Building .NET Applications with Oracle

SPONSORED BY Dr. Dobb's
Building .NET Applications with Oracle

Introduction

This hands on lab is designed to allow the student to pick and choose the content he or she is most interested in learning about. Lessons in this lab cover different levels of experience, including those who are new to .NET and Oracle, those who want to find out what's new, and those who are experienced users.

Please read through the summaries of the lessons below before choosing which one you would like to start with. An estimate of the time required to complete each lesson is provided.

<table>
<thead>
<tr>
<th>List of Lessons</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building ASP.NET Web Applications with Oracle Developer Tools for Visual Studio .NET</td>
<td>7</td>
</tr>
<tr>
<td>Getting Started with Oracle Data Provider for .NET (C# Version)</td>
<td>31</td>
</tr>
<tr>
<td>Getting Started with Oracle Data Provider for .NET (VB.NET Version)</td>
<td>69</td>
</tr>
<tr>
<td>Building .NET Applications Using Oracle Developer Tools for Visual Studio .NET</td>
<td>107</td>
</tr>
<tr>
<td>Debugging Oracle PL/SQL from Visual Studio</td>
<td>177</td>
</tr>
<tr>
<td>Optimizing Data Access Performance with ODP.NET</td>
<td>231</td>
</tr>
<tr>
<td>Using Database Change Notification With ODP.NET and Oracle Database</td>
<td>253</td>
</tr>
<tr>
<td>Developing and Deploying a .NET Stored Function</td>
<td>277</td>
</tr>
</tbody>
</table>

Lesson Summaries

Building ASP.NET Web Applications with Oracle Developer Tools for Visual Studio .NET

Purpose

This tutorial describes how to quickly create a web application using ASP.NET and Oracle without having to write much code.

Time to Complete

Approximately 20 minutes
Getting Started with Oracle Data Provider for .NET (C# Version)

Purpose

This tutorial describes how you can build a simple C# application that uses the Oracle Data Provider for .NET to access data in an Oracle Database.

Time to Complete

Approximately 30 minutes

Getting Started with Oracle Data Provider for .NET (VB.NET Version)

Purpose

This tutorial is identical to the one above, except VB.NET is used.

Time to Complete

Approximately 30 minutes

Building .NET Applications Using Oracle Developer Tools for Visual Studio .NET

Purpose

This tutorial describes how to create tables, indexes, and contraints, modify Oracle table data, automatically generate .NET code, and edit, compile and run PL/SQL using Oracle Developer Tools for Visual Studio .NET.

Time to Complete

Approximately 45 minutes

Debugging Oracle PL/SQL from Visual Studio

Purpose

This tutorial shows how you can debug Oracle PL/SQL from Visual Studio. A C# application that sets up and calls a PL/SQL stored procