it contains only one project, you can use the same name for both.

8. Check **Create directory for solution**.
9. Click **OK**.

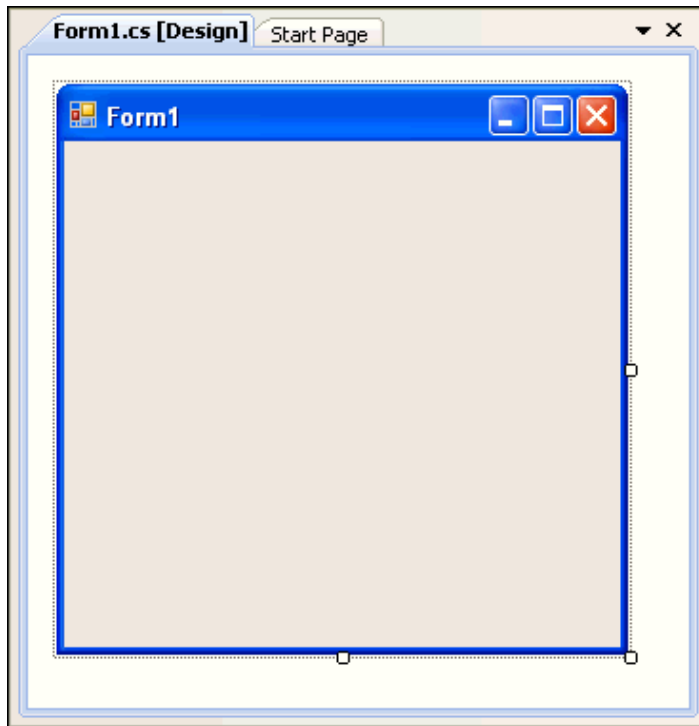


The project is created.

The main window now displays a new title, either `HR_Connect_CS - Microsoft Visual Studio` or `HR_Connect_CS - Microsoft Visual Studio`, depending on the language, and contains `Form1` shown below.

It is important to remember that many projects automatically name the first form `Form1`. This is the name of the form control. Do not confuse this with the actual name given to the code file, which is typically `Form1.cs` or `Form1.vb`.

Both `Form1` and `Form1.xx` can be renamed. For the purposes of this guide, we will rename `Form1.xx` several times.

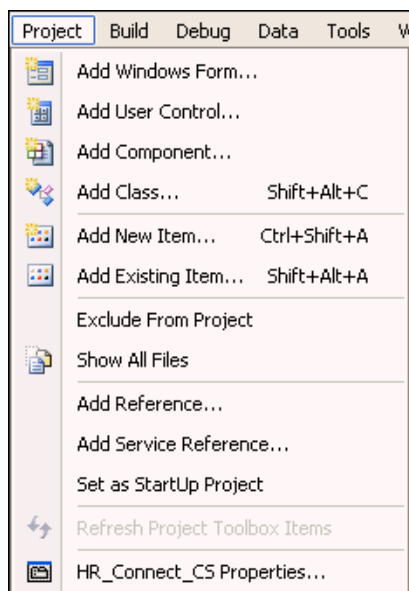


Adding a Reference

This section shows you how to add a reference to the `Oracle.DataAccess.dll` file, which contains the data provider, Oracle Data Provider for .NET.

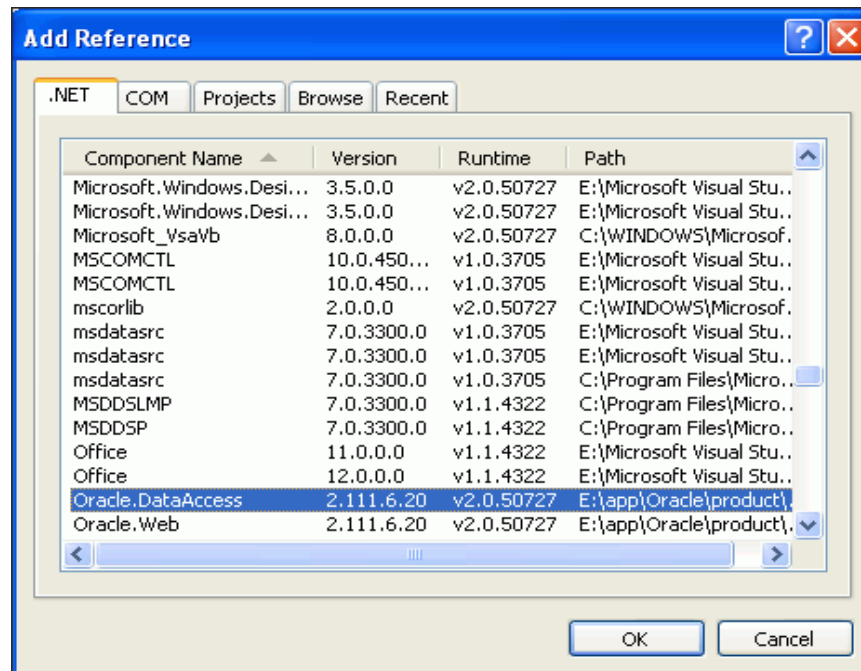
To add a reference:

1. From the **Project** menu, select **Add Reference**.

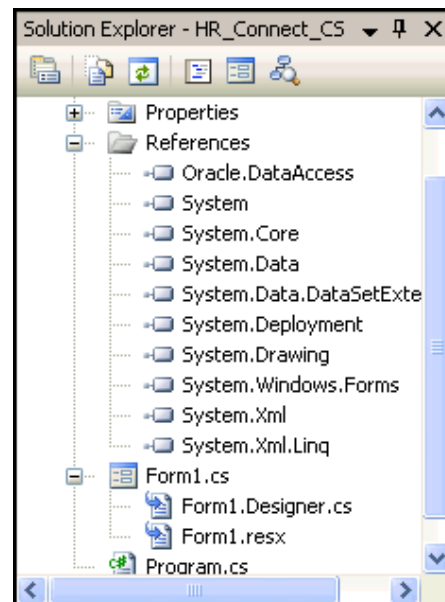


The Add Reference windows appears.

2. In the Add Reference window, under the .NET tab, select **Oracle.DataAccess**. Click **OK**.



Note that the new reference appears in the Solution Explorer.



Adding Namespace Directives

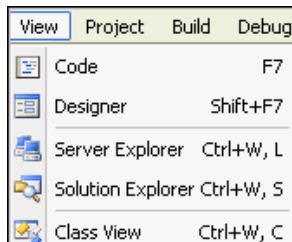
You can add Oracle namespace directives that allow you to indicate an assembly's namespaces within the module. To do this, add C# `using` statements or Visual Basic `Imports` statements, at or near the top of a code file.

Note: Adding a reference makes the namespace available within the application. Adding a namespace directive within the application code makes the namespace more visible and allows for additional scoping.

To add Oracle namespace directives:

1. With Form1 active, from the **View** menu select **Code**.

Alternatively, you can use the **F7** keyboard shortcut.

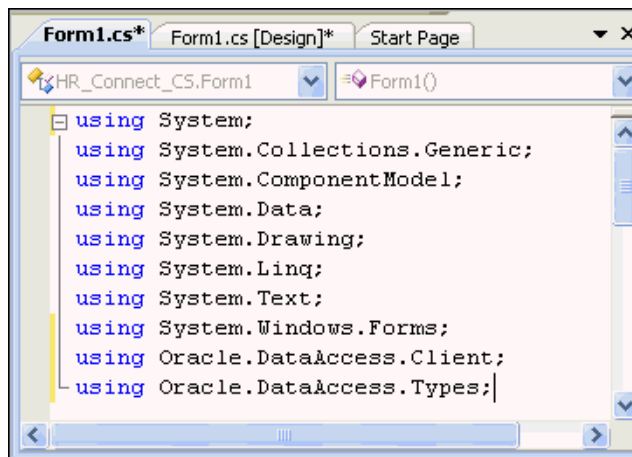


2. Add the following statements to the list of declarations depending on the language you are using.

- **Visual C#:**

Add with other using statements, before the namespace.

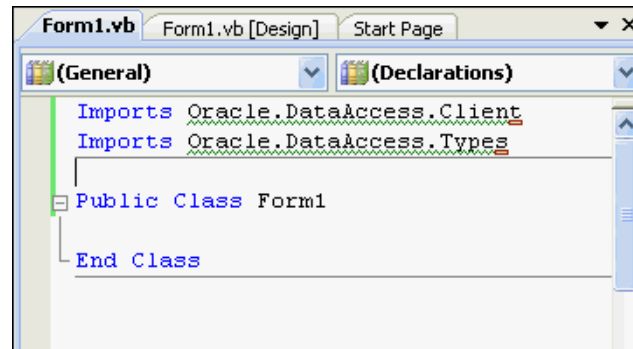
```
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;
```



- **Visual Basic:**

Add to the top of the file, in the declarations section.

```
Imports Oracle.DataAccess.Client
Imports Oracle.DataAccess.Types
```



3. Save the changes by selecting **Save** from the **File** menu, or using the **Ctrl+S** keyboard shortcut.

Designing the User Interface

You can create a user interface by adding the toolbox controls to the design form. This interface accepts connection information from the user.

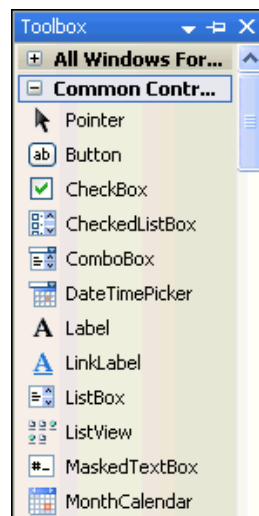
To add toolbox controls:

1. From the **View** menu, select **Designer**.

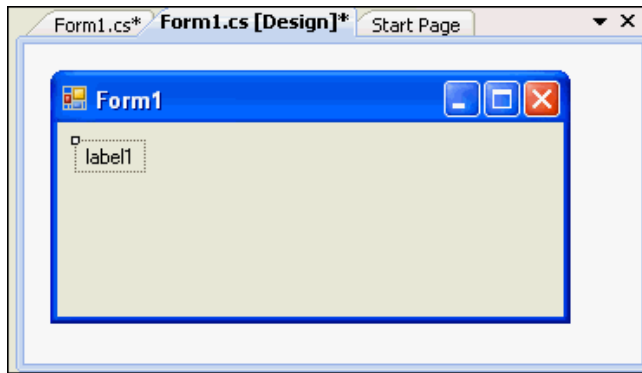
This opens Form1, in design view, if it is not already open.

You will toggle between Code and Designer a lot. The keyboard shortcuts are **F7** and **shift- F7** respectively.

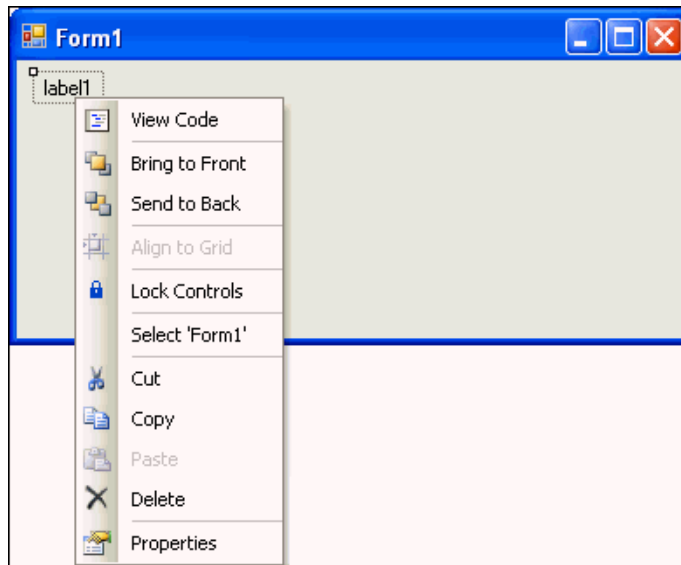
2. From the **View** menu, select **Toolbox**.
3. In the Toolbox, expand **Common Controls**.



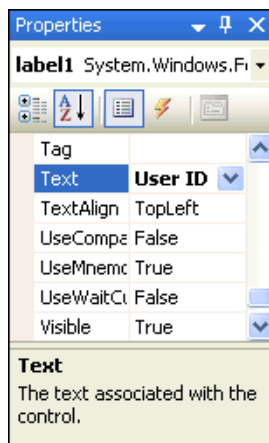
4. In the Toolbox, select **Label**, and drag it onto the Form1.



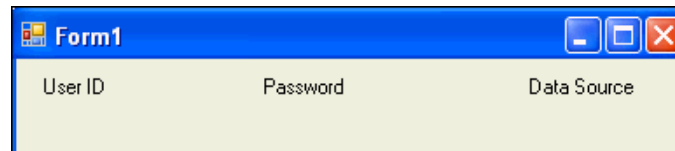
5. On Form1, right-click **label1**.



6. From the menu, select **Properties**, if the Properties Window is not already visible. The Properties Window appears.
7. In the Properties Window, change the Text property from **label1** to **User ID**.



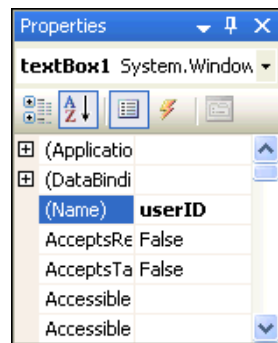
8. Repeat steps 4 through 7 twice, placing two more labels on Form 1 and changing their text properties to **Password** and **Data Source**.



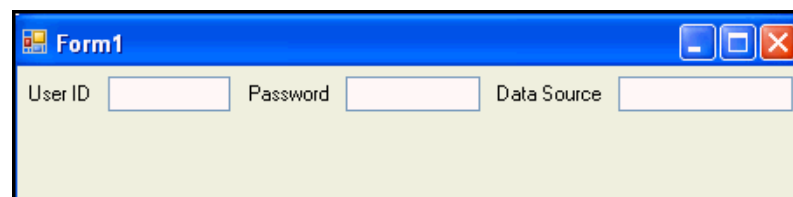
- In the Toolbox, select **TextBox**, and drag it onto the Form1, next to the User ID label.



- In the Properties Window, change the Name property to **userID**.

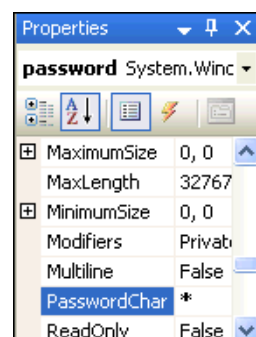


- Repeat steps 9 and 10 twice, positioning two more text boxes next to the existing labels, and setting the Name property to **password** and **dataSource**.



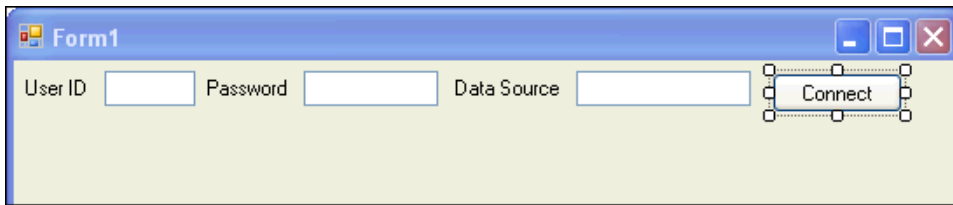
- Select the text box next to the Password label. In the Properties Window, scroll to the PasswordChar property and set it to an asterisk (*).

This masks the password during entry.



- From the Toolbox, select **Button** and drag it onto Form1.

In the Properties Window, change the Text property of the button from **button1** to **Connect**, and change the Name property to **connect**.



- Save.
- Close the Toolbox.

Writing the Connection Code

Now we write the code that takes the information provided to the user interface and connects to the database.

To connect to the database, you must create a connection object.

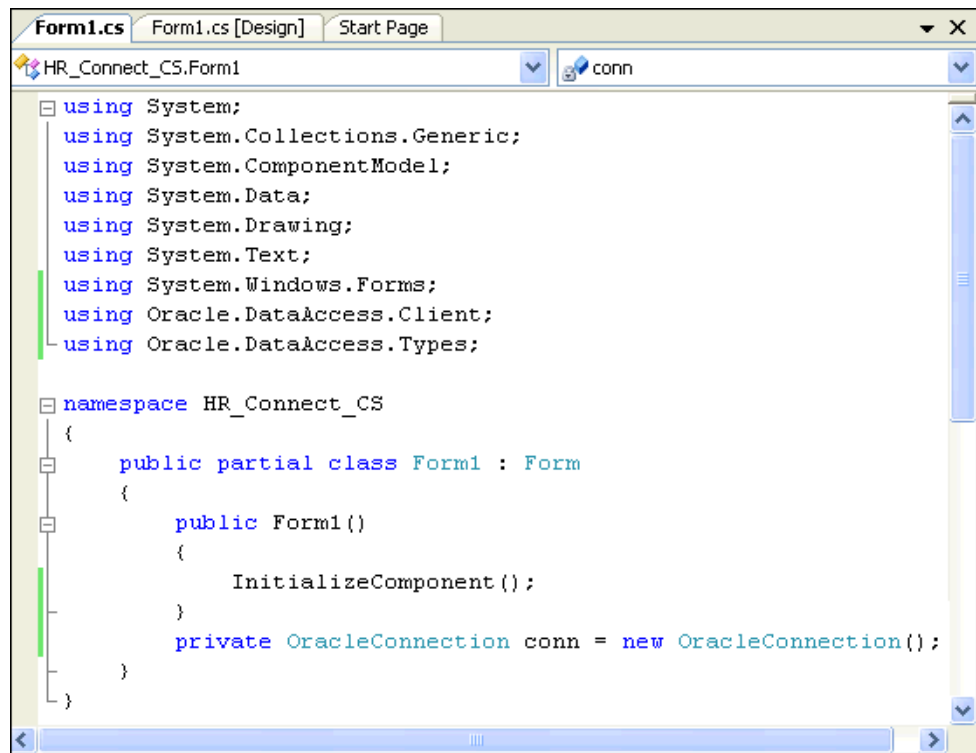
To write code that connects to the database:

These steps enable your application to connect to the database based on data that the user enters into the Form1 control. See ["Compiling and Running the Application"](#) on page 3-13.

- From the **View** menu, select **Code**.
- Add the code indicated to instantiate a database connection string.

Visual C#: Add the class variable `conn` to the Form1 class right after the `public Form1()` block with this code.

```
private OracleConnection conn = new OracleConnection();
```



```

Form1.cs Form1.cs [Design] Start Page
HR_Connect_CS.Form1 conn
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;

namespace HR_Connect_CS
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private OracleConnection conn = new OracleConnection();
    }
}

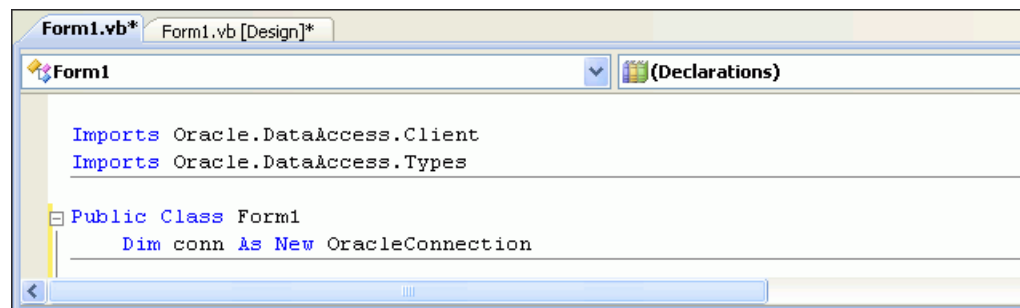
```

Visual Basic: Add the `conn` class variable in the `Form1` class declaration, using this code.

```

Public Class Form1
    Dim conn As New OracleConnection

```



```

Form1.vb* Form1.vb [Design]*
Form1 (Declarations)
Imports Oracle.DataAccess.Client
Imports Oracle.DataAccess.Types

Public Class Form1
    Dim conn As New OracleConnection

```

3. Save your changes.
4. Change to Designer view by clicking on the **View** menu and selecting **Designer**.
5. Double-click the **Connect** button on `Form1` to open the code window to the `connect_Click()` method.

Insert the code indicated into the `connect_Click()` method.

Visual C#:

```

conn.ConnectionString = "User Id=" + userID.Text +
    ";Password=" + password.Text +
    ";Data Source=" + dataSource.Text + ";";

```

```
conn.Open();
```

Visual Basic:

```
conn.ConnectionString = "User Id=" + userID.Text & _  
    ";Password=" + password.Text & _  
    ";Data Source=" + dataSource.Text  
conn.Open()
```

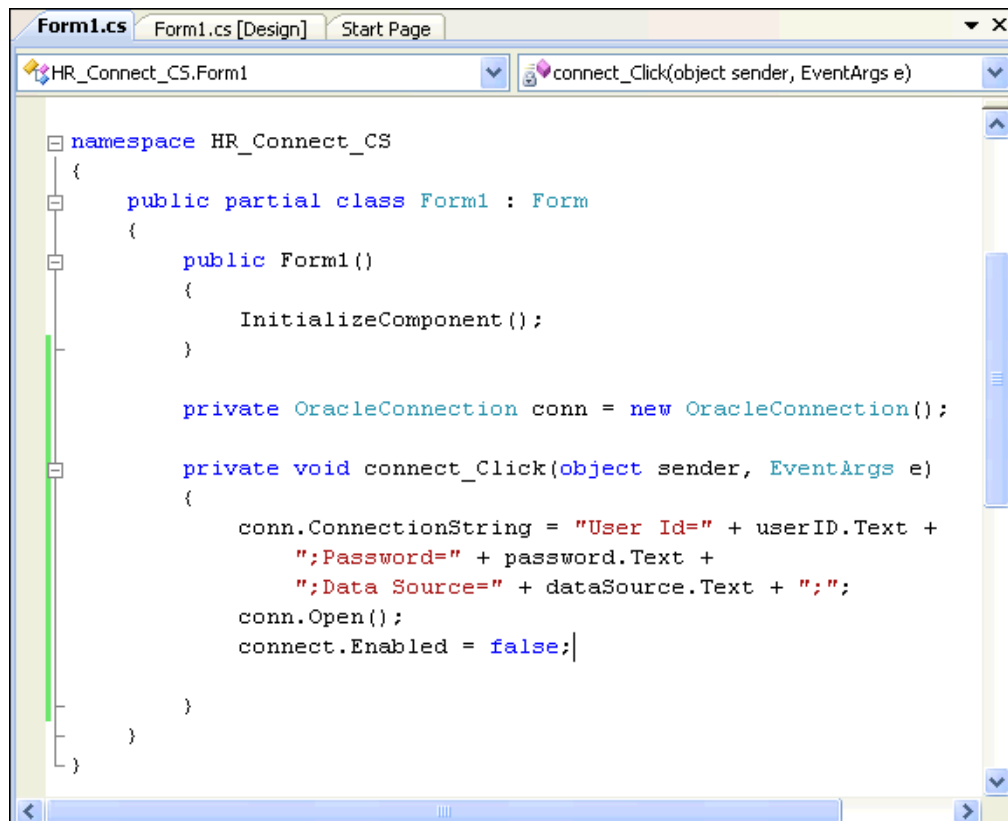
Note: Before a connection can be opened, it must be built from user input for the User ID, Password, and Data Source. The Open() method makes the actual connection.

6. Set the Enabled attribute of the button to false by inserting the indicated code at the end of the connect_Click() method.

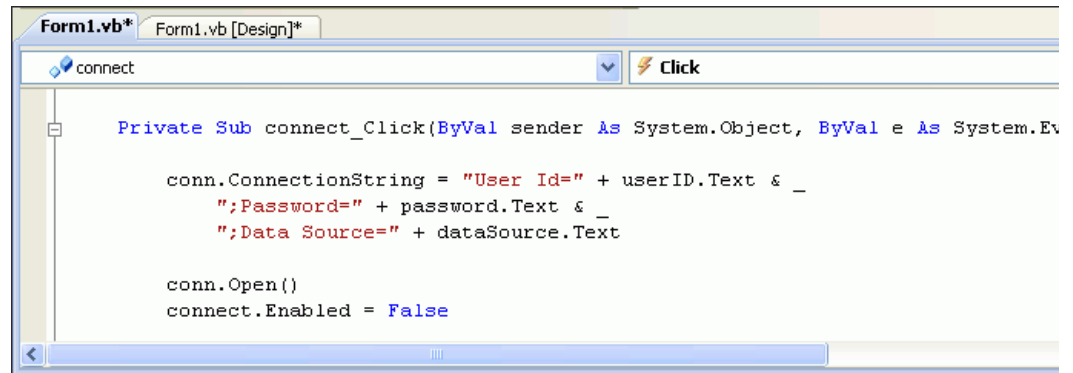
This disables the Connect button, which is a good practice once a connection is successfully made.

Visual C#:

```
connect.Enabled = false;
```

**Visual Basic:**

```
connect.Enabled = false
```



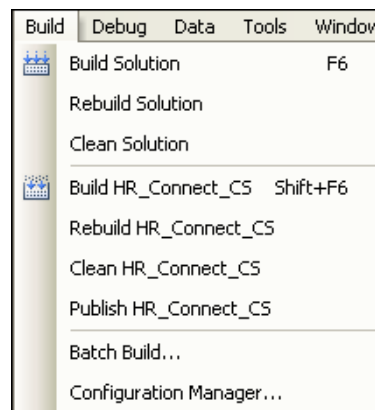
You have now finished writing an application that can connect to the Oracle database. The following sections show how to use it.

Compiling and Running the Application

This section shows how to compile and run the application you created in the previous sections.

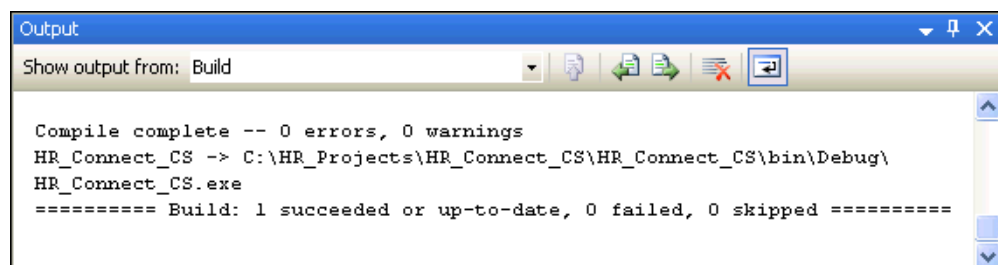
To compile and run the application:

1. From the **Build** menu, select **Build Solution**.

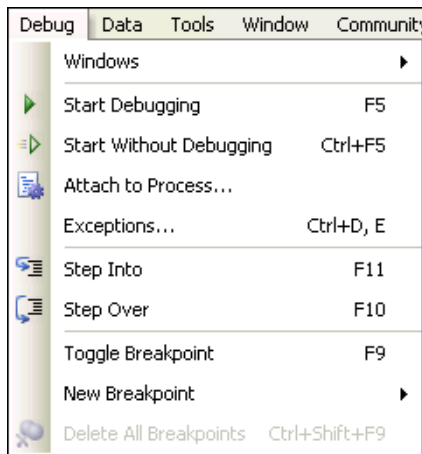


2. Ensure that there are no errors reported in the output window, available from the **View** menu.

The following graphics shows a typical output result.



3. If there are any errors indicated, from the **View** menu, select **Error List** and fix the errors.
4. From the **Debug** menu, select **Start Without Debugging** to run the application.



5. In the Form1 application, enter the User ID, Password, and Data Source. Click **Connect**.

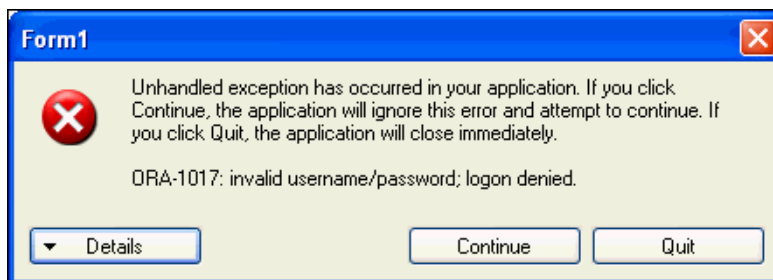
This is where the application makes use of the `tnsnames.ora` file. See ["Configuring a NET Connect Alias"](#) on page 2-7.

Once the connection is opened, the Connect button is disabled. You have succeeded in implementing a connection to an Oracle Database instance.



Error Handling

Applications must be able to handle run-time errors gracefully. For example, if you try to log in using an incorrect password, the application you developed so far cannot establish a connection to the database, and exits with the following unhandled exception error, `ORA-1017: invalid username/password, logon denied`.



You must reselect **Start Without Debugging** to try this with a different password.

Error handling manages occurrences of conditions that change the normal flow of program execution. Oracle Data Provider for .NET contains three classes for error handling and support:

- The `OracleError` class represents a warning or an error reported by Oracle.
- An `OracleErrorCollection` class represents a collection of all errors that are thrown by the Oracle Data Provider for .NET. It is a simple `ArrayList` that holds a list of `OracleErrors`.
- The `OracleException` class represents an exception that is thrown when the Oracle Data Provider for .NET encounters an error. Each `OracleException` object contains at least one `OracleError` object in the `Error` property that describes the error or warning.

Using Try-Catch-Finally Block Structure

.NET languages use Try-Catch-Finally block structure for error handling. With this structure, the Try code is the main code, the goal that the application wants to accomplish. The Catch code catches errors of various types, as shown in the next two sections. The Finally block comes last and always executes.

The Finally block frequently contains the `Dispose` method, which closes and disposes of the connection. Having the `Dispose` method in the Finally block ensures that the database connection is always closed after the Try-Catch-Finally block completes. Closing database connections after the application no longer requires database access is important for many reasons, especially data security.

Attempting to close a closed database connection does not cause an error. The attempt is irrelevant. Nonetheless, placing `Dispose()` in the Finally code block guarantees that the connection is closed.

The next section shows how to use Try-Catch-Finally block structure with general errors, and the section after that, with Oracle errors.

Handling General Errors

This section shows how to handle general errors using a Try-Catch-Finally block.

To handle general errors:

1. Change the code of the `connect_Click()` method in `Form1` by adding an implementation of the Try-Catch-Finally syntax.

New code is in bold font.

Visual C#:

```
private void connect_Click(object sender, EventArgs e)
{
    conn.ConnectionString = "Data Source=ORCL;User Id="
        + userID.Text + ";Password=" + password.Text + ";";
    try
    {
        conn.Open();
        connect.Enabled = false;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message.ToString());
    }
    finally

```

```

    {
        conn.Dispose();
    }
}

```

Alternatively, you can use C# syntax that disposes of a connection when it goes out of scope, with the `using` keyword, as follows:

```

using (OracleConnection conn = new OracleConnection())
{
    conn.Open();
    // application code
    ...
}

```

Visual Basic:

```

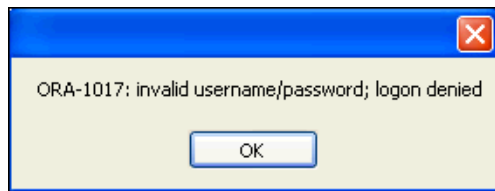
Try
    conn.Open()
    connect.Enabled = false

Catch ex As Exception
    MessageBox.Show(ex.Message.ToString())

Finally
    conn.Dispose()
End Try

```

- From the **Build** menu, select **Rebuild Solution**.
Ensure that there are no errors.
- From the **Debug** menu, select **Start Without Debugging**.
- Run the application again, as described in section "[Compiling and Running the Application](#)" on page 3-13, and attempt to connect using an incorrect password. This time, the application catches the error and displays it in a pop-up window, `ORA-1017: invalid username/password; logon denied`.



Handling Common Oracle Errors

In the completed Try-Catch-Finally block code shown below, the first Catch statement branch is skipped if there are no `OracleExceptions`. The second Catch statement branch catches all other `Exceptions`.

The first catch statement contains `Case` statements, which can be used to trap common database errors and display them in a user-friendly manner.

Note that the second `Case` statement catches a specific example of `OracleException`, when the database is not accessible.

To handle specific errors:

- Stop the database instance. See [Appendix A, "Starting and Stopping an Oracle Database Instance"](#).

2. Add the Catch OracleException block shown below in bold, before the Catch Exception block previously added in the connect_Click() method.

Visual C#:

```

try
{
    conn.Open();
    connect.Enabled = false;
}
catch (OracleException ex)
{
    switch (ex.Number)
    {
        case 1:
            MessageBox.Show("Error attempting to insert duplicate data.");
            break;
        case 12560:
            MessageBox.Show("The database is unavailable.");
            break;
        default:
            MessageBox.Show("Database error: " + ex.Message.ToString());
            break;
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message.ToString());
}
finally
{
    conn.Dispose();
}
}

```

Visual Basic:

```

Try
    conn.Open()
    connect.Enabled = false

Catch ex As OracleException ' catches only Oracle errors
    Select Case ex.Number
        Case 1
            MessageBox.Show("Error attempting to insert duplicate data.")
        Case 12560
            MessageBox.Show("The database is unavailable.")
        Case Else
            MessageBox.Show("Database error: " + ex.Message.ToString())
    End Select

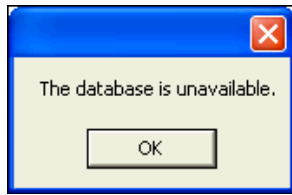
Catch ex As Exception
    MessageBox.Show(ex.Message.ToString())

Finally
    conn.Dispose()
End Try

```

3. Compile and run the application again, as described in section ["Compiling and Running the Application"](#) on page 3-13.

Note that the ORA-12560 error appears in the pop-up window as The database is unavailable with no error number provided.



4. Restart the database instance. See [Appendix A, "Starting and Stopping an Oracle Database Instance"](#).

Retrieving and Updating with Oracle Data Provider for .NET

This chapter contains:

- [Using the Command Object](#)
- [Retrieving Data: a Simple Query](#)
- [Retrieving Data: Bind Variables](#)
- [Retrieving Data: Multiple Values](#)
- [Using the DataSet Class with Oracle Data Provider for .NET](#)
- [Enabling Updates to the Database](#)
- [Inserting, Deleting, and Updating Data](#)

Using the Command Object

To view, edit, insert or delete data in a database, you must encapsulate a request in an `OracleCommand` object specifying a SQL command, stored procedure, or table name. The `OracleCommand` object creates the request, sends it to the database, and returns the result.

To use the command object:

1. Make two copies of `Form1 .xx`, from application `HR_Connect_xx` in [Chapter 3, "Building a Simple .NET Application Using ODP.NET"](#). To make copies, see the instructions in [Appendix B, "Copying a Form"](#).

Name the copies `Form2 .cs` or `Form2 .vb` and `Form3 .cs` or `Form3 .vb`. The first copy is for the first part of the chapter, and the second copy for the second part of the chapter

2. Open `Form2 .cs` or `Form2 .vb`.

Note that the actual form in the designer still says `Form1`, as you renamed code files but not the actual form controls within the project.

3. Create a string that represents the SQL query and add to the body of the `try` statement.

The new code is in bold typeface.

Visual C#:

```
try  
{
```

```

conn.Open();
connect.Enabled = false;

// SQL Statement
string sql = "select department_name from departments"
    + " where department_id = 10";
}

```

Visual Basic:

```

Try
    conn.Open()
    connect.Enabled = False

    Dim sql As String = "select department_name from departments" & _
        "where department_id = 10"

```

4. Use the new `sql` variable to create the `OracleCommand` object, and set the `CommandType` property to run a text command.

Visual C#:

```

try
{
    conn.Open();
    connect.Enabled = false;

    // SQL Statement
    string sql = "select department_name from departments"
        + " where department_id = 10";

    OracleCommand cmd = new OracleCommand(sql, conn);
    cmd.CommandType = CommandType.Text;
}

```

Visual Basic:

```

Try
    conn.Open()
    connect.Enabled = False

    Dim sql As String = "select department_name from departments" & _
        "where department_id = 10"

    Dim cmd As New OracleCommand(sql, conn)
    cmd.CommandType = CommandType.Text

```

5. Save your work.

Retrieving Data: a Simple Query

This section demonstrates retrieving data from the database.

The `ExecuteReader()` method of an `OracleCommand` object returns an `OracleDataReader` object, which can be accessed to display the result on the form. The application uses a `ListBox` to display the results.

To retrieve data:

1. Create an `OracleDataReader` object, by adding the code indicated to the bottom of the `Try` block of the `connect_Click()` method.

This enables you to read the result of the query.

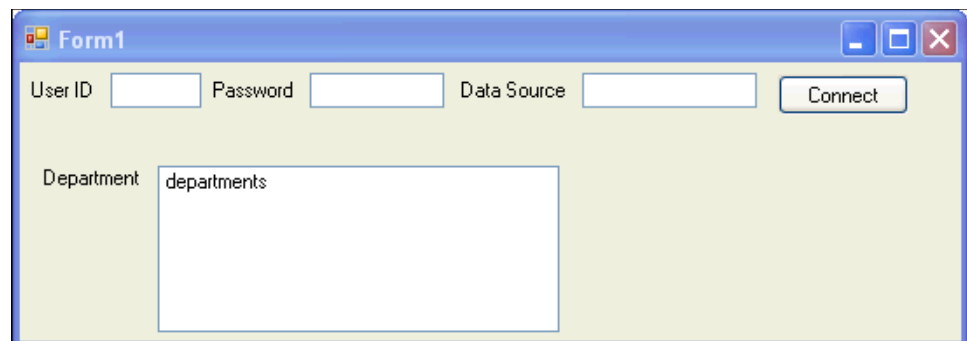
Visual C#:

```
OracleDataReader dr = cmd.ExecuteReader();
dr.Read();
```

Visual Basic:

```
Dim dr As OracleDataReader = cmd.ExecuteReader()
dr.Read()
```

2. Open `Form1` in Design view. From the **View** menu, select **Designer**.
3. From the **View** menu, select **Toolbox**.
4. From the Toolbox, select a **Label** and drag it onto `Form1`.
5. From the **View** menu, select **Properties Window**.
6. In the Properties window, change the **Text** of the label to `Department`.
7. From the Toolbox, under Window forms, select a **ListBox** and drag it onto `Form1`.
8. In the Properties window, under Design, change the **Name** to `departments`.



9. Add accessor type methods for retrieving data from the query result. Double-click the connect button to edit the `connect_click()` method, and add the code indicated to the bottom of the `try` block.

Visual C#:

```
departments.Items.Add(dr.GetString(0));
```

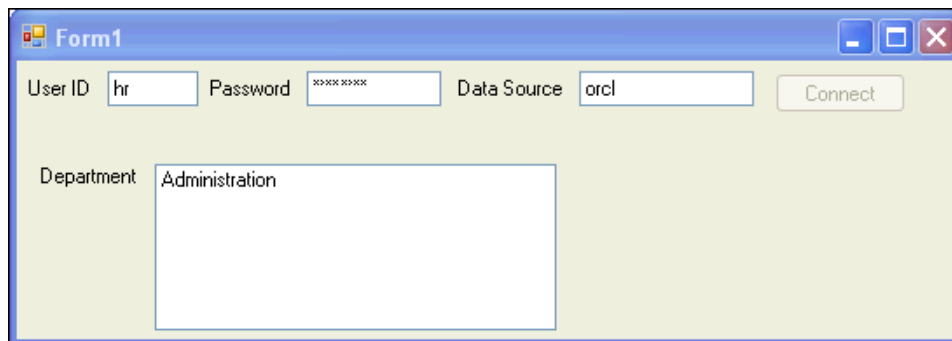
Visual Basic:

```
departments.Items.Add(dr.GetString(0))
```

Typed accessors, such as `GetString`, return native .NET data types and native Oracle data types. Zero-based ordinals passed to the accessors specify which column in the result set to return.

10. Build and save the application.
11. Run the application. Enter the login and data source.

After you connect, the departments list box shows Administration, the correct name for department number 10 in the HR schema, as requested by the SELECT statement.



Retrieving Data: Bind Variables

Bind variables are placeholders inside a SQL statement. When a database receives a SQL statement, it determines if the statement has already been executed and stored in memory. If the statement does exist in memory, Oracle Database can reuse it and skip the task of parsing and optimizing the statement. Using bind variables makes the statement reusable with different input values. Using bind variables also improves query performance in the database, eliminates the need for special handling of literal quotation marks in the input, and protects against SQL injection attacks.

The following code shows a typical SELECT statement that does not use bind variables, with the value 10 specified in the WHERE clause of the statement.

```
SELECT department_name FROM departments WHERE department_id = 10
```

The following code replaces the numerical value with a bind variable :department_id. A bind variable identifier always begins with a single colon (:).

```
SELECT department_name FROM departments WHERE department_id = :department_id
```

Note that bind variables can also be used with UPDATE, INSERT, and DELETE statements, and also with stored procedures. The following code illustrates how to use bind variables in an UPDATE statement:

```
UPDATE departments SET department_name = :department_name
WHERE department_id = : department_id
```

See "[Inserting, Deleting, and Updating Data](#)" on page 4-12 for more details.

You can use the `OracleParameter` class to represent each bind variable in your .NET code. The `OracleParameterCollection` class contains the `OracleParameter` objects associated with the `OracleCommand` object for each statement. The `OracleCommand` class passes your SQL statement to the database and returns the results to your application.

You can bind variables by position or by name by setting the `OracleCommand` property `BindByName` (which defaults to `false`).

- Binding by position

You must use the `Add()` method to add the parameters to the `OracleParameterCollection` in the same order as they appear in the SQL statement or stored procedure.

- Bind by name

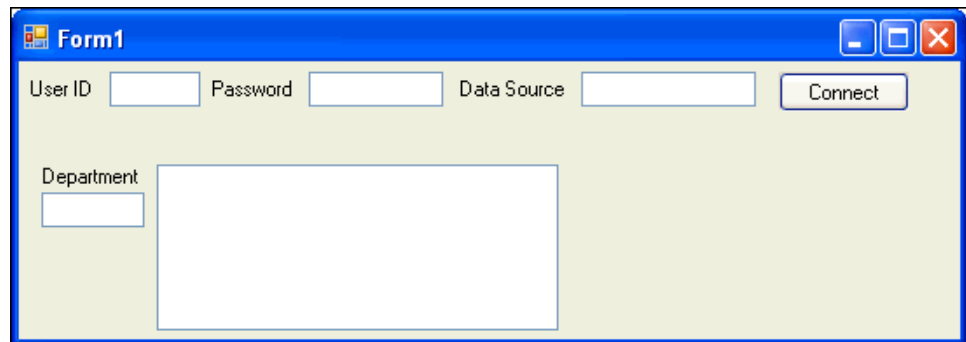
You may add the parameters to the collection in any order; however, you must set the `ParameterName` property for the parameter object to the same name as the bind variable identifier in the stored procedure declaration.

In addition to the binding mode (by position or by name), the .NET developer sets the following properties for each parameter object: `Direction`, `OracleDbType`, `Size`, and `Value`.

- **Direction** Bind variables may be used as output, input, or input/output parameters. The `Direction` property indicates the direction of each parameter. The default value of the `Direction` property is `Input`.
- **OracleDbType** property indicates whether or not the parameter is a number, a date, a `VARCHAR2`, and so on.
- **Size** indicates the maximum data size that parameters with a variable length data type, such as `VARCHAR2`, can hold.
- **Value** contains the parameter value, either before statement execution (for input parameters), after execution (for output parameters), or both before and after (for input/output parameters).

To retrieve data using bind variables:

1. Move the `ListBox` named `Departments` to the right.
2. From the **View** menu, select **Toolbox**.
3. From the **Toolbox**, select a **TextBox** and drag it onto `Form1`, under the label that says `Department`.
4. From the **View** menu, select **Properties Window**.
5. In the `Properties` window, change **Name** to `departmentID`.



6. Change the `SELECT` statement to use the bind variable by adding the code indicated to the `Try` block of the `connect_Click()` method.

Changed or new code is in bold typeface.

Visual C#:

```
string sql = "select department_name from departments where department_id = " +
    "":department_id"";
OracleCommand cmd = new OracleCommand(sql, conn);
cmd.CommandType = CommandType.Text;
OracleParameter p_department_id = new OracleParameter();
p_department_id.OracleDbType = OracleDbType.Decimal;
p_department_id.Value = departmentID.Text;
cmd.Parameters.Add(p_department_id);
```

```
OracleDataReader dr = cmd.ExecuteReader();
dr.Read();
```

```
departments.Items.Add(dr.GetString(0));
```

Visual Basic:

```
Dim sql As String = "select department_name from departments where" & _
    "department_id= :department_id"
```

```
Dim cmd As OracleCommand = New OracleCommand(sql, conn)
cmd.CommandType = CommandType.Text
```

```
Dim p_department_id as OracleParameter = new OracleParameter()
```

```
p_department_id.OracleDbType = OracleDbType.Decimal
```

```
p_department_id.Value = departmentID.Text
```

```
cmd.Parameters.Add(p_department_id)
```

```
Dim dr As OracleDataReader = cmd.ExecuteReader()
```

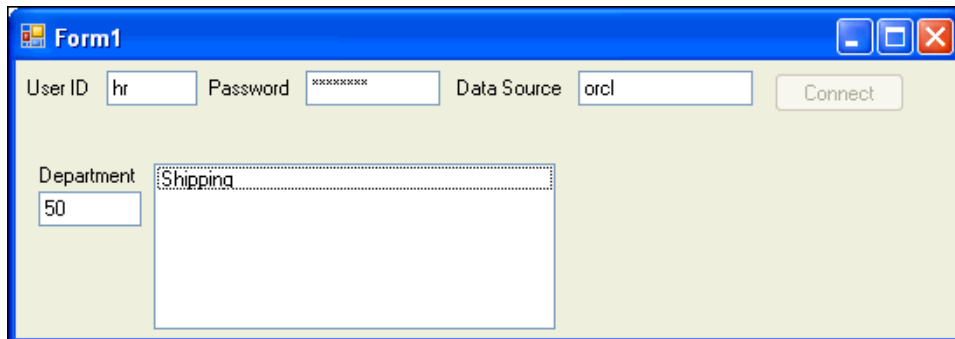
```
dr.Read()
```

```
departments.Items.Add(dr.GetString(0))
```

For this code, the parameter object sets the `OracleDbType` property, but there is no need to set the `Direction` property because it uses the default value, `Input`. There is no need to set the `Size` property because the object is an input parameter, and the data provider can determine the size from the value.

7. Save and run the application.
8. Enter the login information, and a typical department number, such as 50, from the HR schema.
9. Click **Connect**.

The application returns the name of the department that corresponds to the department ID.



Retrieving Data: Multiple Values

You frequently need to retrieve more than just one value from the database. A `DataReader` object can retrieve values for multiple columns and multiple rows. Consider the multiple column, multiple row query in the following example:

```
SELECT department_id, department_name, manager_id, location_id
FROM departments
WHERE department_id < 100
```

Processing multiple rows from the `DataReader` object requires a looping construct. Also, a control that can display multiple rows is useful. Because the `OracleDataReader` object is a forward-only, read-only cursor, it cannot be bound to an updatable or backward scrollable control such as Windows Forms `DataGrid` control. An `OracleDataReader` object is, however, compatible with a `ListBox` control.

To retrieve multiple values:

1. In the `try` block of the `connect_Click()` method, change the SQL query to return a multiple row result set and add a `while` loop to enclose the read method that displays the department names.

Visual C#:

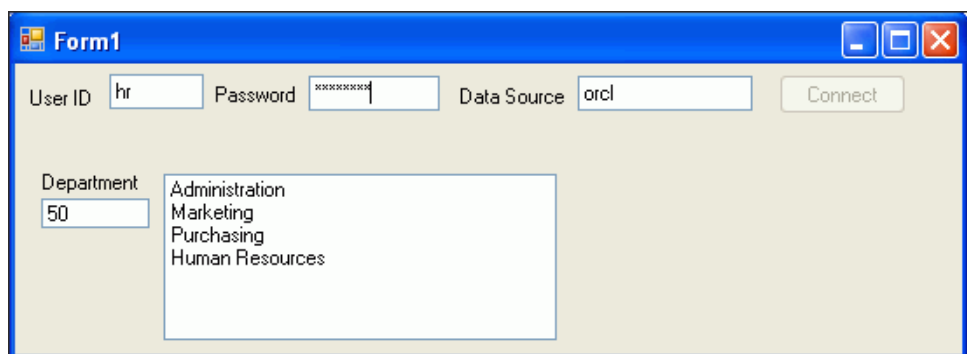
```
try
{
    ...
    string sql = "select department_name from departments where department_id" +
        "< :department_id";
    ...
    while (dr.Read())
    {
        departments.Items.Add(dr.GetString(0));
    }
}
```

Visual Basic:

```
Try
    ...
    Dim sql As String = "select department_name from departments " & _
        "where department_id < :department_id"
    ...
    While (dr.Read())
        departments.Items.Add(dr.GetString(0))
    End While
```

2. Save and run the application.
3. Enter the login information and enter 50 for the department.
4. Click **Connect**.

The application returns the name of the departments that correspond to the query.

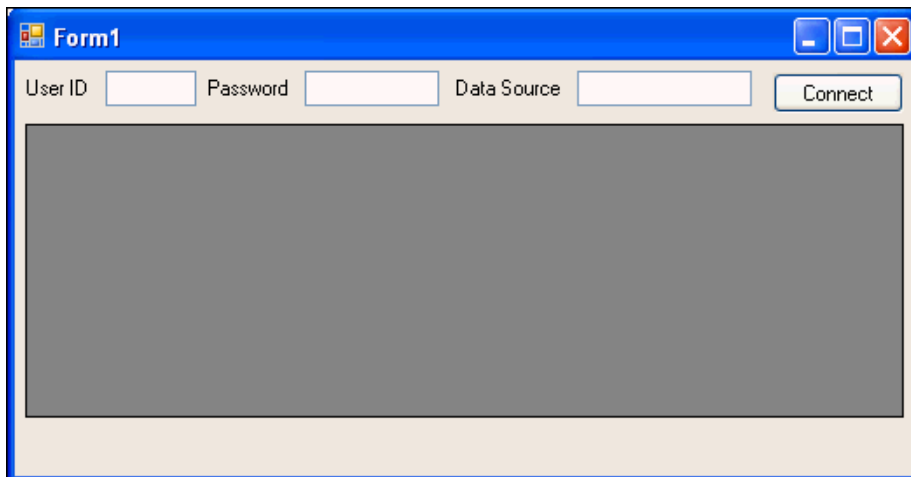


Using the DataSet Class with Oracle Data Provider for .NET

The `DataSet` class provides a memory-resident copy of database data. It consists of one or more tables that store relational or XML data. Unlike an `OracleDataReader` object, a `DataSet` is updatable and backward scrollable.

To use the `DataSet` class:

1. If you have not done so before, make another copy of the `Form1` that you completed in Chapter 3, and name it `Form3.vb` or `.cs`, as described in [Appendix B, "Copying a Form"](#). If `Form1.xx` does not appear in the Solution Explorer, from the **Project** menu, select **Show All Files**.
2. From the **View** menu, select **Designer** view.
3. From the **View** menu, select **Toolbox**.
4. From the **Toolbox**, select a **DataGridView** and drag it onto `Form1`.
5. From the **View** menu, select **Properties Window**.
6. In the **Properties** window, change the **Name** of the data grid view to `departments`.



7. From the **View** menu, select **Code**.
8. Immediately after the `conn` declaration in the code, add variable declarations to the class variables, as indicated.

Visual C#:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private OracleConnection conn = new OracleConnection();
    private OracleCommand cmd;
    private OracleDataAdapter da;
    private OracleCommandBuilder cb;
    private DataSet ds;
    ...
}
```

Visual Basic:

```

Public Class Form1
    Dim conn As New OracleConnection
    Private cmd As OracleCommand
    Private da As OracleDataAdapter
    Private cb As OracleCommandBuilder
    Private ds As DataSet

```

9. Within the `connect_Click()` method try block, add code to:

- Query the database
- Fill the `DataSet` with the result of the command query
- Bind the `DataSet` to the data grid (departments)

Visual C#:

```

conn.Open();
connect.Enabled = false;

string sql = "select * from departments where department_id < 60";
cmd = new OracleCommand(sql, conn);
cmd.CommandType = CommandType.Text;

da = new OracleDataAdapter(cmd);
cb = new OracleCommandBuilder(da);
ds = new DataSet();

da.Fill(ds);

departments.DataSource = ds.Tables[0];

```

Visual Basic:

```

conn.Open()
connect.Enabled = False

Dim sql As String = "select * from departments where department_id < 60"
cmd = New OracleCommand(sql, conn)
cmd.CommandType = CommandType.Text

da = New OracleDataAdapter(cmd)
cb = New OracleCommandBuilder(da)
ds = New DataSet()

da.Fill(ds)

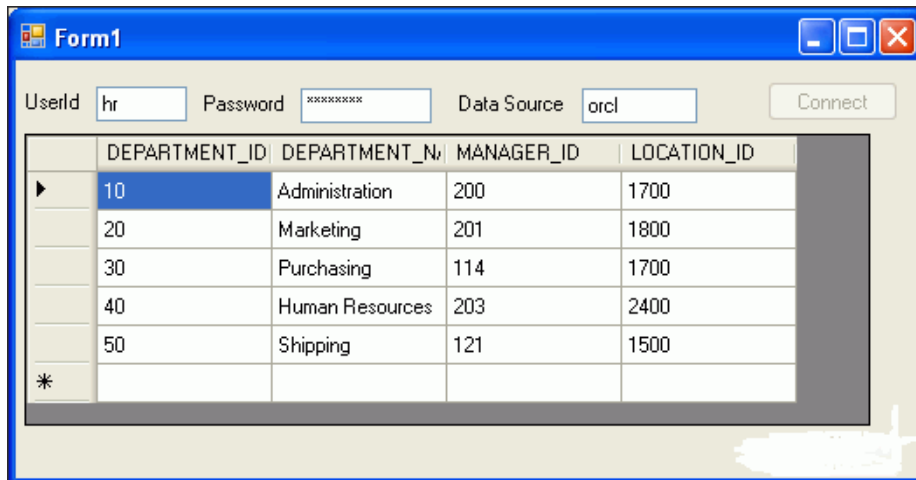
departments.DataSource = ds.Tables(0)

```

10. Build and save the application.

11. Run the application, entering the login and data source.

After you successfully connect to the database, the data grid is populated with the results of the query.

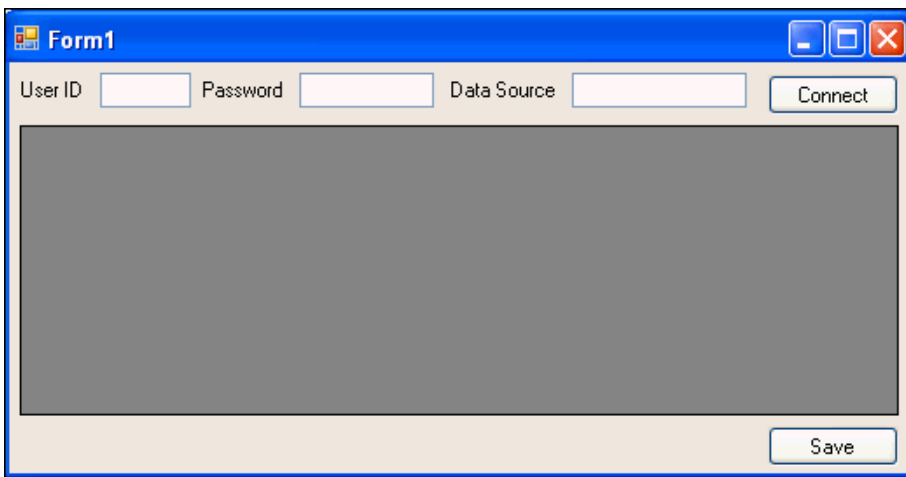


Enabling Updates to the Database

At this point, the DataSet contains a client copy of the database data. In this section, you will add a button that enables client data changes to be saved back to the database. The following section will show you how to test updating, inserting, and deleting the data.

To enable saving data from the DataSet to the database:

1. From the Toolbox, drag and drop a **Button** onto Form1.
2. In the **Properties** window, change the **Name** of the button to save.
Change the **Text** property to Save.
3. At the top of the **Properties** Window, click **Events** (the lightning bolt). In the list of events, select the click event. In the second column, enter the event name, `save_Click`.



4. From the **View** menu, select **Code**.
5. Add code that updates the data, to the body of the `save_Click()` method, as indicated.

Visual C#:

```
da.Update(ds.Tables[0]);
```

Visual Basic:

```
da.Update(ds.Tables(0))
```

You may see some errors show up in the Error List. These will disappear after you add the code in the next step.

6. Within the `Form()` method or `Form1_Load` method, add the code indicated.

Visual C#:

```
public Form1()
{
    InitializeComponent();
    save.Enabled = false;
}
```

Visual Basic:

```
Private Sub Form1_Load(ByVal sender As System.Object, & _
    ByVal e As System.EventArgs) Handles MyBase.Load
    save.Enabled = false
```

7. Within the `connect_Click()` method try block, add code to enable the **Save** button as indicated:

Visual C#:

```
conn.Open();
...
departments.DataSource = ds.Tables[0];

save.Enabled = true;
```

Visual Basic:

```
conn.Open()
...
departments.DataSource = ds.Tables(0)

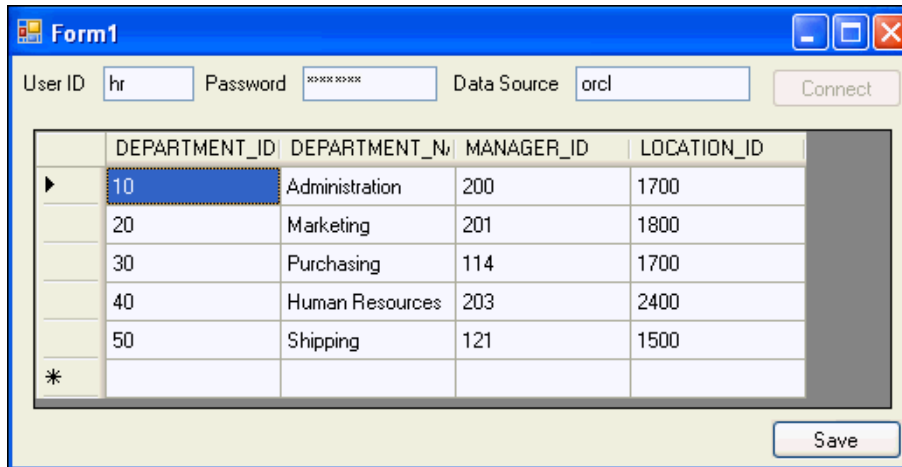
save.Enabled = True
```

8. Remove the `conn.Dispose()` call from the `finally` block in the `connect_Click()` method.

Note: In the previous code used in this example, this method was necessary to dispose or close the connection. However, with these changes to the code, it is necessary to keep the connection open after the query result returns, so that data changes made by the end user are propagated to the database. A general override call, `components.Dispose()`, is already part of the definition of `Form1`.

9. Build and save the application.
10. Run the application, entering the login and data source.

After you successfully connect to the database, the data grid is populated with the results of the query.

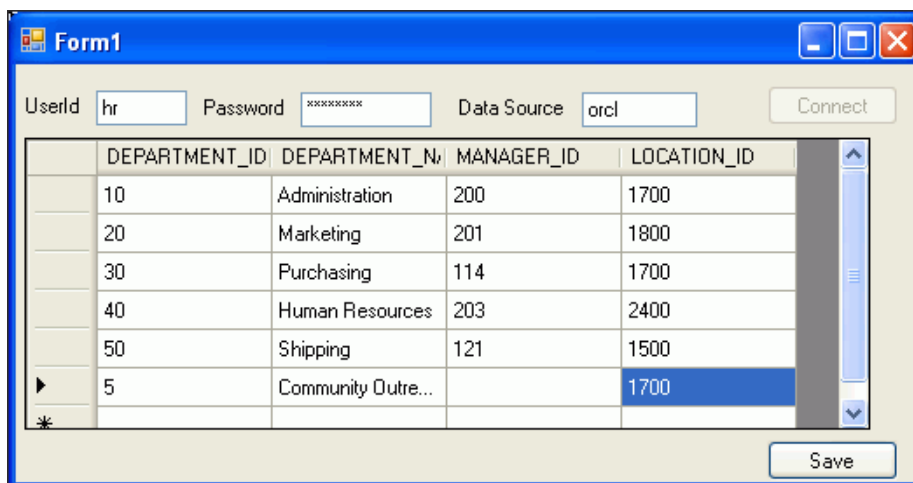


Inserting, Deleting, and Updating Data

This section demonstrates how to use your new application to directly manipulate data in the database.

To insert, delete and update data:

1. Run the application you created in the last section, entering the login and data source, and connecting to the database.
2. At the bottom of the data grid, enter a new record at the * prompt:
 - For DEPARTMENT_ID, enter 5.
 - For DEPARTMENT_NAME, enter Community Outreach.
 - Leave MANAGER_ID without a value.
 - For LOCATION_ID, enter 1700.



3. Click **Save**.
4. Close the application to check if the new record is saved.
5. Run the application again, and connect to the database.

Note that the new department is at the top of the `DEPARTMENTS` table, in numerical order by `DEPARTMENT_ID`.

6. Change the name of the department to `Community Volunteers`, and click the **Save** button.

The screenshot shows a window titled 'Form1' with a table of department data. The table has columns: DEPARTMENT_ID, DEPARTMENT_N, MANAGER_ID, and LOCATION_ID. The record with DEPARTMENT_ID 5 is highlighted in blue and its name is 'Community Volun...'. Other records include Administration (200), Marketing (201), Purchasing (114), Human Resources (203), and Shipping (121). A 'Save' button is visible at the bottom right.

	DEPARTMENT_ID	DEPARTMENT_N	MANAGER_ID	LOCATION_ID
▶	5	Community Volun...		1700
	10	Administration	200	1700
	20	Marketing	201	1800
	30	Purchasing	114	1700
	40	Human Resources	203	2400
	50	Shipping	121	1500

7. Repeat Step 4, run the application again, and connect to the database, and note that the name of the department is changed.
8. Select the entire record you just changed (click the cursor icon in the far left column), and delete it using the **Delete** key. Click the **Save** button.

The screenshot shows the same 'Form1' window. The record with DEPARTMENT_ID 5 is now selected, indicated by a mouse cursor pointing to the leftmost column (the selection column) of that row. The table data is identical to the previous screenshot.

	DEPARTMENT_ID	DEPARTMENT_N	MANAGER_ID	LOCATION_ID
▶	5	Community Volun...		1700
	10	Administration	200	1700
	20	Marketing	201	1800
	30	Purchasing	114	1700
	40	Human Resources	203	2400
	50	Shipping	121	1500

9. Repeat Step 4, run the application again, and connect to the database, and note that the new record is no longer part of the `DEPARTMENTS` table.
10. Close the application.

Using Oracle Developer Tools for Visual Studio

This chapter contains:

- [Using Oracle Developer Tools](#)
- [Connecting to the Oracle Database](#)
- [Creating a Table and Its Columns](#)
- [Creating a Table Index](#)
- [Adding Table Constraints](#)
- [Adding Data to a Table](#)
- [Generating Code Automatically to Display and Update Data](#)

Using Oracle Developer Tools

Oracle Developer Tools for Visual Studio (ODT) is a tightly integrated Add-in for Visual Studio. Using enhancements that ODT brings to the Server Explorer, you can automatically create tables, indexes, constraints, data connections and other database schema objects. Additionally you can automatically generate application code.

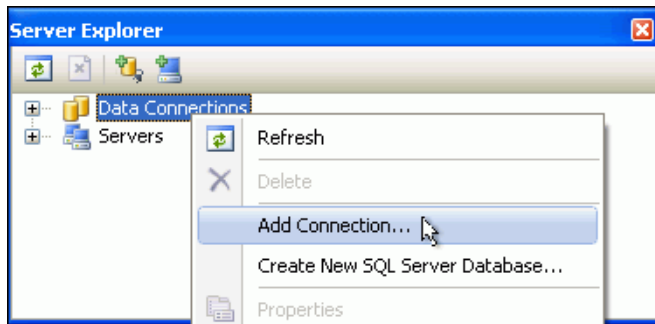
See Also: ["Overview of Oracle Developer Tools for Visual Studio"](#) on page 1-2

Connecting to the Oracle Database

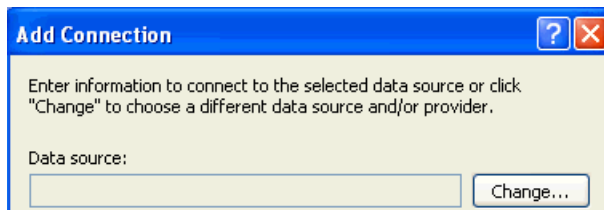
This section shows you how to use the Server Explorer to connect to the Oracle Database for the purpose of automatically creating or modifying database schema objects.

To connect to the database:

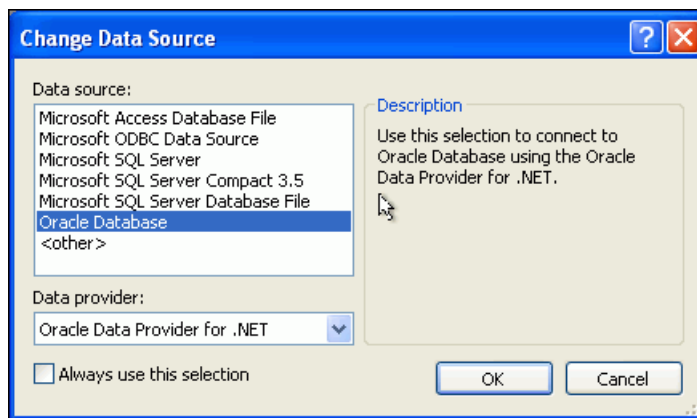
1. From the View menu, select **Server Explorer**.
2. In Server Explorer, right-click **Data Connections**.
3. Select **Add Connection**.



4. When the Add Connection window appears, determine if the Data source says Oracle Database (Oracle ODP.NET).
If it does, skip to Step 6.



If Data source does not say Oracle Database (Oracle ODP.NET), select **Change**.
The Change Data Source window appears.



5. Choose Oracle Database and then select Oracle Data Provider for .NET.
6. On the Connection Details tab, in the Add Connection window, enter the following information:

Data source name: For this example, use the alias of the remote database instance, orcl.

If you are connecting to a database on the same computer, use the Local Database.

Select the **Use a specific user name and password** option.

For **User name**, enter HR.

For **Password**, enter the password created when the `hr` account was unlocked and set up.

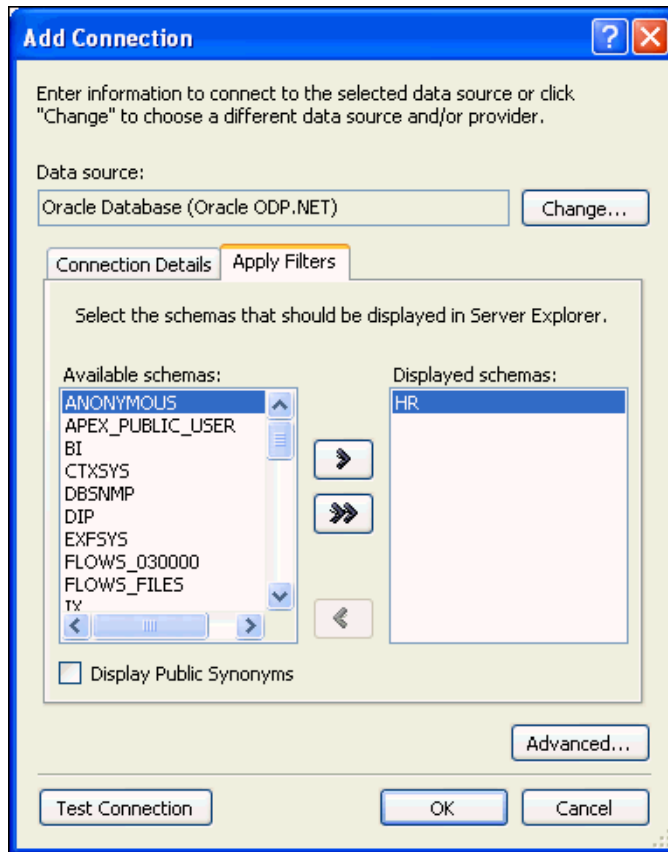
To save the password for future sessions, check the **Save password** box.

Ensure that **Role** is set to `Default`. This refers to the default roles that have been granted to the user `hr`.

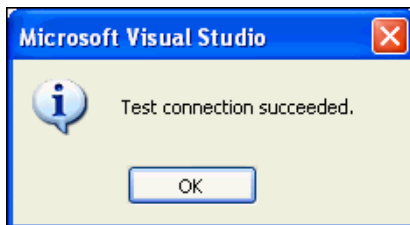
The **Connection name** should be generated automatically from the **Data source name** and the **User name** values. In this exercise, it will be `HR.orcl`.

The screenshot shows the 'Add Connection' dialog box. At the top, it says 'Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.' Below this, the 'Data source' is 'Oracle Database (Oracle ODP.NET)'. The 'Connection Details' tab is active, showing 'Data source name' as 'orcl'. Under authentication, 'Use a specific user name and password' is selected. The 'User name' is 'HR' and the 'Password' is masked. The 'Save password' checkbox is checked. The 'Role' is 'Default' and the 'Connection name' is 'HR.orcl'. Buttons for 'Advanced...', 'Test Connection', 'OK', and 'Cancel' are at the bottom.

7. Click the **Apply Filters** tab, and verify that the `HR` schema is in the **Displayed schemas** column. When you expand the schema category nodes in the data connection, only those schema objects (tables, views, and so on) selected in the **Apply Filters** tab appear.



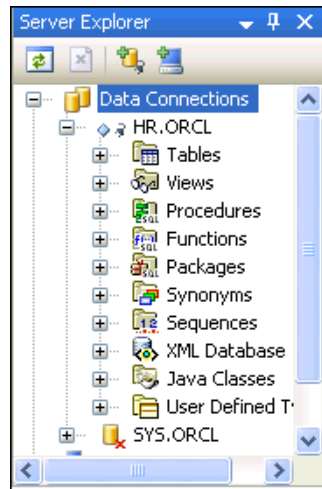
8. Click **Test connection**.



The test should succeed. Click **OK**.

If the test fails, it may be due to one or more of the following issues that you must address before proceeding with further steps:

- The database is not started.
 - The database listener is not started.
 - The database connectivity is not properly configured.
 - You do not have the correct user name, password, or role.
9. In the Add Connection window, click **OK**.
 10. In the Server Explorer, expand the **HR.ORCL** connection to show the contents of the HR schema. You should see Tables, Views, Procedures, Functions, Packages, Synonyms, Sequences, and so on.

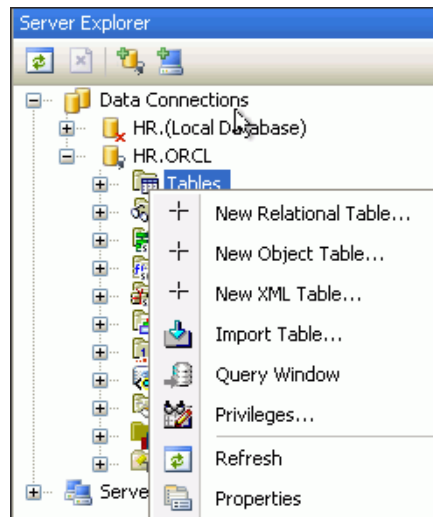


Creating a Table and Its Columns

Oracle Developer Tools includes a user interface for creating database objects. In this section, you will create a table named `DEPENDENTS`.

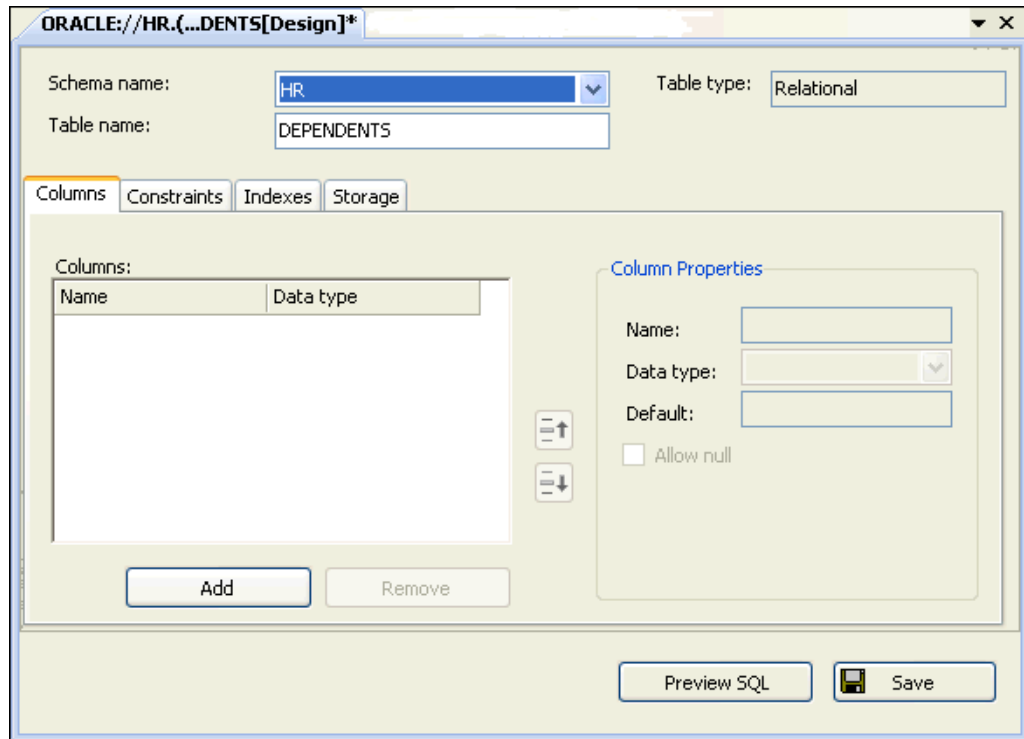
To create a table:

1. In Server Explorer, right-click **Tables** and select **New Relational Table**.



A table design window appears.

2. In design view, enter `DEPENDENTS` for **Table name**.

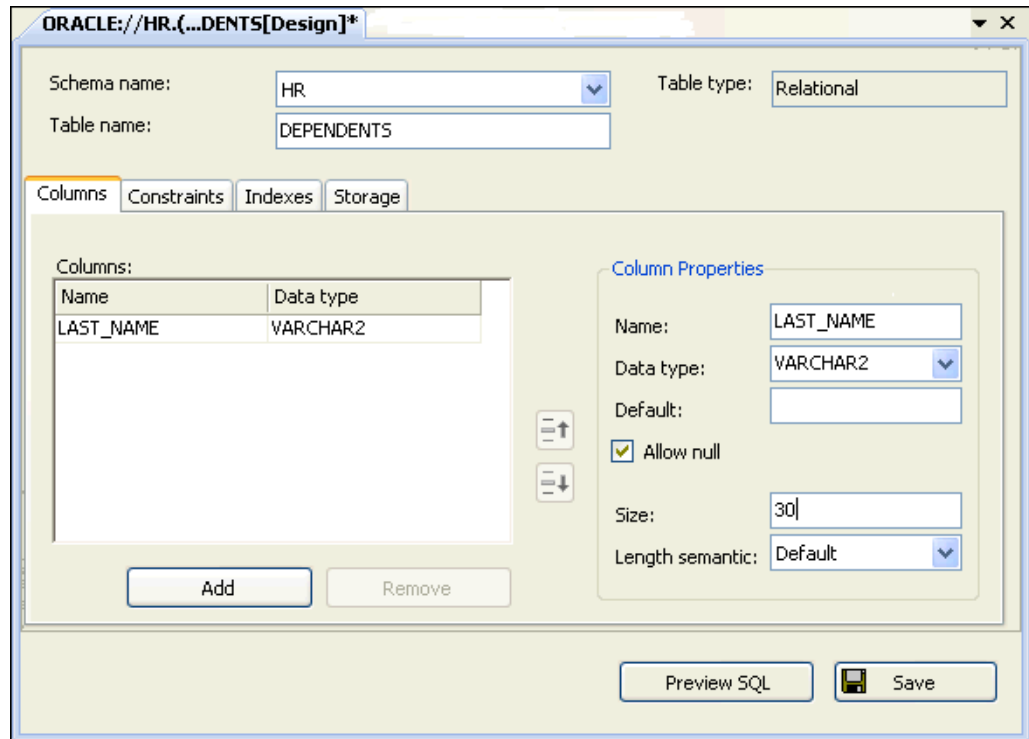


3. In the Column Properties tab, add the following six columns in this manner:

Click **Add**. Then enter the new column information. Keep clicking add until you have added all the new columns.

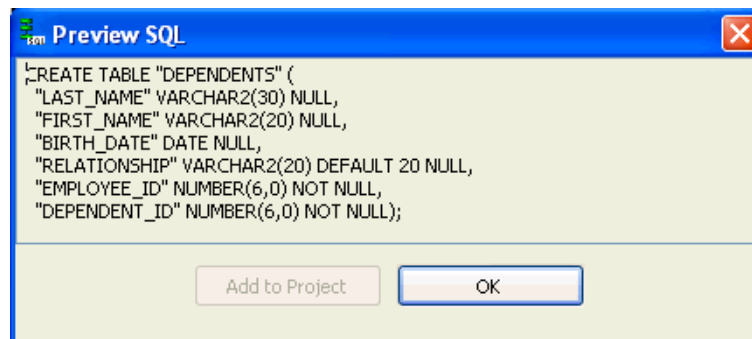
Fields may differ depending on the data type. You might have to close windows such as Server Explorer or Solution Explorer to access the entire tab.

 - **Name** LAST_NAME, **Data Type** VARCHAR2, and **Size** 30. Leave all other properties at their default values.
 - **Name** FIRST_NAME, **Data Type** VARCHAR2, and **Size** 20. Leave all other properties at their default values.
 - **Name** BIRTH_DATE, **Data Type** DATE. Leave all other properties at their default values.
 - **Name** RELATIONSHIP, **Data Type** VARCHAR2, and **Size** 20. Leave all other properties at their default values.
 - **Name** EMPLOYEE_ID, **Data Type** NUMBER, deselect **Allow null**, enter **Precision** 6 and **Scale** 0.
 - **Name** DEPENDENT_ID, **Data Type** NUMBER, deselect **Allow null** check box, enter **Precision** 6 and **Scale** 0.



4. Click **Preview SQL**.

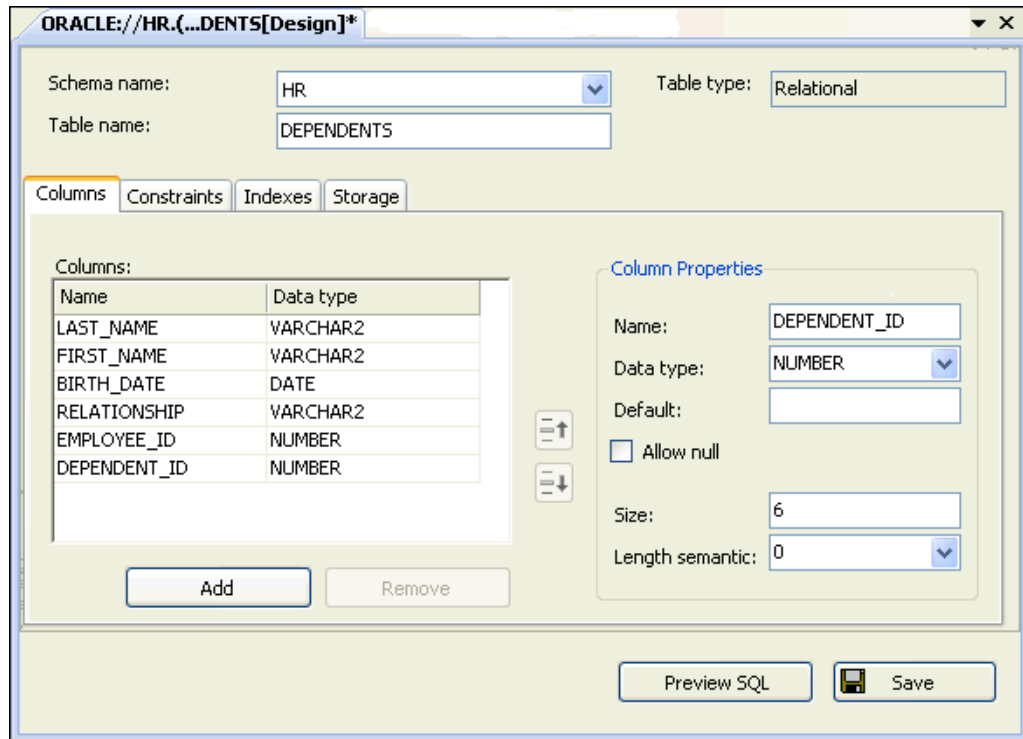
The SQL statement for constructing the table appears in the Preview SQL window, similar to this.



Click **OK** to close the Preview SQL window.

5. In the table design view, click **Save**.

This action creates the new table `DEPENDENTS` in the `HR` schema. The new table is listed in the Server Explorer.

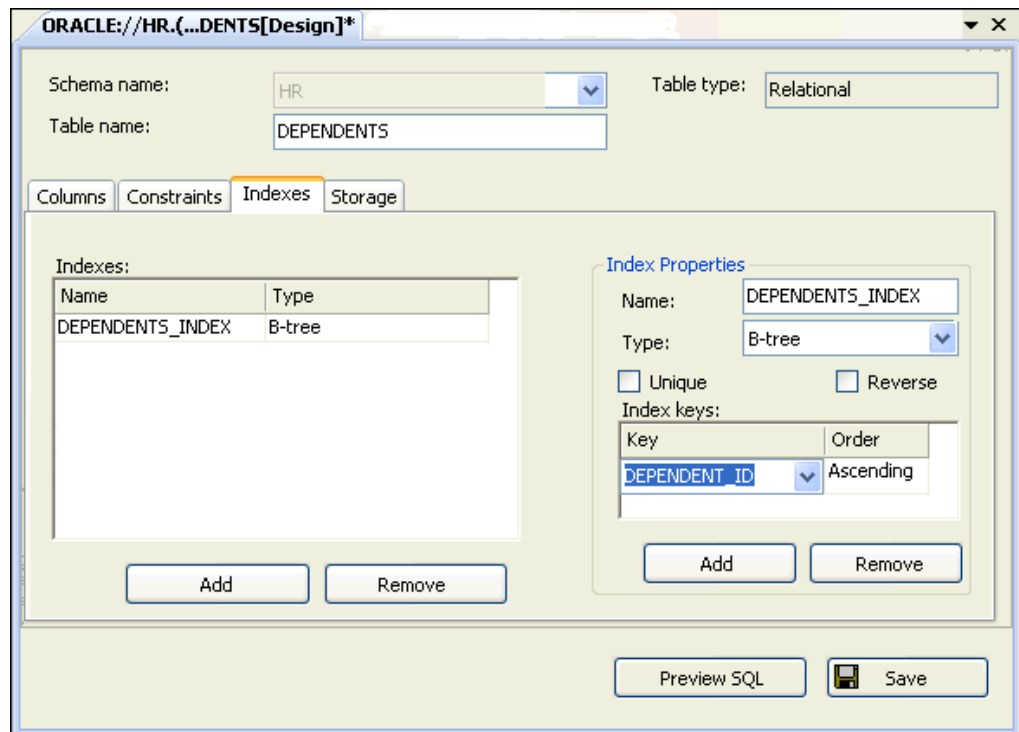


Creating a Table Index

Indexes are an optional but very powerful feature of relational databases. An index enables quick access to the rows (or records) in a table. In this section, you will create an index for the DEPENDENTS table.

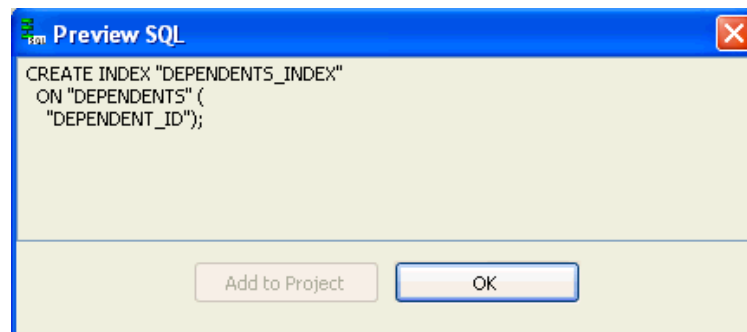
To create an index:

1. In the DEPENDENTS Table Design view, click the **Indexes** tab.
2. Click **Add** under the Indexes area.
The Index Properties area becomes active.
3. Under Index Properties (to the right), enter the **Name** DEPENDENTS_INDEX, and leave all other properties in their default state.
4. At the bottom of the Index Properties area, click **Add**.
5. Under Index keys, click in the first cell of the **Key** column, and select DEPENDENT_ID from the list.



6. Click **Preview SQL**

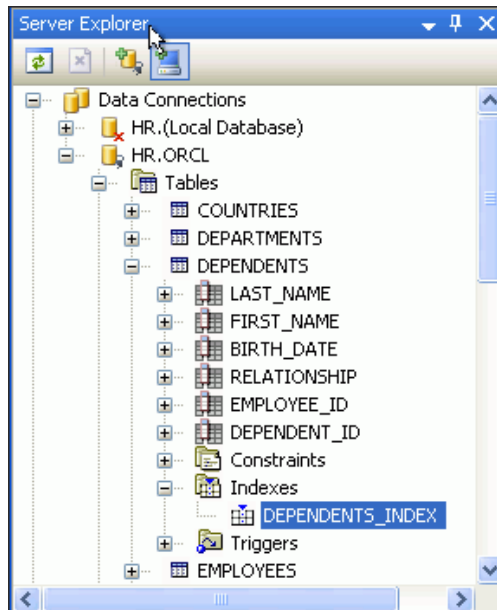
A Preview SQL window appears, displaying SQL statement to construct the index.



Click **OK** to close the Preview SQL window.

7. In the table design view, click **Save**.

This creates the new index on the table DEPENDENTS in the HR schema. To see this in the Server Explorer, expand the DEPENDENTS table and related Indexes.



Adding Table Constraints

The database uses constraints to automatically enforce data integrity defining rules for permissible data values. Constraints also implement primary and foreign keys in the table. In this section, you will add such constraints to the new table `DEPENDENTS`.

How to add foreign and primary keys:

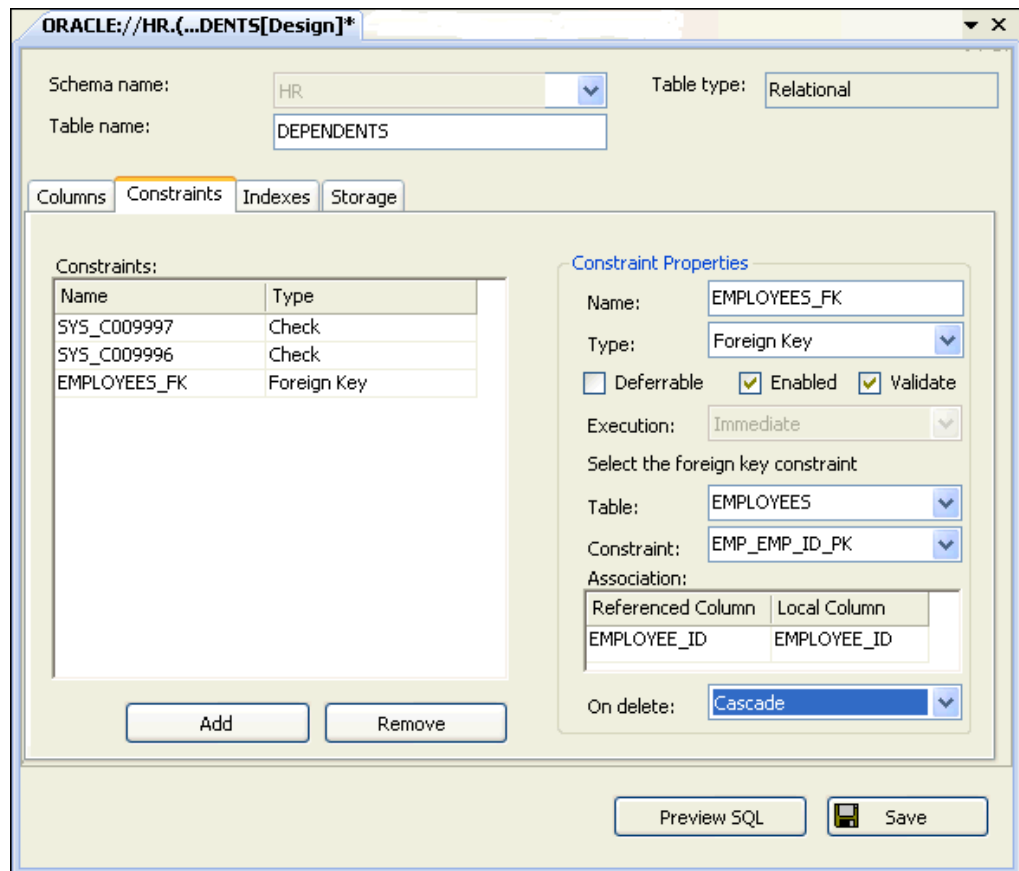
1. In the `DEPENDENTS` table design view, click the **Constraints** tab.

Note that depending on your configuration, there may already be default check constraints in the list.

2. Under the Constraints area, add the following constraints in this manner:

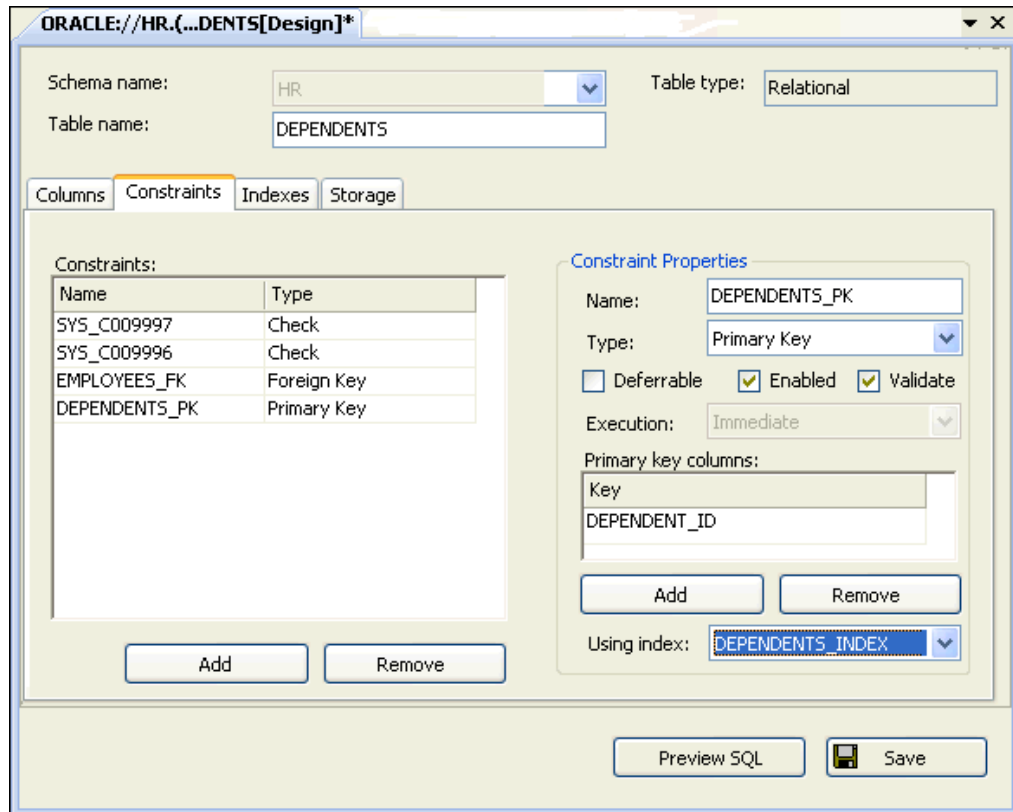
Under Constraint Properties, Click **Add**. Then enter the new constraint information. Keep clicking add until you have added all the new constraints.

- **Name** `EMPLOYEES_FK`, **Type** Foreign Key, **Table** `EMPLOYEES`, **Constraint** `EMP_EMP_ID_PK`. Under Association, select **Referenced Column:** `EMPLOYEE_ID`, and **Local Column:** `EMPLOYEE_ID`, set the **On delete** value to `Cascade`. Leave all other properties at their default values.



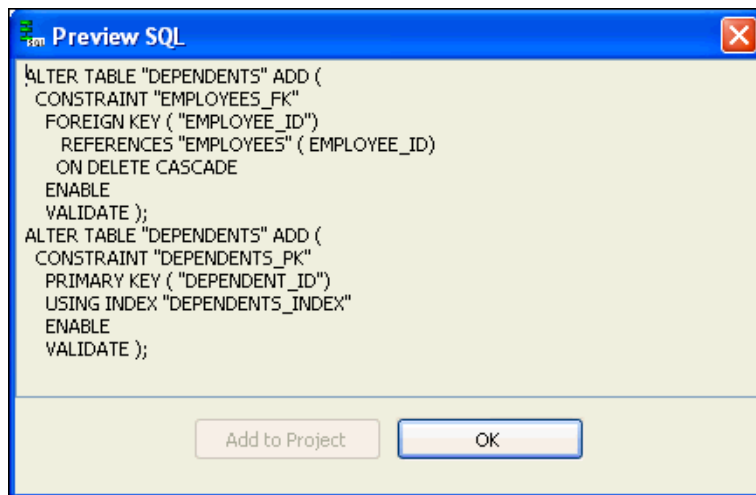
- **Name** DEPENDENTS_PK, **Type** Primary Key.

Under the Primary key columns area, click **Add** (you may need to scroll down). Under Primary Key Columns, select **Key:** DEPENDENT_ID, set the **Using index** value to DEPENDENTS_INDEX. Leave all other properties at their default values.



3. Click **Preview SQL**.

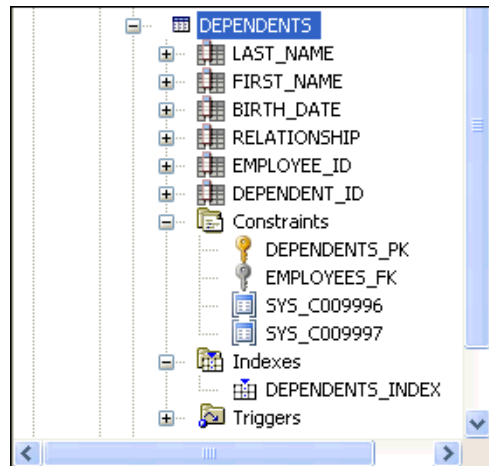
The Preview SQL window displays the code generated for constraints on table DEPENDENTS. Note that adding constraints is an ALTER TABLE command because constraints change the definitions of the DEPENDENT_ID and EMPLOYEE_ID columns of the table.



Click **OK** to close the Preview SQL window.

4. In the table design view, click **Save**

This action creates the two new constraints on the `DEPENDENTS` table in the `HR` schema. To see the Server Explorer, expand the hierarchy tree for the table `DEPENDENTS` and constraints.

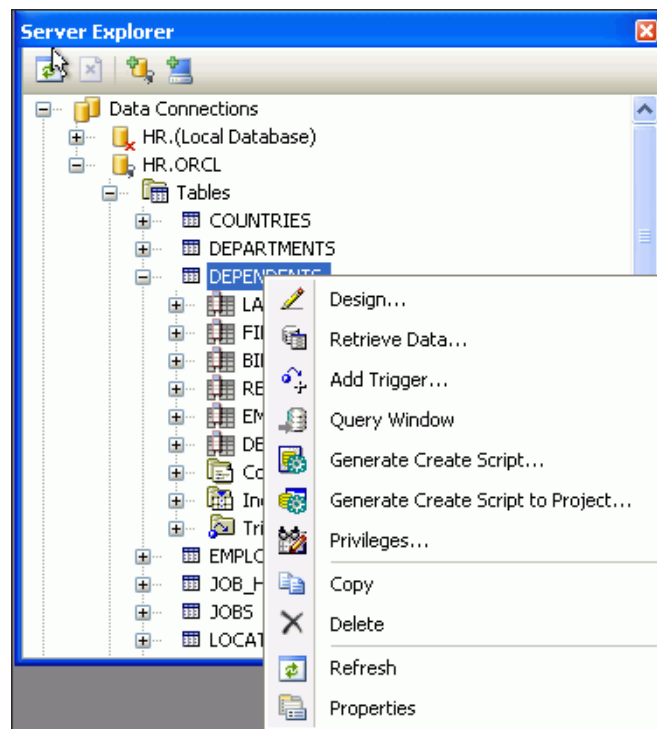


Adding Data to a Table

You must now add data to the new `DEPENDENTS` table.

To populate a table:

1. In Server Explorer, right-click the `DEPENDENTS` table and select **Retrieve Data**.



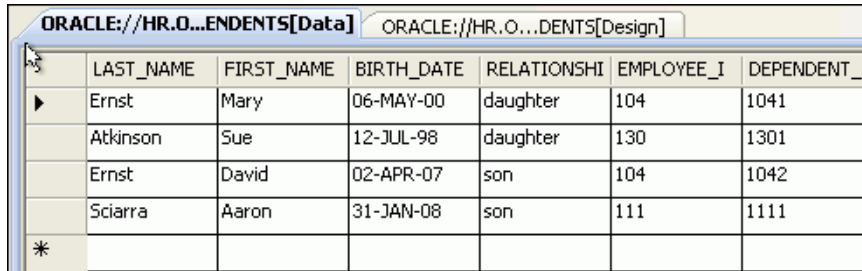
A table grid for `DEPENDENTS` appears in design view.

2. Enter the four records listed in [Table 5-1](#) into the table grid.

Table 5–1 New Data for the DEPENDENTS Table

LAST_NAME	FIRST_NAME	BIRTH_DATE	RELATIONSHIP	EMPLOYEE_ID	DEPENDENT_ID
Ernst	Mary	06-MAY-2000	daughter	104	1041
Atkinson	Sue	12-JUL-1998	daughter	130	1301
Ernst	David	02-APR-2007	son	104	1042
Sciarra	Aaron	31-JAN-2008	son	111	1111

The grid now looks as follows:



Note that the data is automatically saved as you move between rows.

Generating Code Automatically to Display and Update Data

To explore the content of the DEPENDENTS table, we will build a form that uses a simple table query. In this section you will use the Visual Studio integrated development environment (IDE), to automatically generate the code that corresponds to your actions.

To create a new Data Source:

1. Start a new project, as described in ["Creating a New Project"](#) on page 3-1. Name the new project as indicated.

Visual C#:

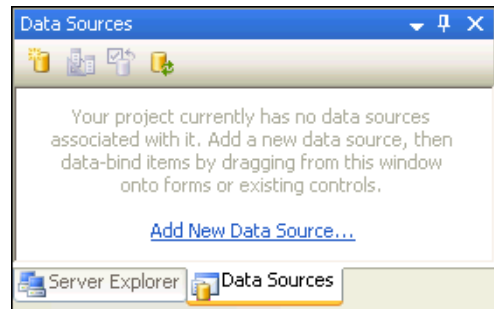
HR_ODT_CS.

Visual Basic:

HR_ODT_VB.

2. Check **Create Directory for Solution**. Click **OK**.
3. Switch to the Form1 design view, if you are not already in it.
 Note: All applications start with Form1, but this is not related to applications created in previous chapters.
4. Click on the Server Explorer window to enable the Show Data Sources window.
5. From the Visual Studio **Data** menu, select **Show Data Sources**.

The Data Source window appears.

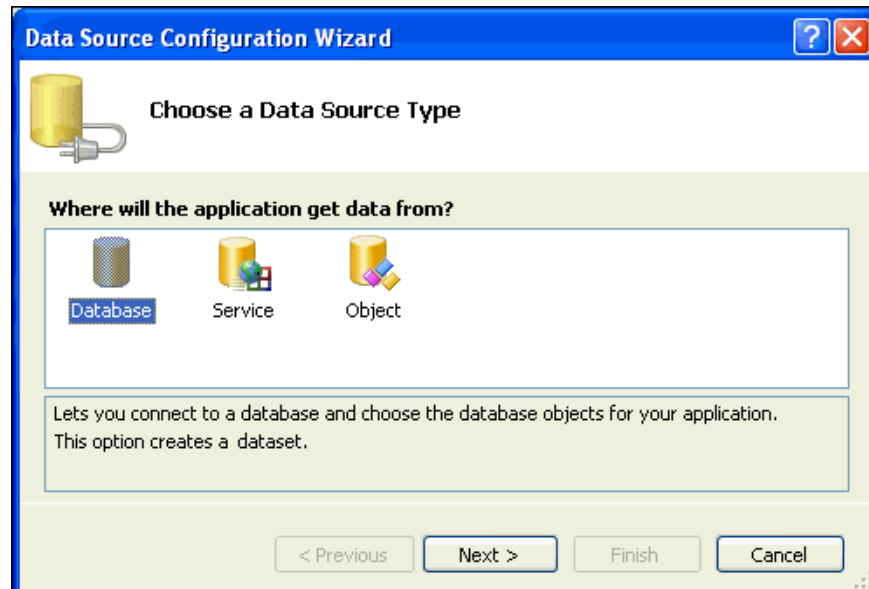


6. In the Data Sources window, click **Add New Data Source**.

The Data Source Configuration Wizard opens.

7. In the Data Source Configuration Wizard, under Choose a Data Source Type, select **Database**.

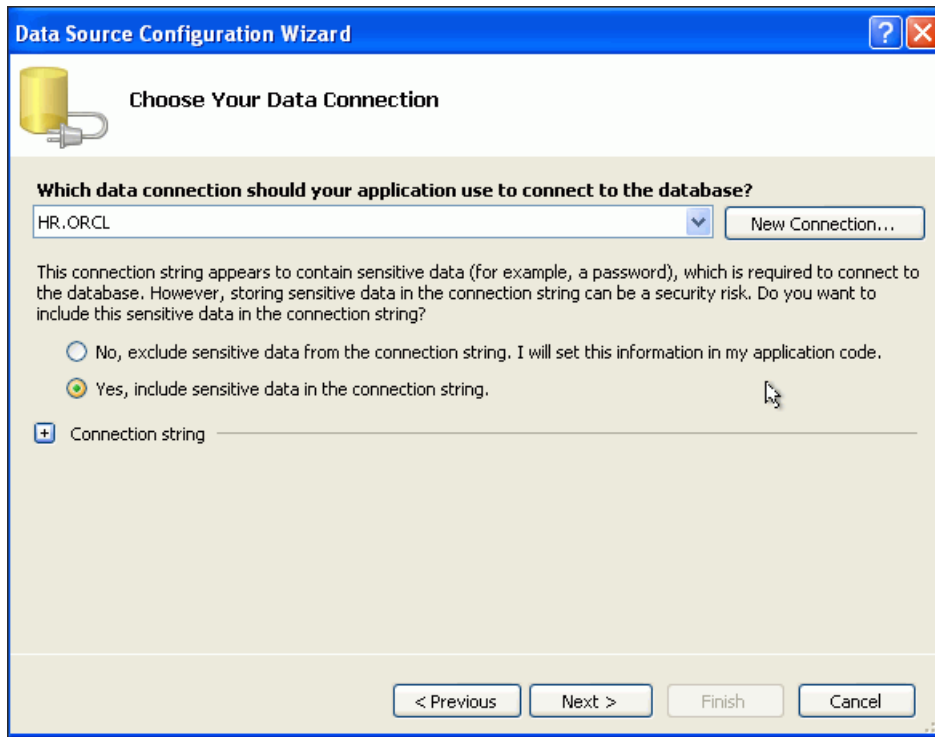
Click **Next**.



8. Under Choose Your Data Connection, select **HR.ORCL**, or **HR.(Local Database)**. For this example, we will use HR.ORCL.

Select **Yes, include sensitive data in the connection string**.

Click **Next**.



9. Under Save the Connection String to the Application Configuration File, select **Yes, save the connection as: ConnectionString**.

Click **Next**.

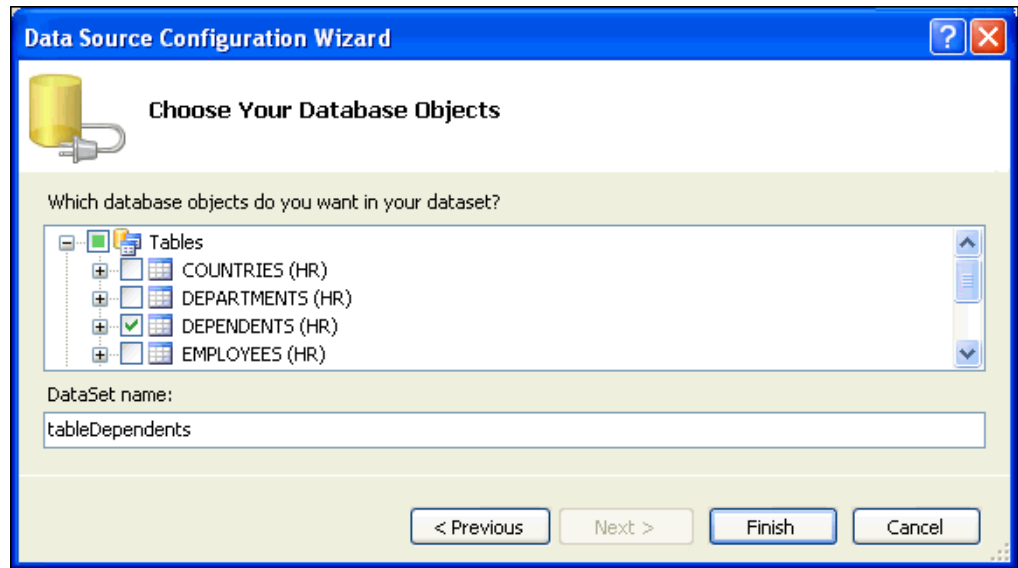


10. Under Choose Your Database Objects, expand **Tables**.

Check the **DEPENDENTS(HR)** table.

Change the **DataSet** name to **tableDependents**.

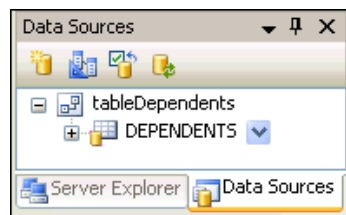
Click **Finish**.



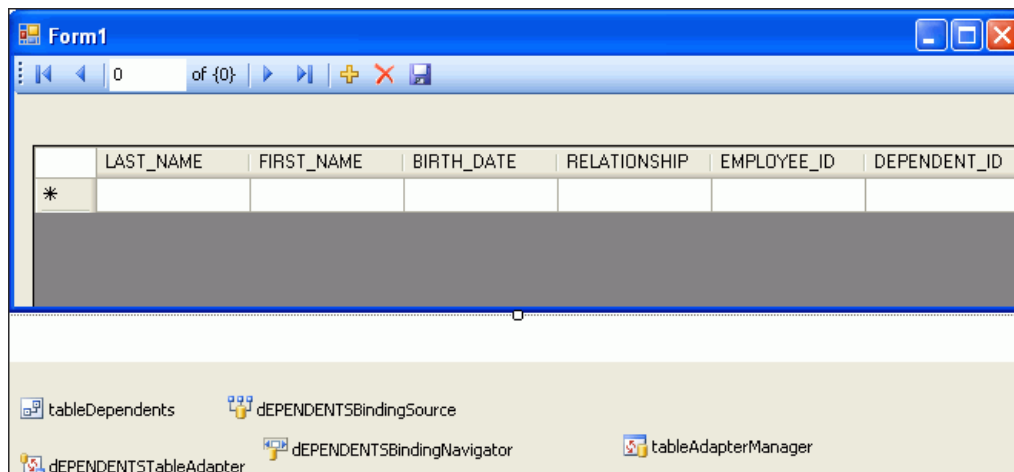
See Also: ["Using the DataSet Class with Oracle Data Provider for .NET"](#) on page 4-8 for information about the DataSet Class

To automatically generate code using drag-and-drop:

1. Switch to the Form1 Design view.
2. In the Data Sources window, expand **tableDependents**.



3. Select the **DEPENDENTS** table, and drag it onto Form1.
You may need to resize both the form and the table grid.



Note that along with the table grid (which includes record navigation elements), the following components were added to the design view of your project. These objects represent automatically generated code for Form1.

Visual C#:

```
tableDependents, DEPENDENTSBindingSource,  
DEPENDENTSTableAdapter, tableAdapterManager, and  
DEPENDENTSBindingNavigator
```

Visual Basic:

```
TableDependents, DEPENDENTSBindingSource,  
DEPENDENTSTableAdapter, TableAdapterManager, and  
DEPENDENTSBindingNavigator
```

4. Double-click the **Save** icon (floppy disk) near the top of Form1.
This opens the code window for the Save icon for Form1.
5. In the private method, `xxxSaveItem_Click()`, encapsulate the existing code in a `try...catch` block. See the code listed for the complete Visual C# and Visual Basic names of this automatically generated method.

Also, add a `MessageBox.Show()` call to both the try and catch sections. The updated method code follows, with new or changed code in bold font.

Visual C#:

```
private void DEPENDENTSBindingNavigatorSaveItem_Click(object sender, EventArgs  
e)  
{  
    try  
    {  
        this.Validate();  
        this.DEPENDENTSBindingSource.EndEdit();  
        this.tableAdapterManager.UpdateAll(this.tableDependents);  
  
        MessageBox.Show("Update successful");  
    }  
    catch (System.Exception ex)  
    {  
        MessageBox.Show("Update failed: " + ex.Message.ToString());  
    }  
}
```

Visual Basic#:

```
Private Sub DEPENDENTSBindingNavigatorSaveItem_Click(  
    ByVal sender As System.Object, ByVal e As System.EventArgs)  
    Handles DEPENDENTSBindingNavigatorSaveItem.Click  
  
    Try  
        Me.Validate()  
        Me.DEPENDENTSBindingSource.EndEdit()  
        Me.TableAdapterManager.UpdateAll(Me.TableDependents)  
        MessageBox.Show("Update successful")  
  
    Catch ex As Exception  
        MessageBox.Show("Update failed: " + ex.Message.ToString())  
  
    End Try
```

End Sub

6. To compile and run the application, follow the instructions in section "[Compiling and Running the Application](#)" on page 3-13.

You can test the new application in the following manner. The floppy disk icon represents the Save command.

To test the application:

1. Change the `DEPENDENT_ID` value for Mary Ernst to 1110 and click the **Save** icon. The message box `Update successful` should appear. Click **OK** to dismiss the message box.
2. Change the `EMPLOYEE_ID` value for David Ernst to 99999 and click the **Save** icon. The following message should appear: `Update failed: ORA-02291: integrity constraint (HR.EMPLOYEES_FK) violated - parent key not found`. Click **OK** to dismiss the message box.

Using PL/SQL Stored Procedures and REF CURSORS

This chapter contains:

- [Introduction to PL/SQL Stored Procedures](#)
- [Introduction to PL/SQL Packages and Package Bodies](#)
- [Introduction to REF CURSORS](#)
- [Creating a PL/SQL Stored Procedure that Uses REF CURSORS](#)
- [Modifying an ODP.NET Application to Run Stored Procedures](#)
- [Running a PL/SQL Stored Procedure Using an ODP.NET Application](#)

Introduction to PL/SQL Stored Procedures

A stored procedure is a named set of PL/SQL statements designed to perform an action. Stored procedures are stored inside the database. They define a programming interface for the database rather than allowing the client application to interact with database objects directly. Stored procedures are typically used for data validation or to encapsulate large, complex processing instructions that combine several SQL queries.

Stored functions have a single return value parameter. Unlike functions, procedures may or may not return values.

Introduction to PL/SQL Packages and Package Bodies

A PL/SQL package stores related items as a single logical entity. A package is composed of two distinct pieces:

- The **package specification** defines what is contained in the package; it is analogous to a header file in a language such as C++. The specification defines all public items. The specification is the published interface to a package.
- The **package body** contains the code for the procedures and functions defined in the specification, and the code for private procedures and functions that are not declared in the specification. This private code is only visible within the package body.

The package specification and body are stored as separate objects in the data dictionary and can be seen in the `user_source` view. The specification is stored as the `PACKAGE` type, and the body is stored as the `PACKAGE BODY` type.

While it is possible to have a specification without a body, as when declaring a set of public constants, it is not possible to have a body with no specification.

Introduction to REF CURSORS

Using REF CURSORS is one of the most powerful, flexible, and scalable ways to return query results from an Oracle Database to a client application.

A REF CURSOR is a PL/SQL data type whose value is the memory address of a query work area on the database. In essence, a REF CURSOR is a pointer or a handle to a result set on the database. REF CURSORS are represented through the `OracleRefCursor` ODP.NET class.

REF CURSORS have the following characteristics:

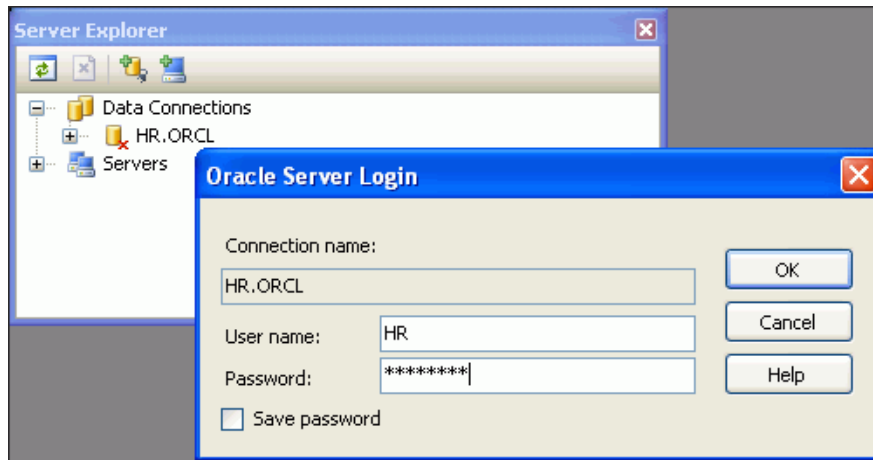
- A REF CURSOR refers to a memory address on the database. Therefore, the client must be connected to the database during the lifetime of the REF CURSOR in order to access it.
- A REF CURSOR involves an additional database round-trip. While the REF CURSOR is returned to the client, the actual data is not returned until the client opens the REF CURSOR and requests the data. Note that data is not be retrieved until the user attempts to read it.
- A REF CURSOR is not updatable. The result set represented by the REF CURSOR is read-only. You cannot update the database by using a REF CURSOR.
- A REF CURSOR is not backward scrollable. The data represented by the REF CURSOR is accessed in a forward-only, serial manner. You cannot position a record pointer inside the REF CURSOR to point to random records in the result set.
- A REF CURSOR is a PL/SQL data type. You create and return a REF CURSOR inside a PL/SQL code block.

Creating a PL/SQL Stored Procedure that Uses REF CURSORS

This section demonstrates how to create a PL/SQL stored procedure.

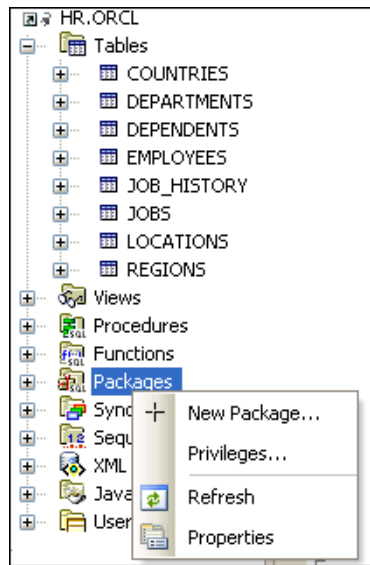
To create a stored procedure:

1. Open Server Explorer and double-click HR to open the connection to the HR schema created in ["Connecting to the Oracle Database"](#) on page 5-1.



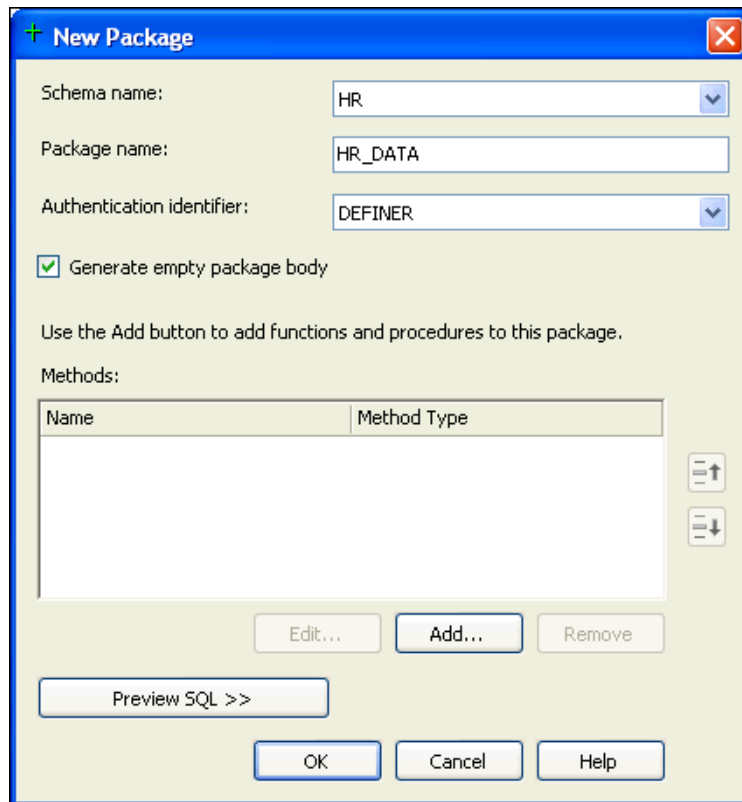
If you have not previously saved the password, the Oracle Server Login opens and you can enter the password now. If you have saved the password, then the connection expands immediately.

- In Server Explorer, right-click **Packages** and select **New Package**.



The New Package window appears.

- In the New Package window, change the **Package Name** to HR_DATA.
- Under the Methods area, click **Add**.



The Add Method window appears.

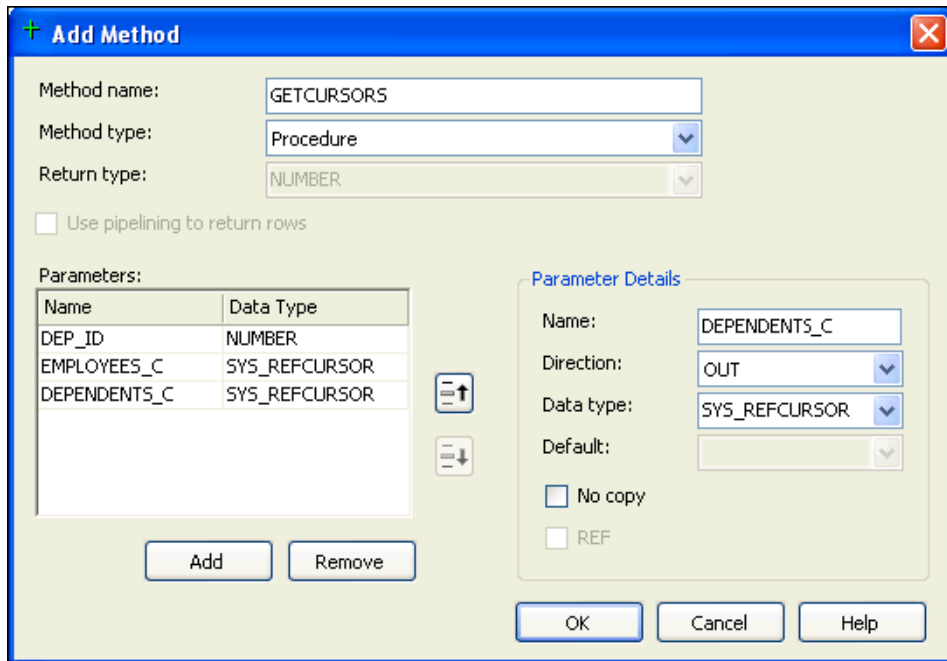
- In the Add Method window, enter **Method Name** GETCURSORS, and change **Method Type** to Procedure.

6. Under Parameters, click **Add**.

This starts the process of adding parameters.

Under the Parameter Details group on the right, enter these three parameters. Click **Add** before each parameter that you need to add.

- **Name:** DEP_ID, **Direction:** select IN , **Data Type:** select NUMBER.
- **Name:** EMPLOYEES_C, **Direction:** select OUT, **Data Type:** select SYS_REFCURSOR .
- **Name:** DEPENDENTS_C , **Direction:** OUT, **Data Type:** select SYS_REFCURSOR.



7. Click **OK** when you finish adding parameters.

The New Package window reappears.

8. In the New Package window, click **Preview SQL** to see the SQL code created.

A Preview SQL window appears, containing code similar to the following. Note that this code has been abbreviated by removing most of the comments.

```
CREATE PACKAGE "HR"."HR_DATA" IS

  -- Declare types, variables, constants, exceptions, cursors,
  -- and subprograms that can be referenced from outside the package.

  PROCEDURE "GETCURSORS" (
    "DEP_ID" IN NUMBER,
    "EMPLOYEES_C" OUT SYS_REFCURSOR,
    "DEPENDENTS_C" OUT SYS_REFCURSOR);

END "HR_DATA" ;

CREATE PACKAGE BODY "HR"."HR_DATA" IS
```

```
-- Implement subprograms, initialize variables declared in package
-- specification.

-- Make private declarations of types and items, that are not accessible
-- outside the package

PROCEDURE "GETCURSORS" (
  "DEP_ID" IN NUMBER,
  "EMPLOYEES_C" OUT SYS_REFCURSOR,
  "DEPENDENTS_C" OUT SYS_REFCURSOR) IS

-- Declare constants and variables in this section.

  BEGIN -- executable part starts here

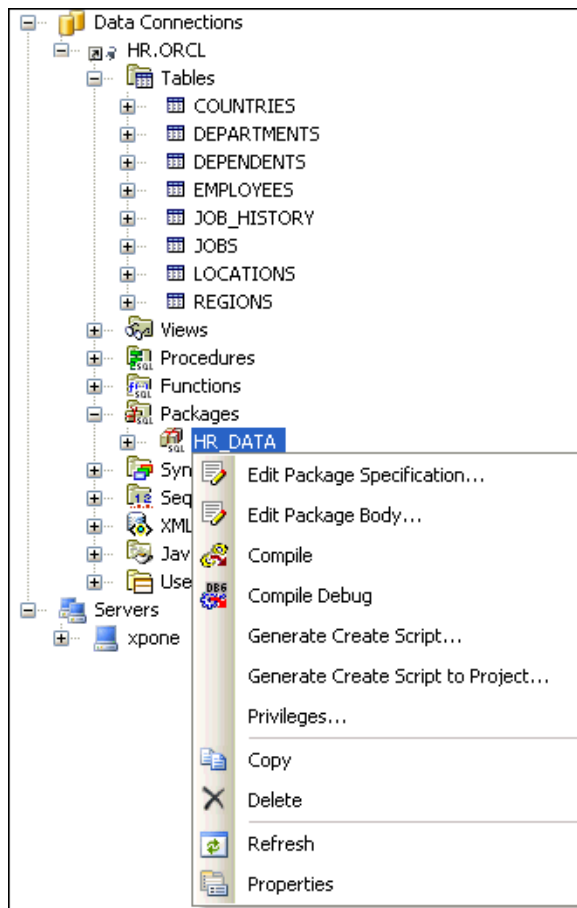
    NULL;

-- EXCEPTION -- exception-handling part starts here

  END "GETCURSORS";

END "HR_DATA";
```

9. Click **OK** to close the Preview SQL window.
10. In the New Package window, click **OK** to save the new package.
The new package, HR_DATA, now appears in the Server Explorer.
11. In the Server Explorer, right-click the package HR_DATA, and select **Edit Package Body**.

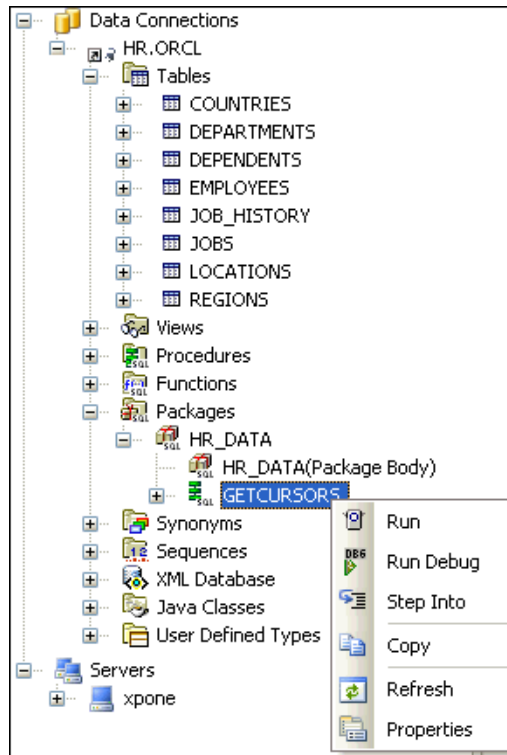


The code for the package appears.

12. Scroll down to the body of the GETCURSORS procedure, and after BEGIN, replace the line `NULL;` with the following code:

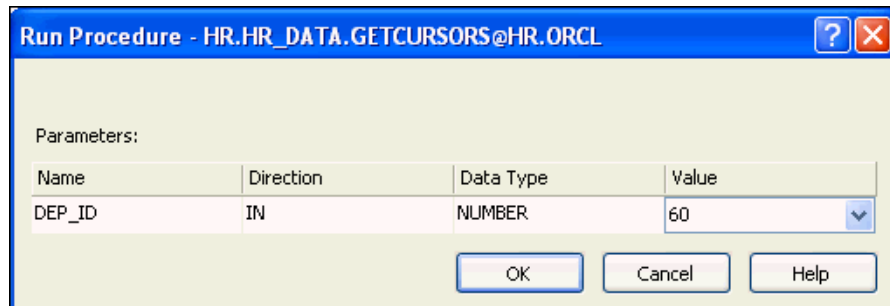
```
OPEN EMPLOYEES_C FOR SELECT * FROM EMPLOYEES
    WHERE DEP_ID=DEPARTMENT_ID;
OPEN DEPENDENTS_C FOR SELECT * FROM DEPENDENTS;
```

13. Save the changes to the package.
14. To run the stored procedure, in Server Explorer, expand the HR_DATA package. Right-click the GETCURSORS method, and select **Run**.



The Run Procedure window appears.

15. In the Run Procedure window, enter a **Value** of 60 for dep_id.



16. Click **OK**.

The Output window appears, showing that the run was successful.

In the result window, the following message appears:

Procedure <HR.HR_DATA.GETCursors@hr.database> was run successfully.

Under this message, note two output parameters (together with DEP_ID): EMPLOYEES_C and DEPENDENTS_C.

17. Select the **Value** column entry for EMPLOYEES_C.

The Parameter Details area appears, showing the employees in department 60. The value for DEP_ID is 60.

Parameters:			
Name	Direction	Data Type	Value
DEP_ID	IN	NUMBER	60
EMPLOYEES_C	OUT	REF CURSOR	<Click here for details...>
DEPENDENTS_C	OUT	REF CURSOR	<Click here for details...>

Parameter Details - EMPLOYEES_C:						
EMPLOYEE_I	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUM	HIRE_DATE	JOB_ID
103	Alexander	Hunold	AHUNOLD	590.423.4567	1/3/1990	IT_PROG
104	Bruce	Ernst	BERNST	590.423.4568	5/21/1991	IT_PROG
105	David	Austin	DAUSTIN	590.423.4569	6/25/1997	IT_PROG
106	Valli	Pataballa	VPATABAL	590.423.4560	2/5/1998	IT_PROG
107	Diana	Lorentz	DLORENTZ	590.423.5567	2/7/1999	IT_PROG

18. Select the **Value** column entry for `DEPENDENTS_C`.

The Parameter Details area appears, showing the value of the `DEPENDENTS_C`.

Parameters:			
Name	Direction	Data Type	Value
DEP_ID	IN	NUMBER	60
EMPLOYEES_C	OUT	REF CURSOR	<Click here for details...>
DEPENDENTS_C	OUT	REF CURSOR	<Click here for details...>

Parameter Details - DEPENDENTS_C:					
LAST_NAME	FIRST_NAME	BIRTH_DATE	RELATIONSHI	EMPLOYEE_I	DEPENDENT_
Ernst	Mary	5/6/2000	daughter	104	1122
Atkinson	Sue	7/12/1998	daughter	130	1301
Ernst	David	4/2/2007	son	104	1042
Sciarra	Aaron	1/31/2008	son	111	1111

Modifying an ODP.NET Application to Run Stored Procedures

This section demonstrates how to modify your Oracle Data Provider for .NET application to run a PL/SQL stored procedure, using the `GETCURSORS` stored procedure as a sample.

To modify your application to run a stored procedure:

1. Open the application `HR_Connect_CS` or `HR_Connect_VB`.
2. Make a copy of `Form3 .xx`, which you finished at the end of [Chapter 4](#) and name it `Form4 .xx`, following the instructions in [Appendix B, "Copying a Form"](#).
3. With `Form1` selected, switch to code view.
4. In the `try` block of the `connect_Click()` method, replace the two command assignment lines, starting with `cmd = New OracleCommand...` with the code indicated.

Visual C#:

```
cmd = new OracleCommand("HR_DATA.GETCURSORS", conn);
cmd.CommandType = CommandType.StoredProcedure;
```

Visual Basic:

```
cmd = new OracleCommand("HR_DATA.GETCURSORS", conn)
cmd.CommandType = CommandType.StoredProcedure
```

5. Under the code added in Step 3, add definitions and bindings for the three parameters of the GETCURSORS stored procedure as OracleParameter objects, calling them dep_id, employees_c and dependents_c.

Visual C#:

```
OracleParameter dep_id = new OracleParameter();
dep_id.OracleDbType = OracleDbType.Decimal;
dep_id.Direction = ParameterDirection.Input;
dep_id.Value = 60;
cmd.Parameters.Add(dep_id);
```

```
OracleParameter employees_c = new OracleParameter();
employees_c.OracleDbType = OracleDbType.RefCursor;
employees_c.Direction = ParameterDirection.Output;
cmd.Parameters.Add(employees_c);
```

```
OracleParameter dependents_c = new OracleParameter();
dependents_c.OracleDbType = OracleDbType.RefCursor;
dependents_c.Direction = ParameterDirection.Output;
cmd.Parameters.Add(dependents_c);
```

Visual Basic:

```
Dim dep_id As OracleParameter = New OracleParameter
dep_id.OracleDbType = OracleDbType.Decimal
dep_id.Direction = ParameterDirection.Input
dep_id.Value = 60
cmd.Parameters.Add(dep_id)
```

```
Dim employees_c As OracleParameter = New OracleParameter
employees_c.OracleDbType = OracleDbType.RefCursor
employees_c.Direction = ParameterDirection.Output
cmd.Parameters.Add(employees_c)
```

```
Dim dependents_c As OracleParameter = New OracleParameter
dependents_c.OracleDbType = OracleDbType.RefCursor
dependents_c.Direction = ParameterDirection.Output
cmd.Parameters.Add(dependents_c)
```

6. Build the application.

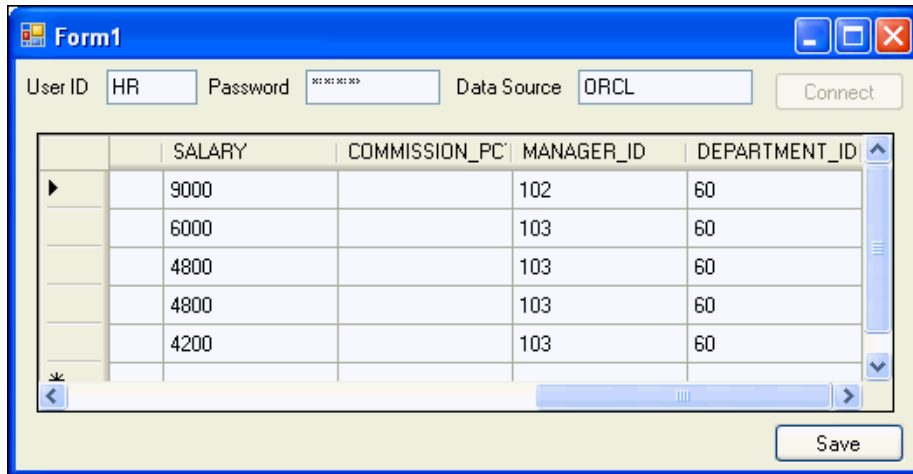
Running a PL/SQL Stored Procedure Using an ODP.NET Application

This section demonstrates how to run a PL/SQL stored procedure, such as the GETCURSORS stored procedure, from your ODP application.

To run a stored procedure:

1. Run the application.
A Form1 window appears.
2. In the Form1 window, enter the connection information, and click **Connect**.
3. In the DataGrid object, scroll horizontally to verify that the values in the last column, DEPARTMENT_ID are only 60.

Note that the DataGrid contains the first result set from the stored procedure, which matches the query of the EMPLOYEES table.



4. Close the application.

Developing and Deploying .NET Stored Procedures

This chapter contains:

- [Overview of .NET Stored Procedures](#)
- [Starting the Common Language Runtime Service](#)
- [Creating a Connection as SYSDBA](#)
- [Creating an Oracle Project](#)
- [Creating .NET Stored Functions and Procedures](#)
- [Deploying .NET Stored Functions and Procedures](#)
- [Running .NET Stored Functions and Procedures](#)
- [Running .NET Stored Procedure in a Query Window](#)

Overview of .NET Stored Procedures

.NET stored procedures are methods or procedures written in a .NET language which contains SQL or PL/SQL statements.

You can write custom stored procedures and functions using any .NET compliant language, such as C# and VB.NET, and use these .NET stored procedures in the database, in the same manner as other PL/SQL or Java stored procedures. .NET stored procedures can be called from PL/SQL packages, procedures, functions, and triggers; from SQL statements, or from anywhere a PL/SQL procedure or function can be called.

Oracle Database Extensions for .NET (a database option that allows you to write .NET stored procedures) must be installed and configured in the database to run the examples in this chapter.

This chapter discusses how to use and deploy .NET stored procedures in your application.

Starting the Common Language Runtime Service

To use .NET stored procedures, you must first start the common language runtime agent, represented by the `OraClrAgent` service. This service may not start by default. Note that it is located on the Oracle database, not on the client.

Note: OraClrAgnt can be accessed through the Services Control Panel, as OracleOracleHomeNameClrAgnt, where OracleHomeName represents your Oracle home.

To start the common language runtime service:

1. From the **Start** menu, select **All Programs**, then select **Administrative Tools**, and finally, select **Services**.
2. In the Services window, click the **Extended** tab.
Scroll down the list of Services, and select OracleOracleHomeNameClrAgnt.
3. Click the **Start** hyperlink.
The Service Control window shows that the OracleClrAgent is starting.
4. When the Service Control window closes, note that the status of the OracleClrAgent is changed to Started.

Creating a Connection as SYSDBA

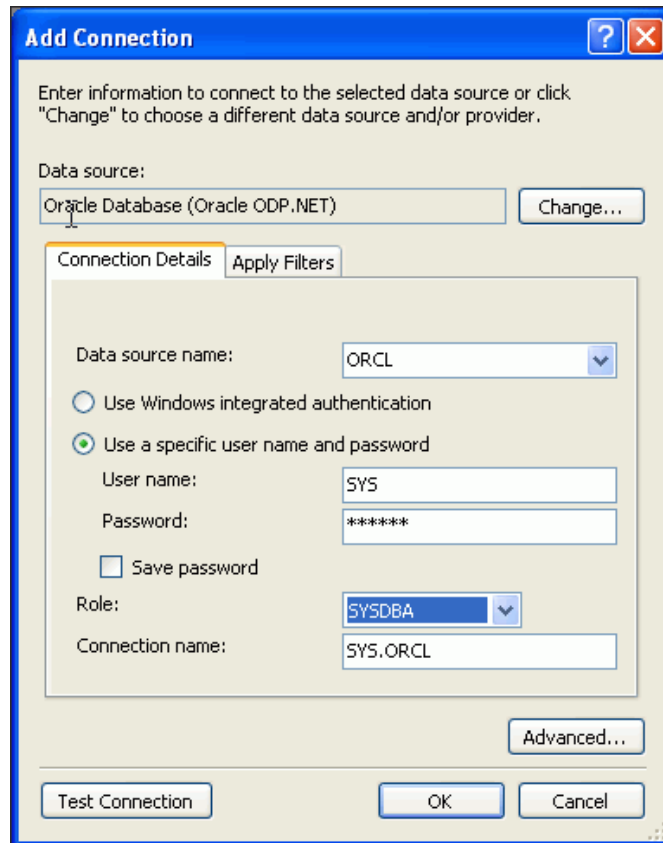
Next, you must create a database connection as SYSDBA which enables you to deploy your Oracle Project.

Note: You must have administrative privileges as SYSDBA to perform this task.

To create a database connection:

1. Follow Steps 1 to 10, in [Chapter 5, "Using Oracle Developer Tools for Visual Studio"](#) with the following exceptions.
2. In the Add Connection window, use the following:
 - For **User name**, enter *sys*.
 - For **Password**, enter the password set by the administrator who unlocked and set up the *sys* account.
To use the Enterprise Manager to set the *sys* account password, see About Administrative Accounts and Privileges in the *Oracle Database 2 Day DBA*.
 - Ensure that the **Role** is set to Sysdba.

The **Connection name** is generated automatically from the **Data source name** and the **User name** values.



The Server Explorer window should now contain the `SYS.ORCL` connection.

To use the Enterprise Manager to set the `sys` account password, see *About Administrative Accounts and Privileges* in the *Oracle Database 2 Day DBA*.

Creating an Oracle Project

To use stored procedures in .NET, you must first create a new Oracle Project to hold the stored procedures.

To create a project for .NET stored procedures:

1. From the **File** menu, select **New**, and then select **Project**.

A New Project dialog box appears.

2. In Project Types, select the type of project you are creating:

- **Visual C#:**

Visual C#, then select **Database**, and under Templates: **Oracle Project**

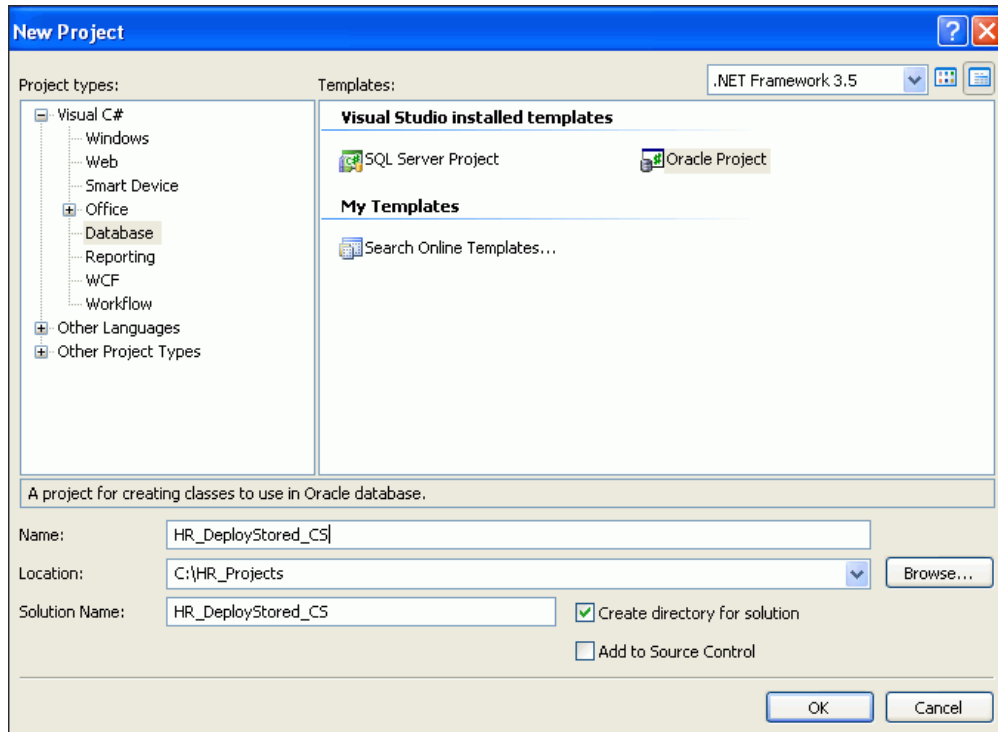
Enter **Name:** `HR_DeployStored_CS`.

- **Visual Basic:**

Other Languages, then select **Visual Basic** and **Database**, then under Templates: **Oracle Project**

Enter **Name:** `HR_DeployStored_VB`.

3. Enter **Location**: C:\HR_Projects.
4. Click **OK**.



Creating .NET Stored Functions and Procedures

You are now ready to create a .NET stored procedure.

To create a .NET stored procedure:

1. In Solution View, select the **Class1.cs** or **Class1.vb** tab in your project.
2. Add these namespace directives for the specific language, as described in ["Adding Namespace Directives"](#) on page 3-5.

Visual C#:

```
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;
```

Visual Basic:

```
Imports Oracle.DataAccess.Client
Imports Oracle.DataAccess.Types
```

3. Add Reference to `Oracle.DataAccess.dll` as described in ["Adding a Reference"](#) on page 3-4.
4. Copy the `getDepartmentno()` method into the `Class1` declaration, as indicated

Visual C#

```
public static int getDepartmentno(int employee_id)
```

```

{
    int department_id = 0;

    // Get a connection to the db
    OracleConnection conn = new OracleConnection();
    conn.ConnectionString = "context connection=true";
    conn.Open();

    // Create and execute a command
    OracleCommand cmd = conn.CreateCommand();
    cmd.CommandText = "select department_id from employees where employee_id =
:1";
    cmd.Parameters.Add(":1", OracleDbType.Int32, employee_id,
        ParameterDirection.Input);
    OracleDataReader rdr = cmd.ExecuteReader();

    while(rdr.Read())
        department_id=rdr.GetInt32(0);

    rdr.Close();
    cmd.Dispose();

    // Return the employee's department number
    return department_id;
}

```

Visual Basic:

```

Public Shared Function getDepartmentno(ByVal employee_id As Integer) As Integer
    Dim department_id As Integer = 0

    ' Get a connection to the db
    Dim conn As OracleConnection = New OracleConnection()
    conn.ConnectionString = "context connection=true"
    conn.Open()

    ' Create and execute a command
    Dim cmd As OracleCommand = conn.CreateCommand()
    cmd.CommandText = "select department_id from employees where employee_id =
:1"
    cmd.Parameters.Add(":1", OracleDbType.Int32, employee_id,
        ParameterDirection.Input)
    Dim rdr As OracleDataReader = cmd.ExecuteReader()

    While rdr.Read()
        department_id = rdr.GetInt32(0)

    End While

    rdr.Close()
    cmd.Dispose()

    ' Return the employee's department number
    Return department_id

End Function

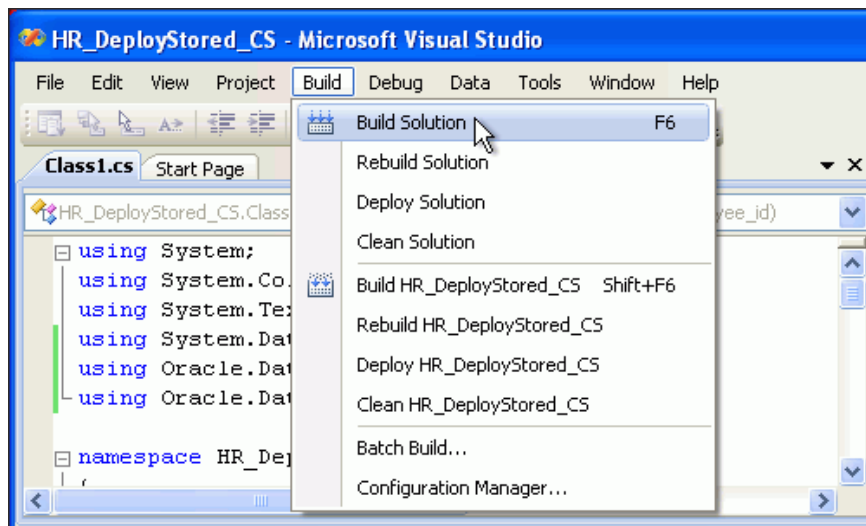
```

```

public class Class1
{
    public static int getDepartmentno(int employee_id)
    {
        int department_id = 0;

        // Get a connection to the db
        OracleConnection conn = new OracleConnection();
        conn.ConnectionString = "context connection=true";
    }
}
    
```

5. Save Class1.
6. From the **Build** menu, select **Build Solution**.



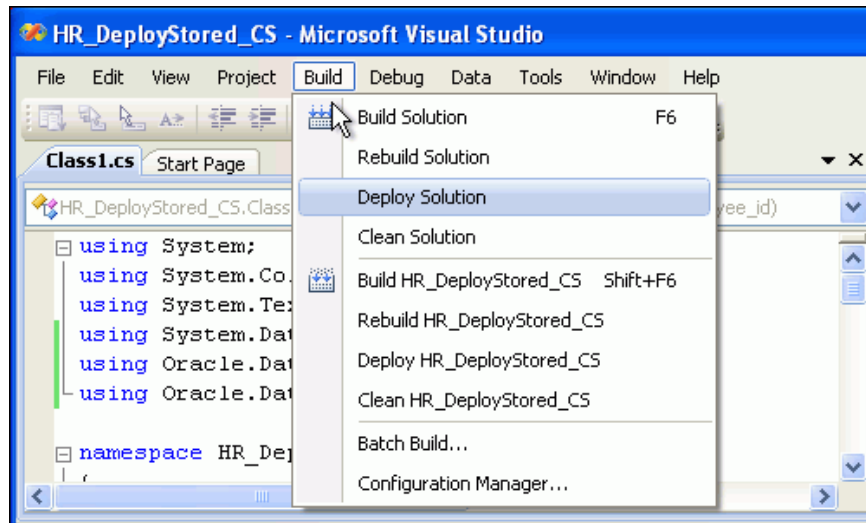
7. Check that the Output window indicates a successful build and close it.

Deploying .NET Stored Functions and Procedures

You can now deploy the .NET stored procedure that you created "[Creating .NET Stored Functions and Procedures](#)" on page 7-4.

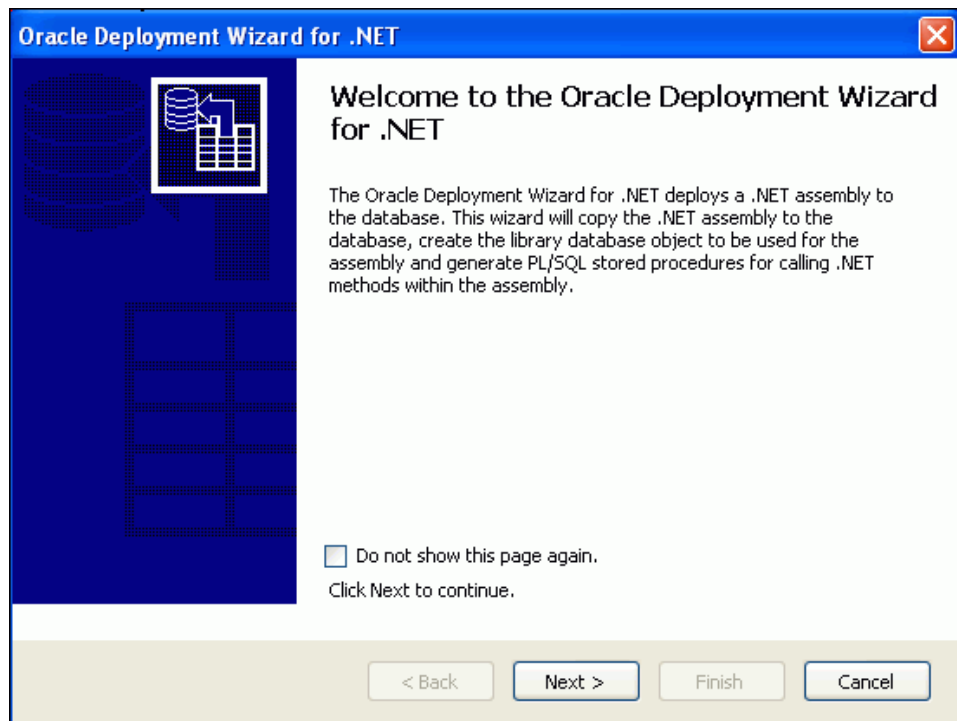
To deploy a .NET stored procedure:

1. From the **Build** menu, select **Deploy Solution**.

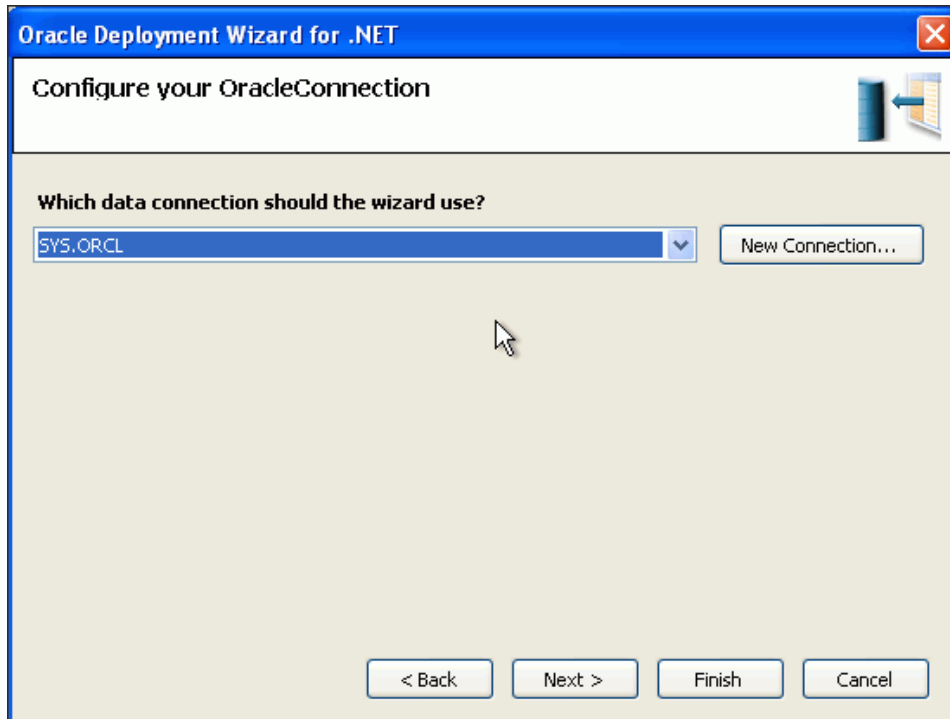


An Oracle Deployment Wizard for .NET window appears.

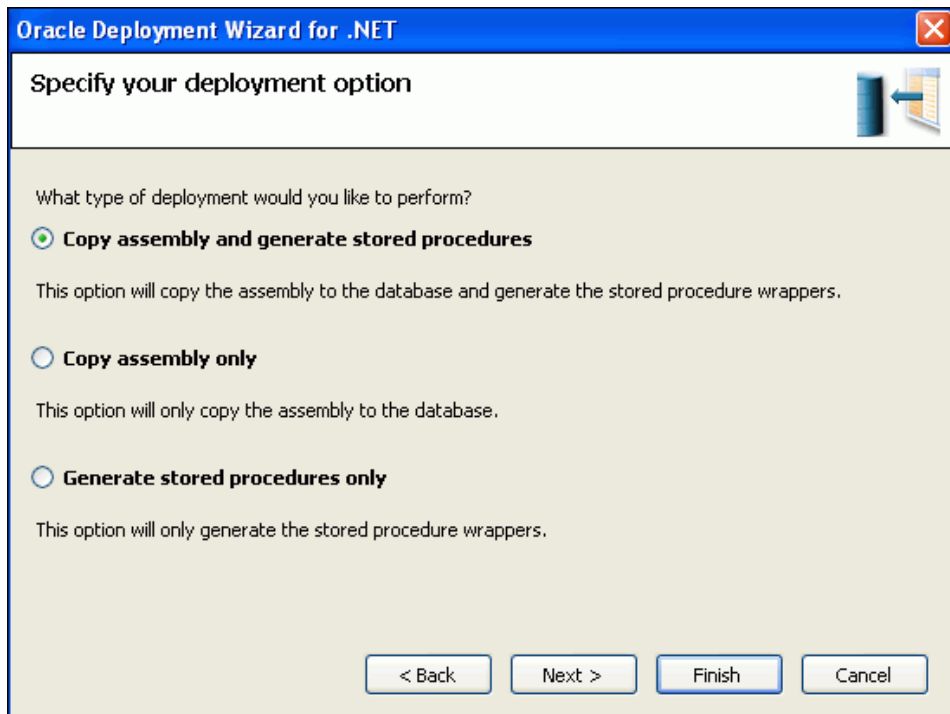
2. In the Oracle Deployment Wizard for .NET window, click **Next**.



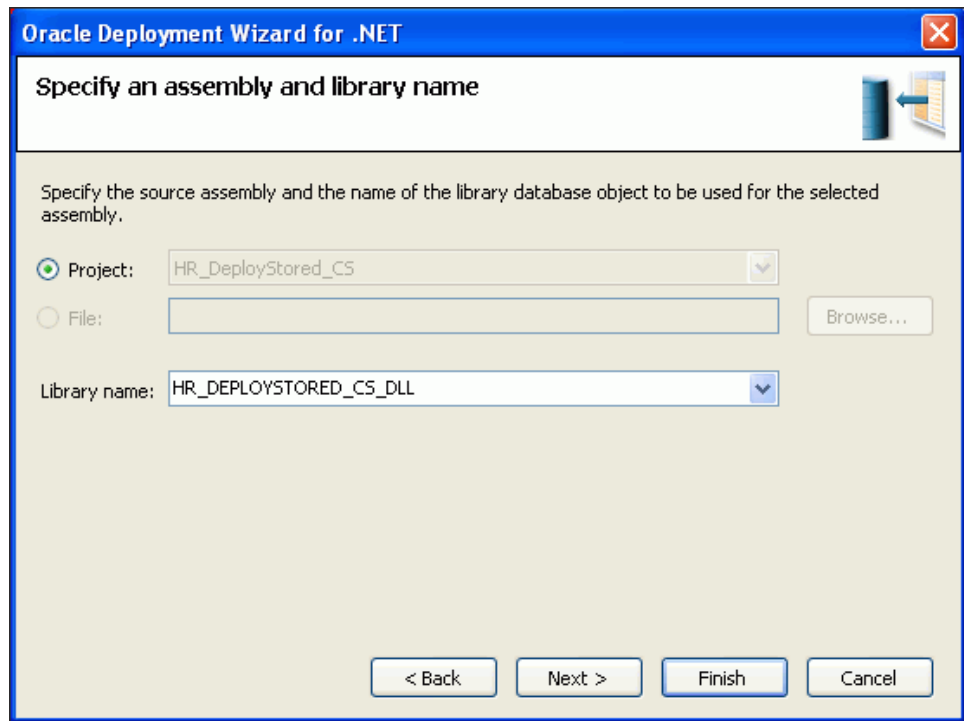
3. On the **Configure your OracleConnection** window, click **Next**.



4. On the Specify your deployment option window, ensure that the first option, **Copy assembly and generate stored procedures** is selected, and click **Next**.



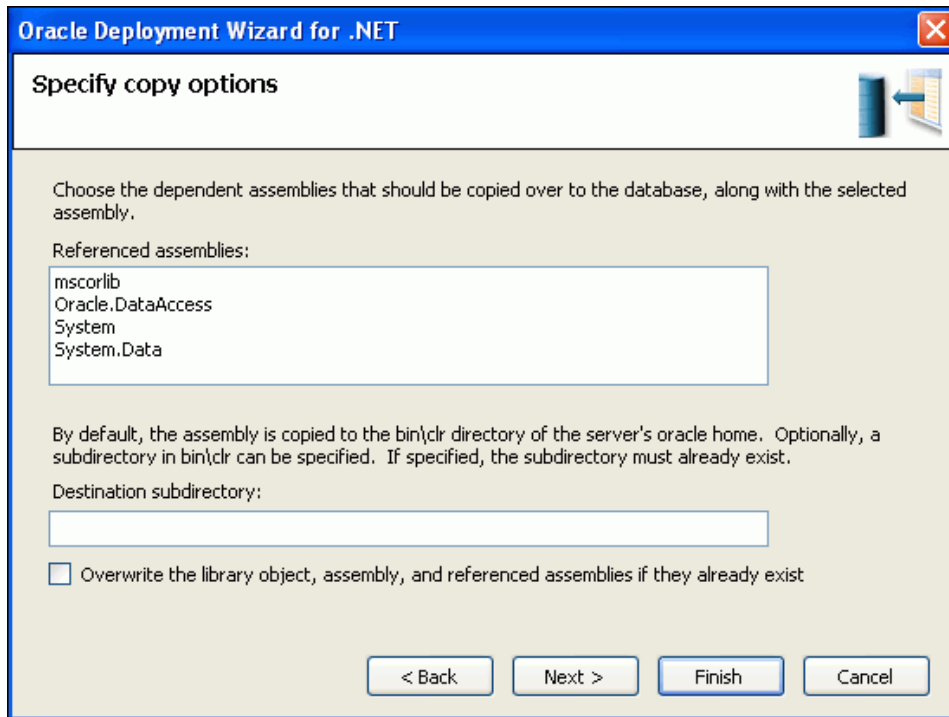
5. On the Specify an assembly and library name window, accept the defaults and click **Next**.



6. On the Specify copy options window, accept the defaults and click **Next**.

Visual Basic:

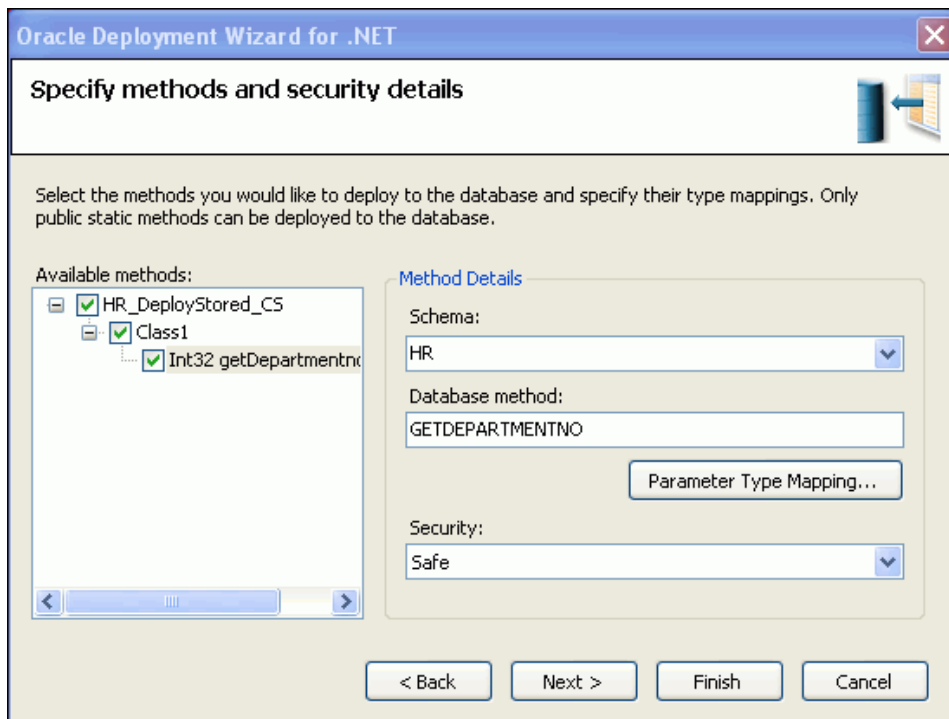
If you are using Visual Basic, the `Microsoft.VisualBasic` assembly also appears as a referenced assembly.



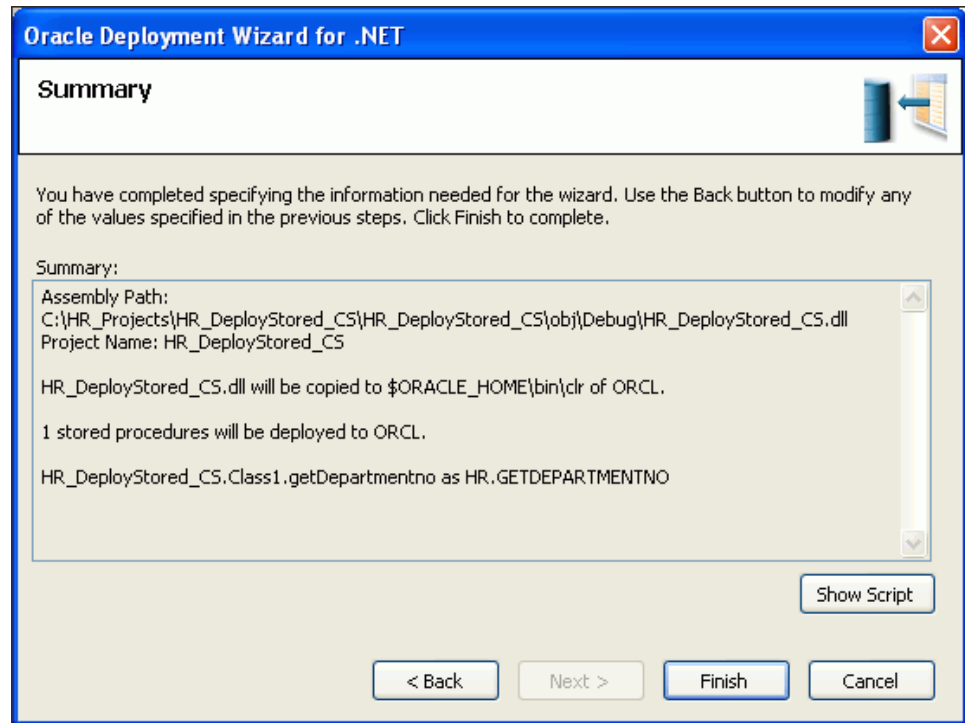
7. On the Specify methods and security details window, under Available methods, expand **HR_DeployStored_CS** or **HR_DeployStored_VB**, then expand **Class1**, and select the **getDepartmentno ()** method.

Under Method Details, select **HR** from the Schema list.

Click **Next**.



- On the Summary window, click **Finish**.

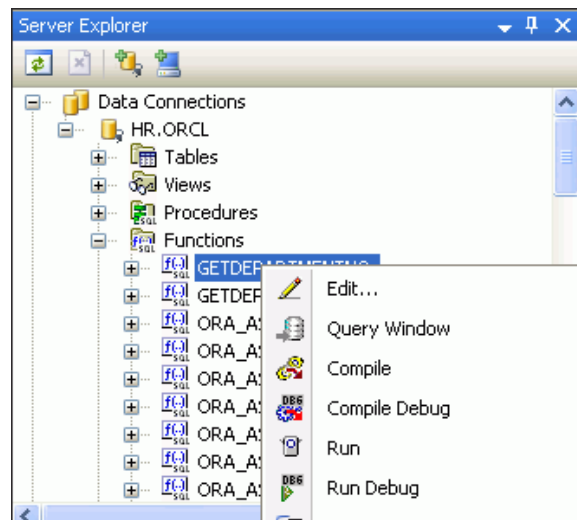


Running .NET Stored Functions and Procedures

You are now ready to run the .NET stored procedure you deployed earlier.

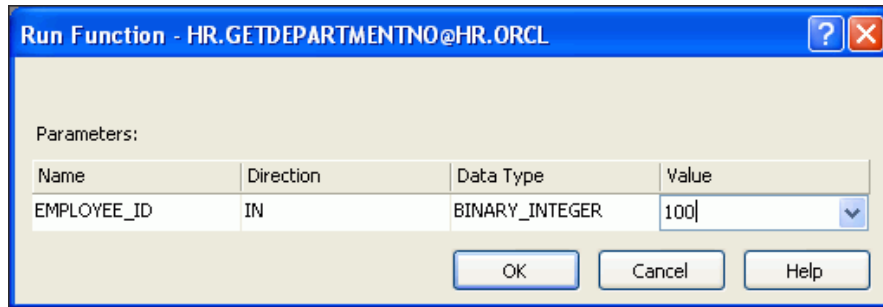
To run a .NET stored procedure:

- In Server Explorer, open and expand the HR . ORCL connection. Expand Functions. Right-click GETDEPARTMENTNO and select **Run**.

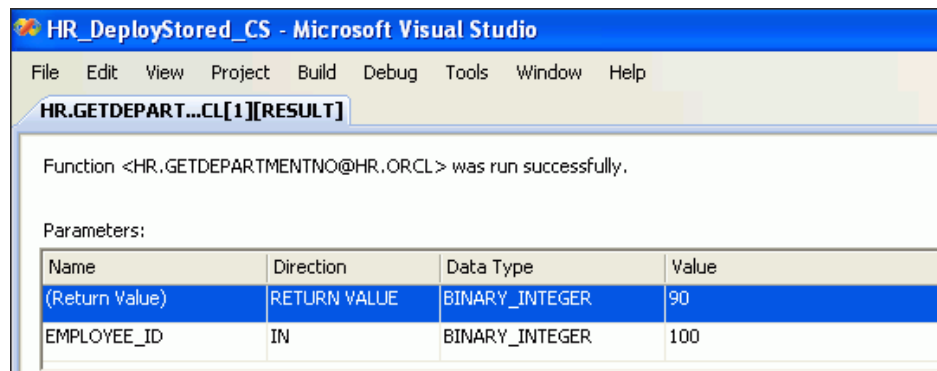


The Run Function window appears.

- In the Run Function window, enter a Value of 100 for `EMPLOYEE_ID`.
Click **OK**.



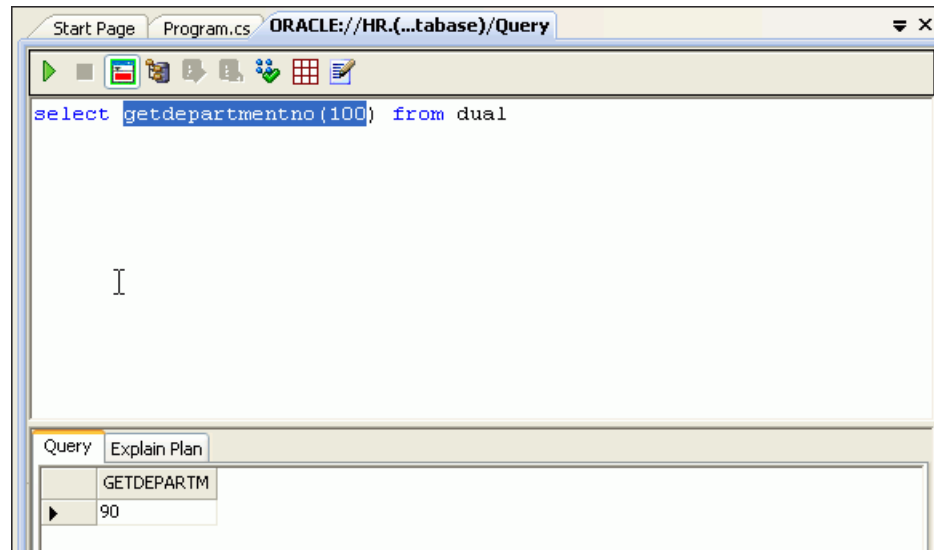
- Note that the return value for department is 90, indicating that `EMPLOYEE_ID` 100 is in department 90.



Running .NET Stored Procedure in a Query Window

You can run the .NET stored procedure that you have just created using the ODT Query Window, in addition to running it from Server Explorer.

- Open the Server Explorer in the `HR. ORCL` schema.
- Expand Functions and select `GETDEPARTMENTNO`.
- Right-click and select Query Window.
- Enter `Select getdepartmentno(100) from dual.`
- Click Execute from the toolbar.



Including Globalization Support

This chapter contains:

- [Introduction to Global Applications](#)
- [Developing Global Applications with the .NET Framework](#)
- [Presenting Data in the Correct User Local Convention](#)
- [Synchronizing the .NET and Oracle Database Locale Environments](#)
- [Client Globalization Support in Oracle Data Provider for .NET](#)

See Also:

- Chapter 8, "Oracle Data Provider for .NET Globalization Classes" in *Oracle Data Provider for .NET Developer's Guide*
- "Working in a Global Environment" in the *Oracle Database 2 Day Developer's Guide*
- Microsoft .NET Internationalization Internet site, <http://www.microsoft.com/globaldev/getwr/>

Introduction to Global Applications

This chapter discusses global application development with Oracle Database in .NET. It addresses the basic tasks associated with developing applications that are ready for global deployment, such as developing locale awareness and presenting data with cultural conventions of the user's locale. It also discusses globalization support features available in Oracle Data Provider for .NET.

Building a global-ready application that supports different locales requires good development practices.

A locale refers to a national language and the region in which the language is spoken. The application itself must be aware of the user's locale preference and be able to present content following the cultural convention expected by the user. It is important to present data with appropriate locale characteristics, such as the correct date and number formats. Oracle Database is fully internationalized to provide a global platform for developing and deploying global applications.

Developing Global Applications with the .NET Framework

When planning a global-ready application, you have to consider two main tasks:

- **Globalization** is the process of designing applications that can adapt to different cultures.

- **Localization** is the process of translating resources for a specific culture.

In the .NET Framework, the `System.Globalization` namespace contains classes that define information related to culture, such as language, country and region, calendars, format patterns for dates, currency, and numbers, and the sort order for strings. These classes simplify the process of developing a global-ready application, so that passing a `CultureInfo` object that represents the user's culture to methods in `System.Globalization` namespace initiates the correct set of rules and data.

The .NET Framework also supports the creation and localization of resources, and offers a model for packaging and deploying them. Localizing the application's resources for specific cultures supports development of translated versions of the application. The .NET Framework base class library provides several classes in the `System.Resources` namespace for building and manipulating application resources.

Presenting Data in the Correct User Local Convention

Data in the application must be presented in a way that meets the user's expectations, or its meaning can be misinterpreted. For example, 12/11/05 implies December 11, 2005 in the United States and November 12, 2005 in the United Kingdom. Similar confusion exists for number and monetary formats. For example, the period (.) is a decimal separator in the United States and a thousand separator throughout Europe.

Different languages have their own sorting rules: some languages are collated according to the letter sequence in the alphabet, others according to stroke count in the letter, still others are ordered by the pronunciation of the words. Presenting data that is not sorted according to the linguistic sequence that the user is accustomed to can make searching for information difficult and time-consuming.

Depending on the application logic and the volume of data retrieved from the database, it may be more appropriate to format the data at the database level rather than at the application level. Oracle Database offers many features that refine the presentation of data when the user locale preference is known.

Connecting to SQL*Plus

Several of the following examples require that you use SQL*Plus to connect as a user with database administrator privileges such as `SYS` or `SYSTEM`.

See Also: "Locking and Unlocking User Accounts" in the *Oracle Database 2 Day DBA* for further information

Using Oracle Date Formats

There are three different date presentation formats in Oracle Database: standard, short, and long. The following steps illustrate the difference between the short and long date formats for United States and Germany.

To change the Oracle date format:

1. From a Windows command prompt, enter the following

```
C:\>sqlplus "sys as sysdba"
Enter password:passwd
```

where *passwd* is the `Sys` password that was established when the database was installed. The password does not appear when you type the characters.

```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"

C:\>sqlplus "sys/as sysdba"

SQL*Plus: Release 11.1.0.6.0 - Production on Tue Apr 1 10:43:59 2008
Copyright (c) 1982, 2007, Oracle. All rights reserved.

Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Oracle Label Security, OLAP, Data Mining,
Oracle Database Vault and Real Application Testing options

SQL> _

```

2. Enter this command at the SQL prompt:

```
SQL> ALTER SESSION SET NLS_TERRITORY=america NLS_LANGUAGE=american;
```

This message appears: Session altered.

There is no problem with setting a parameter to its current setting. You may want to do this for security. To determine what your current settings are enter:

```
SQL> select * from v$nls_parameters;
```

or

```
select * from v$nls_parameters where parameter = 'NLS_LANGUAGE';
```

3. At the SQL prompt, enter the following query:

```
SQL> SELECT employee_id "ID",
       SUBSTR (first_name,1,1)||'. '||last_name "Name",
       TO_CHAR (hire_date, 'DS') "Short Hire",
       TO_CHAR (hire_date, 'DL') "Long Hire Date"
FROM hr.employees
WHERE employee_id < 105;
```

Note that you must use `hr.employees` in order to access the employees table in the hr schema because you are currently logged in as `sys`, not `hr`.

The result of the query returns in the American format specified in Step 1.

```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"

      ID Name                                Short Hire Long Hire Date
-----
    100 S. King                               6/17/1987  Wednesday, June 17, 1987
    101 N. Kochhar                             9/21/1989  Thursday, September 21, 1989
    102 L. De Haan                             1/13/1993  Wednesday, January 13, 1993
    103 A. Hunold                               1/3/1990   Wednesday, January 03, 1990
    104 B. Ernst                                5/21/1991  Tuesday, May 21, 1991

SQL>

```

4. Enter the following command at the SQL prompt:

```
SQL> ALTER SESSION SET NLS_TERRITORY=germany NLS_LANGUAGE=german;
```

This message appears: Session altered.

- At the SQL prompt, enter the query from Step 3.

The result of the query returns in the German format specified in Step 4.

```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"
-----
ID Name                               Short Hire
-----
Long Hire Date
-----
      100 S. King                       17.06.1987
Mittwoch, 17. Juni 1987
      101 N. Kochhar                     21.09.1989
Donnerstag, 21. September 1989
      102 L. De Haan                     13.01.1993
Mittwoch, 13. Januar 1993
-----
ID Name                               Short Hire
-----
Long Hire Date
-----
      103 A. Hunold                       03.01.1990
Mittwoch, 3. Januar 1990
      104 B. Ernst                        21.05.1991
Dienstag, 21. Mai 1991
SQL>

```

Using Oracle Number Formats

There are also differences in the decimal character and group separator. The following steps illustrate these difference between United States and Germany.

To change the Oracle number format:

- Enter the following command at the SQL prompt:

```
SQL> ALTER SESSION SET NLS_TERRITORY=america NLS_LANGUAGE=american;
```

This message appears: Session altered.

- At the SQL prompt, enter the following query:

```
SQL> SELECT employee_id "ID",
SUBSTR (first_name,1,1)||'. '||last_name "Name",
TO_CHAR (salary, '99G999D99') "Salary"
FROM hr.employees
WHERE employee_id < 105;
```

The result of the query returns in the American format specified in Step 1.

```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"
-----
ID Name                               Salary
-----
      100 S. King                          24,005.00
      101 N. Kochhar                       17,000.00
      102 L. De Haan                       17,000.00
      103 A. Hunold                         9,000.00
      104 B. Ernst                          6,000.00
SQL>

```

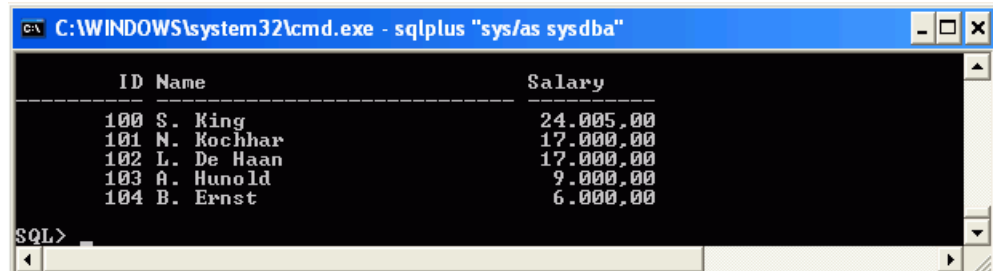
3. Enter the following command at the SQL prompt:

```
SQL> ALTER SESSION SET NLS_TERRITORY=germany;
```

This message appears: `Session altered.`

4. At the SQL prompt, enter the query in Step 2.

The result of the query returns in the German format specified in Step 3.



```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"

-----
ID Name                Salary
-----
100 S. King            24.005,00
101 N. Kochhar         17.000,00
102 L. De Haan         17.000,00
103 A. Hunold           9.000,00
104 B. Ernst           6.000,00

SQL>

```

Using Oracle Linguistic Sorts

Spain traditionally treats *ch*, *ll*, and *ñ* as letters of their own, ordered after *c*, *l* and *n*, respectively. The following steps illustrate the effect of using a Spanish sort against the employee names Chen, Chung, and Colmenares.

To change the Oracle linguistic sort:

1. Enter the following command at the SQL prompt.

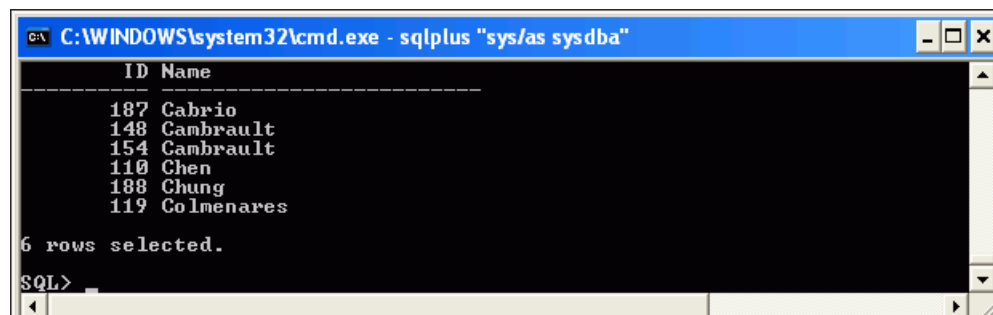
```
SQL> ALTER SESSION SET NLS_SORT=binary;
```

This message appears: `Session altered.`

2. At the SQL prompt, enter the following query:

```
SQL> SELECT employee_id "ID",
           last_name "Name"
FROM hr.employees
WHERE last_name LIKE 'C%'
ORDER BY last_name;
```

The result of the query returns in the binary sort specified in Step 1.



```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"

ID Name
-----
187 Cabrio
148 Cambrault
154 Cambrault
110 Chen
188 Chung
119 Colmenares

6 rows selected.

SQL>

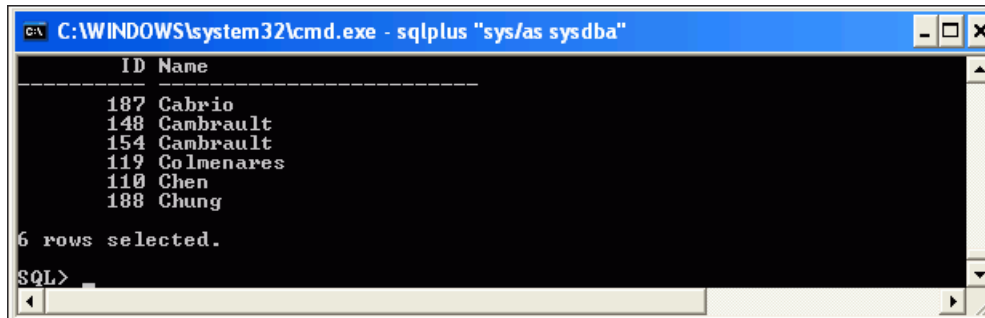
```

3. Enter the following command at the SQL prompt.

```
SQL> ALTER SESSION SET NLS_SORT=spanish_m;
```

This message appears: `Session altered.`

4. At the SQL prompt, enter the query in Step 2.
5. The result of the query returns in the Spanish sort specified in Step 3.



```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"
-----
ID Name
-----
187 Cabrio
148 Cambrault
154 Cambrault
119 Colmenares
110 Chen
188 Chung

6 rows selected.
SQL>

```

Oracle Error Messages

The `NLS_LANGUAGE` parameter also controls the language of the database error messages. Setting this parameter prior to submitting a SQL query ensures the return of local language-specific error messages, as shown in these steps:

To change the Oracle NLS language parameter:

1. Enter the following command at the SQL prompt.

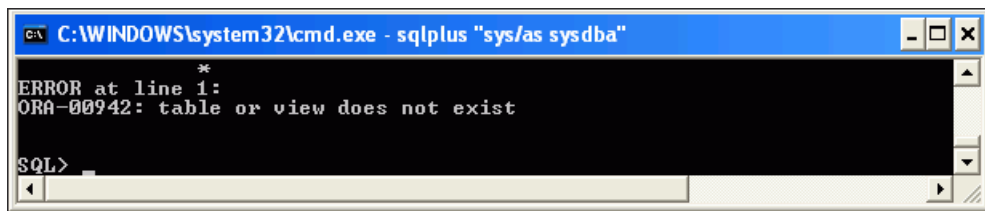
```
SQL> ALTER SESSION SET NLS_LANGUAGE=american;
```

This message appears: `Session altered.`

2. At the SQL prompt, enter the following query.

```
SQL> SELECT * FROM managers;
```

The result of the query return the error message in the language specified in Step 1.



```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"
*
ERROR at line 1:
ORA-00942: table or view does not exist

SQL>

```

3. Enter the following command at the SQL prompt.

```
SQL> ALTER SESSION SET NLS_LANGUAGE=french;
```

This message appears: `Session altered.`

4. At the SQL prompt, enter the query in Step 2.

The result of the query returns the error message in the language specified in Step 3.

```

C:\WINDOWS\system32\cmd.exe - sqlplus "sys/as sysdba"
*
ERROR at line 1:
ORA-00942: Table ou vue inexistante

SQL>

```

5. Reset your language, local, and sort settings back to their original values.

Synchronizing the .NET and Oracle Database Locale Environments

When you are developing global applications, always synchronize the user locale settings between the database and clients. Otherwise, the application may present conflicting culture-sensitive information. For example, a .NET application must map the Culture ID of the application user to the correct NLS_LANGUAGE and NLS_TERRITORY parameter values before performing SQL operations.

Table 8–1 shows some of the more common locales, as defined in .NET and Oracle environments.

Table 8–1 Common NLS_LANGUAGE and NLS_TERRITORY Parameters

Culture	Culture ID	NLS_LANGUAGE	NLS_TERRITORY
Chinese (P.R.C.)	zh-CN	SIMPLIFIED CHINESE	CHINA
Chinese (Taiwan)	zh-TW	TRADITIONAL CHINESE	TAIWAN
English (U.S.A.)	en-US	AMERICAN	AMERICA
English (U.K.)	en-GB	ENGLISH	UNITED KINGDOM
French (Canada)	fr-CA	CANADIAN FRENCH	CANADA
French (France)	fr-FR	FRENCH	FRANCE
German	de	GERMAN	GERMANY
Italian	it	ITALIAN	ITALY
Japanese	ja	JAPANESE	JAPAN
Korean	ko	KOREAN	KOREA
Portuguese (Brazil)	pt-BR	BRAZILIAN PORTUGUESE	BRAZIL
Portuguese	pt	PORTUGUESE	PORTUGAL
Spanish	es	SPANISH	SPAIN

Client Globalization Support in Oracle Data Provider for .NET

Oracle Data Provider for .NET enables applications to manipulate culture-sensitive data, such as ensuring proper string format, date, time, monetary, numeric, sort order, and calendar support using culture conventions defined in the Oracle Database. The default globalization settings are determined by the client's NLS_LANG parameter, which is defined in the Windows Registry of the local computer. When the `OracleConnection.Open` method establishes a connection, it implicitly opens a session with globalization parameters specified by the value of the NLS_LANG parameter.

Client Globalization Settings

The client globalization parameter settings are read-only and remain constant throughout the lifetime of the application. Changing the `OracleGlobalization` object properties does not change the globalization settings of the session or the thread. The following sections describe how to modify the globalization settings at the session and thread level.

Your .NET application can obtain globalization settings by calling the `OracleGlobalization.GetClientInfo()` static method. The `OracleGlobalization` sample code below demonstrates how to obtain some of the values in .NET.

Visual C#:

```
using System;
using Oracle.DataAccess.Client;

class ClientGlobalizationSample
{
    static void Main()
    {
        OracleGlobalization ClientGlob = OracleGlobalization.GetClientInfo();
        Console.WriteLine("Client machine language: " + ClientGlob.Language);
        Console.WriteLine("Client characterset: " + ClientGlob.ClientCharacterSet);
    }
}
```

Visual Basic:

```
Imports System
Imports Oracle.DataAccess.Client

Class ClientGlobalizationSample
    Shared Sub Main()
        Dim ClientGlob As OracleGlobalization = OracleGlobalization.GetClientInfo()
        Console.WriteLine("Client machine language: " + ClientGlob.Language)
        Console.WriteLine("Client characterset: " + ClientGlob.ClientCharacterSet)
    End Sub
End Class
```

Using Session Globalization Settings

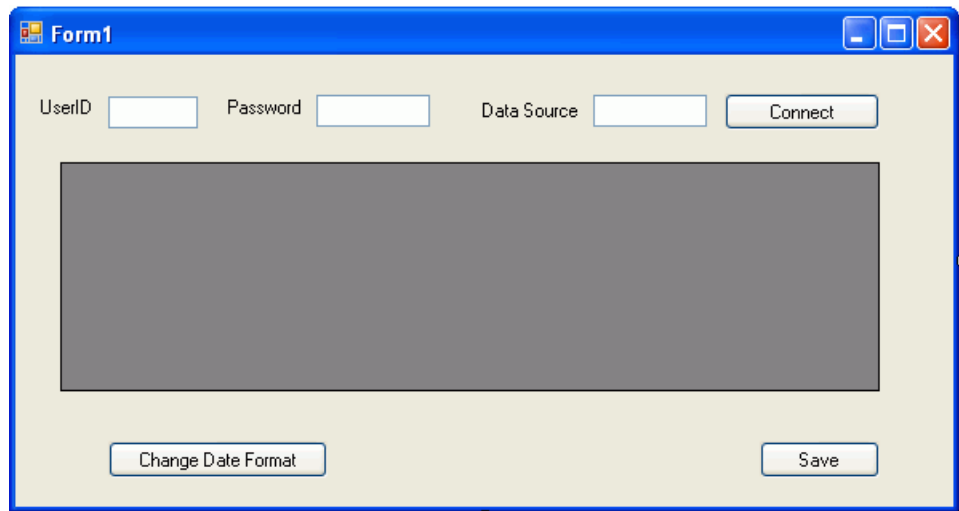
Session globalization parameters are initially identical to client globalization settings, but they can be modified. To modify the session parameters, you must establish a connection to the database, and then call the `GetSessionInfo()` method of an `OracleConnection` object to retrieve the session globalization settings. Next, you modify the globalization settings as needed, then save the settings back to the `OracleConnection` object through the `SetSessionInfo(OracleGlobalization)` method.

To specify the globalization session setting:

1. Open the application `HR_Connect_CS` or `HR_Connect_VB`.
2. Make a copy of `Form3 .xx`, which you finished at the end of [Chapter 4](#) and name it `Form5 .xx`, following the instructions in [Appendix B, "Copying a Form"](#).
3. Open **Form1** of the project, and switch to design view.
4. From the **View** menu, select **Toolbox**.

5. From the Toolbox, under Windows Forms, drag and drop a **Button** onto Form1.
6. Right-click the new **Button**, select **Properties**. The Properties window appears.
7. In the Properties window, set these properties:
 - Under Appearance, change **Text** to Change Date Format.
 - Under Design, change (**Name**) to date_change.

Form1 should look much like this:



In the properties window, if you click **Events** (lightning bolt icon), `date_change_Click()` now shows as the Event for the date button.

8. Open the new `date_change_Click()` method just created and add the following code to change the date format from the standard DD-MON-RR to YYYY-MM-DD and to update the `DataSet`.

Visual C#:

```
si.DateFormat = "YYYY-MM-DD";
conn.SetSessionInfo(si);

ds.Clear();
da.Fill(ds);
departments.DataSource = ds.Tables[0];
```

Visual Basic:

```
si.DateFormat = "YYYY-MM-DD"
conn.SetSessionInfo(si)

ds.Clear()
da.Fill(ds)
departments.DataSource = ds.Tables(0)
```

Note that the `ds.Clear()` call will clear the old results before posting the changed data.

Also, the `si` class variable will be declared and session globalization information retrieved in Step 10 and Step 11.

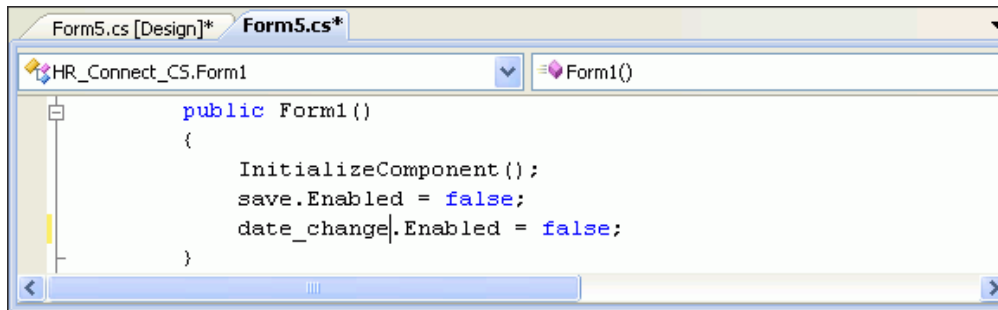
9. Within the appropriate method, add the code indicated.

Visual C#: In the Form1() method

```
date_change.Enabled = false;
```

Visual Basic: In the Form1_Load method

```
date_change.Enabled = false
```



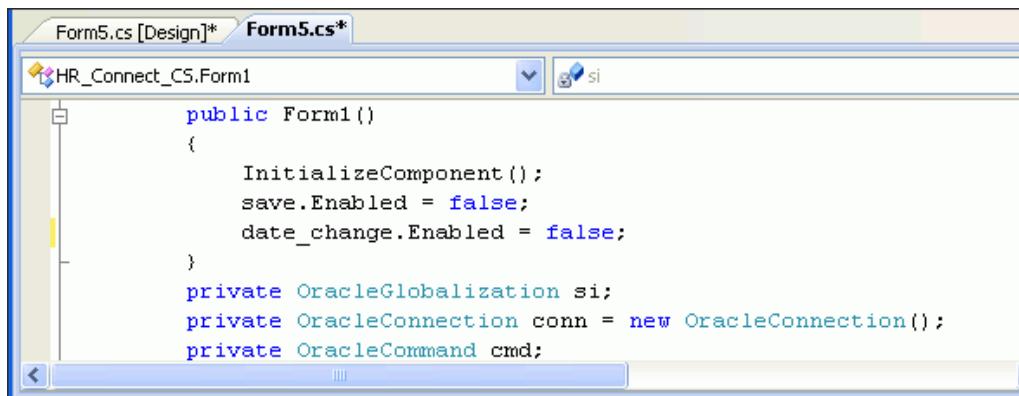
10. Add the following class variable to the existing Form1 class declarations right after the public Form1() block with this code as indicated.

Visual C#:

```
private OracleGlobalization si;
```

Visual Basic:

```
private si As OracleGlobalization
```



11. Within the connect_Click() method try block, add the indicated code which does the following:

- Retrieve the value of the OracleGlobalization object.
- Retrieve data from the EMPLOYEES table (note the new query).
- Enable the **Change Date Format** button.

The changed code is in bold typeface.

Visual C#:

```

conn.Open();
connect.Enabled = false;

```

```

si = conn.GetSessionInfo();

string sql = "select employee_id, first_name, last_name, TO_CHAR(hire_date)" +
    " \"Hire Date\" from employees where employee_id < 105";
cmd = new OracleCommand(sql, conn);
cmd.CommandType = CommandType.Text;

da = new OracleDataAdapter(cmd);
cb = new OracleCommandBuilder(da);
ds = new DataSet();

da.Fill(ds);

departments.DataSource = ds.Tables[0];

save.Enabled = true;
date_change.Enabled = true;

```

Visual Basic:

```

conn.Open()
connect.Enabled = false

si = conn.GetSessionInfo()

Dim sql As String = "select employee_id, first_name, last_name, " & _
    "TO_CHAR(hire_date) \"Hire Date\" from employees where employee_id < 105"
cmd = new OracleCommand(sql, conn)
cmd.CommandType = CommandType.Text

da = new OracleDataAdapter(cmd)
cb = new OracleCommandBuilder(da)
ds = new DataSet()

da.Fill(ds)

departments.DataSource = ds.Tables[0]

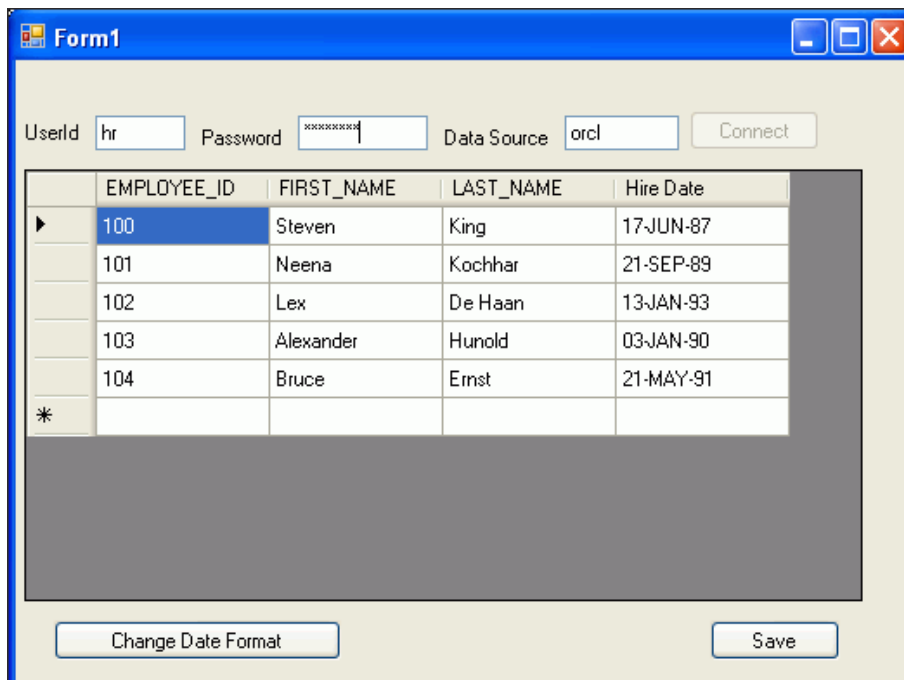
save.Enabled = true
date_change.Enabled = true

```

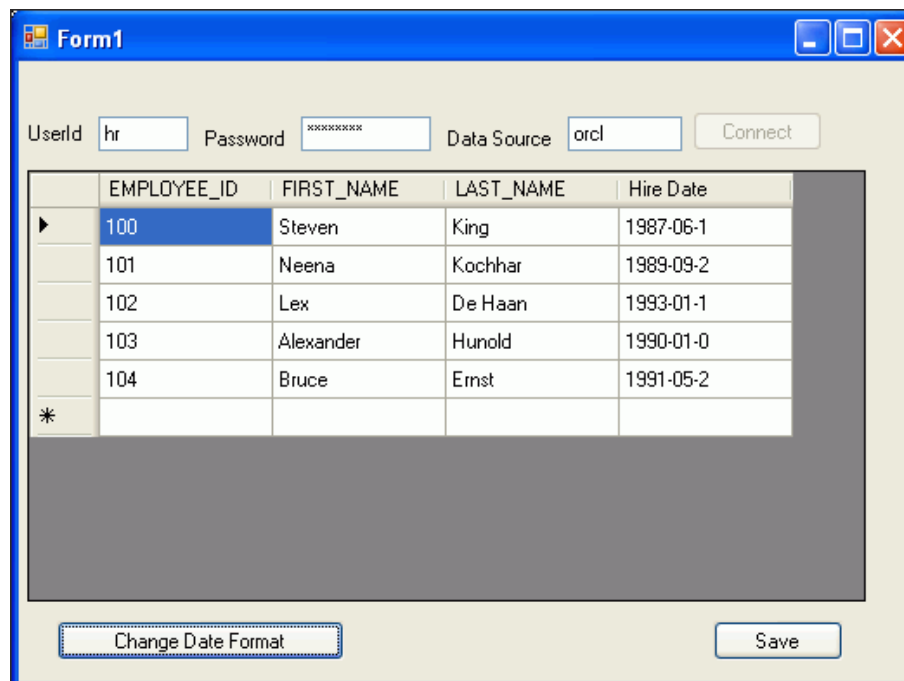
12. Save Form1.

13. Run the application using the F5 keyboard shortcut.

The application successfully connects to the database so the data grid is populated with the results of the query.



14. Click **Change Date Format**.



Note that the date format changed from the original DD-MON-RR to YYYY-MM-DD.

15. Close the application.

Thread-Based Globalization Settings

Thread-based globalization parameter settings are specific to each thread. Initially, these settings are identical to the client globalization parameters, but they can be changed programmatically. When converting ODP.NET Types to and from strings, use the thread-based globalization parameters, if applicable.

Thread-based globalization parameter settings are obtained by calling the `GetThreadInfo()` static method of the `OracleGlobalization` class. A call to `SetThreadInfo()` static method sets the globalization settings of the thread.

ODP.NET classes and structures rely solely on the `OracleGlobalization` settings when manipulating culture-sensitive data. They do not use .NET thread culture information. If the application uses only .NET types, `OracleGlobalization` settings have no effect. However, when conversions are made between ODP.NET Types and .NET Types, `OracleGlobalization` settings are used where applicable.

Note: Changes to the `System.Threading.Thread.CurrentCulture.CurrentCulture` property do not impact the `OracleGlobalization` settings of the thread or the session. The reverse is also true.

Starting and Stopping an Oracle Database Instance

You may need to frequently stop and restart the database.

To start an Oracle Database Instance:

1. From the **Start** button, select **Programs**, then **Administrative Tools**, then **Services**, and select **OracleServiceDatabaseName** where DatabaseName is the service_name of the database as indicated in the `tnsnames.ora` file. See "[Configuring a NET Connect Alias](#)" on page 2-7 for further details.
2. In the left panel, click the link to Start the service.
3. The database services will begin to start. You should see a **Start Database** window. Do not proceed with the following steps until it indicates that the "OracleService service was started successfully".

To stop an Oracle Database Instance:

1. From the **Start** button, select **Programs**, then **Administrative Tools**, then **Services**, and select **OracleServiceDatabaseName**.
2. In the left panel, click the link to Stop the service.
3. The database will begin to shut down. You should see a **Stop Database** window. Do not proceed until it indicates that the "OracleService service was stopped successfully".

B

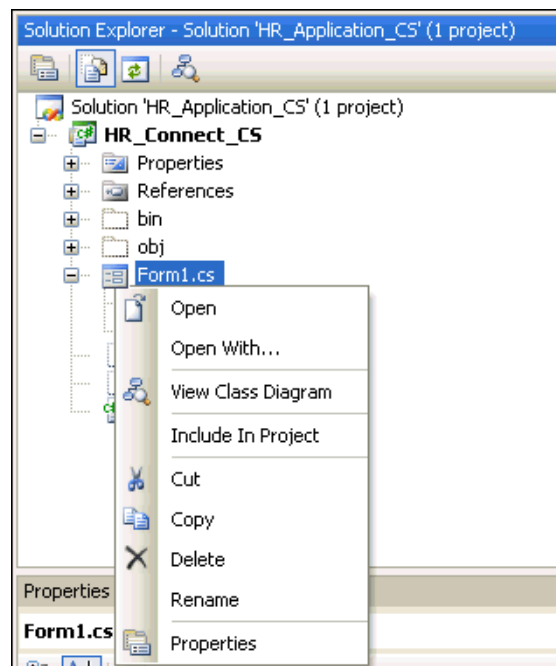
Copying a Form

Because you will be using this application to learn about various aspects of application development with Oracle, you should make copies of your form for reuse.

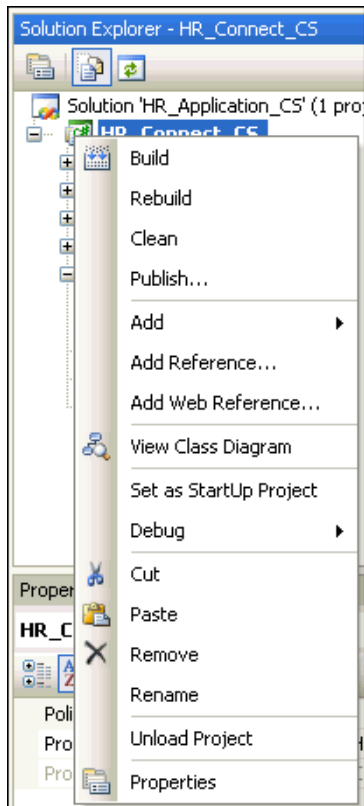
To create a copy of an existing form:

1. In the Solution Explorer, right-click on `Form1.xx` or any other file you need to copy. Select **Copy**.

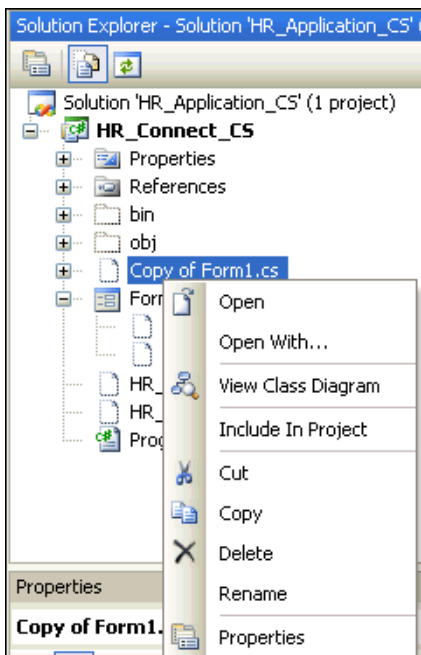
If `Form1.xx` does not appear in the Solution Explorer, from the **Project** menu, select **Show All Files**.



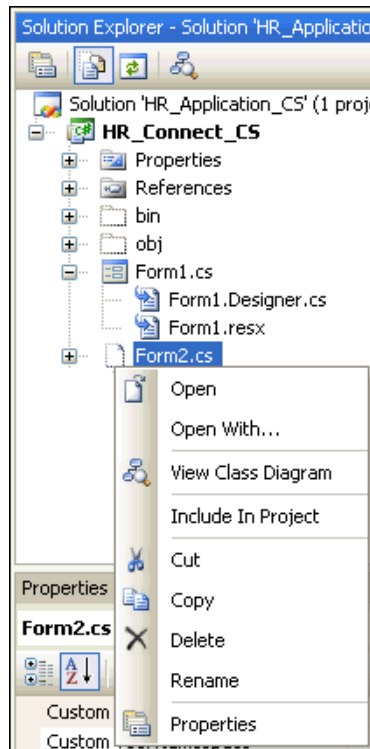
2. Right-click `HR_Connect_CS` or other project. Select **Paste**.



3. Right-click **Copy of Form1.cs**. Select **Rename**. Change the name of the form to **Form2.cs**.



4. Right-click on Form2.cs, and select **Include In Project**.



5. Right-click on `Form1.cs`, and select **Exclude From Project**.

You can include and exclude forms from the project just by reversing these steps.

Note: This process generally works smoothly, but sometimes if you have done it several times, there can be little problems. If so, try running **Rebuild Solution** from the **Build** menu.

A

account
 unlocking, 5-1,7-2
Add() method, 4-4
adding reference, 3-4
alias
 database, 5-1
ALTER TABLE, 5-12
apply filters, 5-1,7-2
automatic naming, 3-3

B

bind variable, 4-4
 definition, 4-4
 name, 4-4
 position, 4-4
 update, 4-4
binding data, 4-8
building connection, 3-11
Button control, 3-7

C

C# statement
 using, 3-6
case statement, 3-18
class variable, 4-8
click event, 4-8
client globalization settings, 8-8
CLR, 1-2
Code and Designer toggle, 3-7
Code view, 3-6
command
 database query, 4-1
 definition, 4-1
 query, 4-1
 using, 4-1
CommandType property, 4-1
Common Language Runtime
 definition, 1-2
Common Language Runtime agent, 7-1
Common Language Runtime Service
 starting, 7-1
Configure Your OracleConnection window, 7-7

connect, 5-1
connect alias, 2-8
connect project, 3-4
connecting
 as SYSDBA
 creating, 7-2
connection
 add, 7-2
 building, 3-11
 data source name, 5-1,7-2
 details, 7-2
 dispose, 4-8
 hr.(Local Database), 7-2
 name, 5-1
 new, 7-2
 opening, 3-11
 password, 5-1
 role, 5-1
 specific user name and password, 5-1
 SYSDBA, 7-7
 user name, 5-1,7-2
 user name and password, 7-2
connection control, 3-7
constraint
 add, 5-12
constraint properties, 5-10
constraints tab, 5-10
control
 Textbox, 3-7
controls, 3-7
 Button, 3-7
 DataGrid, 4-7
 Label, 3-7
 Listbox, 4-2
 toolbox, 3-7
copying a form, B-1
creating Oracle Project, 7-3
cultural conventions, 8-1
Culture ID parameter, 8-8
Culture parameter, 8-8
CultureInfo object, 8-1
culture-sensitive data, 8-8
CurrentCulture parameter, 8-14

D

- data entry control, 3-7
- data grid, 6-10
- data provider, 3-4
 - Oracle Data Provider for .NET, 1-2
- data source name, 5-1,7-2
- database error message, 3-18
- DataGrid class, 6-10
- DataGrid control, 4-7
- DataReader class, 4-7
- DataSet class, 4-8
 - updating, 8-9
- date format, 8-2
 - change, 8-9
- default role, 5-1
- deleting data, 4-13
- Design view, 5-5
- Designer, 3-7
- Designer and Code toggle, 3-7
- designing user interface, 3-7
- dialog
 - new project, 3-1
- Direction property, 4-4
- display schema, 5-1,7-2
- Dispose() method, 3-17

E

- Enterprise Manager, 2-1,7-7
- error handling
 - Try-Catch-Finally, 3-17
 - type
 - exceptions with ODP.NET, 3-16
 - ODP.NET, 3-16
 - Oracle, 3-16
- error message, 8-7
- Error property, 3-16
- events
 - click, 4-8
- examples
 - names of, 3-3
- Exception class, 3-18
- ExecuteReader() method, 4-2

F

- FCL, 1-2
- File menu, 3-1,3-2
- finally block, 4-8
- foreign key, 5-10
- form, 3-7
- Form1, 3-3
- form1.cs, 3-3
- form1.vb, 3-3
- Framework Class Libraries
 - definition, 1-2

G

- GetSessionInfo() method, 8-9

- GetThreadInfo() method, 8-14
- global application
 - development, 8-1
 - .NET framework, 8-1
- global applications
 - introduction, 8-1
- globalization
 - definition, 8-1
- globalization session information, 8-9
- globalization support
 - client, 8-8
 - ODP for .NET, 8-8

I

- Imports statement, 3-6
- In the left panel, click the link to Start the service., A-1
- index
 - add, 5-8
 - creating, 5-8
 - properties, 5-8
- Indexes tab, 5-8
- inserting data, 4-13

L

- Label control, 3-7
- linguistic sort, 8-6
- ListBox, 4-2
- local user convention, 8-2
- locale
 - definition, 8-1
 - synchronizing, 8-8
- locale awareness, 8-1
- locale characteristics
 - definition, 8-1
- localization
 - resources, 8-1

M

- memory location, 6-2
- menu
 - File, 3-2
 - file, 3-1
 - View, 3-7
- method
 - Add(), 4-4
 - Dispose(), 3-17
 - Open(), 3-11
- method parameter
 - binding, 6-10
 - definition, 6-10
- Microsoft internationalization
 - URL, 8-1
- Microsoft .NET Framework
 - definition, 1-2
- Microsoft Visual Studio, 1-3

N

- name of code file, 3-3
- name of forms, 3-3
- Name property, 3-7
- namespace directives, 3-6
- .NET assembly, 1-3
- NET connect, 2-8
- .NET languages, 1-2
- .NET stored functions and procedures
 - creating, 7-4
 - deploying, 7-7
 - running, 7-14
- .NET stored procedure, 1-3
- .NET Stored Procedures, 2-1
- .NET stored procedures, 1-2, 2-2
 - deployment, 7-1
- .NET Types, 8-14
- New Package Window, 6-2, 6-9
- new project, 3-1
- New Project dialog, 3-1
- NLS error messages setting, 8-7
- NLS number format
 - settings, 8-5
- NLS sort order, 8-6
- NLS_LANG parameter, 8-8
- NLS_LANGUAGE parameter, 8-2, 8-7, 8-8
- NLS_SORT parameter, 8-6
- NLS_TERRITORY parameter, 8-2, 8-5, 8-8

O

- ODAC (Oracle Data Access Components), 2-2
- ODP.NET
 - definition, 1-2
 - globalization, 8-1
 - using, 5-1
- ODP.NET Types, 8-14
- Open() method, 3-11, 8-8
- opening connection, 3-11
- Oracle Data Access Components
 - ODAC, 2-1
- Oracle Data Access Components (ODAC)
 - downloading, 2-2
- Oracle Data Provider for .NET, 2-1
 - definition, 1-2
 - installation, 2-2
 - using, 4-1
- Oracle Database, 2-1
- Oracle Database Extensions for .NET
 - installing, 2-2
 - upgrades, 2-2
- Oracle Database installation, 2-1
- Oracle date format, 8-2
- Oracle Deployment Wizard for .NET, 1-3, 7-7
- Oracle Developer Tools
 - definition, 1-2
 - features
 - designer, 1-2
 - drag and drop, 1-2
 - dynamic help, 1-2

- Oracle Data Window, 1-2
- Oracle Query Window, 1-2
- PL/SQL editor, 1-2
 - wizard, 1-2
- installation, 2-2
- Oracle Developer Tools for Visual Studio
 - using, 5-1
- Oracle error message, 8-7
- Oracle linguistic sort, 8-6
- Oracle number format, 8-5
- Oracle Project
 - creating, 7-3
- Oracle Universal Installer (OUI), 2-2
- OracleClrAgent service, 7-1
- OracleCommand class, 4-1, 4-2, 4-4
 - using stored procedure, 6-10
- OracleConnection class, 3-11, 8-8
 - GetSessionInfo() method, 8-9
 - Open() method, 8-8
- OracleDataAccess.dll, 3-4
- OracleDataReader class, 4-2, 4-7, 4-8
- OracleDbType property, 4-4
- OracleError class, 3-16
- OracleErrorCollection class, 3-16
- OracleException class, 3-16, 3-18
- OracleGlobalization class, 8-9
 - GetThreadInfo() method, 8-14
 - SetThreadInfo() method, 8-14
- OracleGlobalization.GetClientInfo() method, 8-8
- OracleParameter class, 4-4, 6-10
- OracleParameterCollection class, 4-4
- OracleRefCursor class, 6-2
- OracleService, A-1
- OUI (Oracle Universal Installer), 2-2

P

- package
 - new, 6-2, 6-9
- package body, 6-1
- package interface, 6-1
- PACKAGE type, 6-1
- ParameterName, 4-4
- password
 - save, 5-1
- PL/SQL package
 - body, 6-1
 - definition, 6-1
 - interface, 6-1
- PL/SQL packages
 - introduction, 6-1
- PL/SQL stored procedure
 - definition, 6-1
 - in ODP.NET, 6-10
 - introduction, 6-1
 - ref cursor, 6-2, 6-9
- preview SQL, 5-5, 6-2, 6-9
- primary key
 - column, 5-10
- procedure

- run, 7-14
- project
 - add reference, 3-4
 - connect, 3-4
 - new, 3-1
 - solution, 3-1
 - type, 3-1
 - Visual Basic, 3-1
 - Visual C#, 3-1
- Properties window, 3-7
- property
 - Direction, 4-4
 - Error, 3-16
 - OracleDbType, 4-4
 - OracleDbType property, 4-4
 - ParameterName, 4-4
 - Size, 4-4
 - Value, 4-4

Q

- query performance, 4-4
- Query Window
 - running .NET procedures, 7-16
- query work area
 - definition, 6-2

R

- Rebuild Solution, 3-17
- record
 - add, 5-13
- records, 4-13
- REF CURSOR
 - accessibility, 6-2
 - assigning, 6-2, 6-9
 - definition, 6-2
 - introduction, 6-2
 - PL/SQL data type, 6-2
- ref cursor
 - PL/SQL stored procedure, 6-2, 6-9
- reference
 - adding, 3-4
- result set, 6-2
- retrieving data
 - accessor type, 4-2
 - bind variable, 4-4
 - looping, 4-7
 - multiple column, 4-7
 - multiple row, 4-7
 - multiple values, 4-7
 - simple query, 4-2
 - value methods, 4-2
- role
 - user, 5-1
 - user default, 5-1
- Run Function window, 7-14
- running .NET procedures in Query Window, 7-16
- running .NET procedures in SQL, 7-16

S

- Sample Data, 2-1
- Save command, 3-6
- schema
 - display, 5-1, 7-2
- schema object, 1-2, 7-2
- SELECT statement
 - bind variable, 4-4
 - simple, 4-4
- Server Explorer, 1-2, 5-13, 6-2, 6-9, 7-14
- service_name, A-1
- Services, A-1
- session globalization setting, 8-9
- SetThreadInfo() method, 8-14
- simple query, 4-2
- Size property, 4-4
- solution, 3-1
- Specify copy options window, 7-7
- Specify methods and security details window, 7-7
- Specify your deployment option window, 7-7
- SQL
 - preview, 5-5
- SQL query, 4-1
- SQL statement string, 4-1
- SQL*Plus, 8-2
- sqlnet.ora, 2-8
- start Oracle Database Instance, A-1
- statement
 - case, 3-18
 - Imports, 3-6
 - optimizing, 4-4
 - parsing, 4-4
 - reusing, 4-4
 - using, 3-6
- stop Oracle Database Instance, A-1
- stop Oracle Database instance, A-1
- stored procedure
 - definition, 6-1
- Summary window
 - window
 - Summary, 7-7
- SYSDBA
 - connection, 7-7
- System.Globalization, 8-1
- System.Resources, 8-1
- System.Threading.Thread.CurrentThread.CurrentCulture parameter, 8-14

T

- table
 - add data, 5-13
 - constraint, 5-10
 - add, 5-12
 - constraint name, 5-10
 - constraint properties, 5-10
 - creating, 5-5
 - data, 5-13
 - grid, 5-13
 - new, 5-5

- new relational, 5-5
- query, 5-15
- record, 5-13
- relational, 5-5
- retrieve data, 5-13
- simple query, 5-15
- table design view, 5-8
- table design window, 5-5
- Text property, 4-2
- Textbox control, 3-7
- thread-based globalization setting, 8-14
- tnsnames.ora, A-1
 - configuring, 2-8
- toolbox, 3-7
- try code block, 4-1, 4-8
- Try-Catch-Finally block, 3-17
- Try-Catch-Finally error handling, 3-17

U

- unlocking account, 5-1, 7-2
- unlocking user account
 - Oracle Database interface, 7-7
- updating data
 - bind variable, 4-4
- user
 - locale settings, 8-8
 - role, 5-1, 7-2
- user interface
 - designing, 3-7
- user_source view, 6-1
- using statement, 3-6

V

- Value property, 4-4
- variable declaration, 4-8
- VB statement
 - Imports, 3-6
- view
 - Design, 5-5
 - table design, 5-8
 - user_source, 6-1
- View menu, 3-7
- Visual Studio, 1-3
- Visual Studio .NET 2008, 2-2

W

- warning, 3-16
- window
 - Run Function, 7-14
 - Specify copy options, 7-7
 - Specify methods and security details, 7-7
 - Specify your deployment option, 7-7
 - table design, 5-5
- Windows Registry, 8-8

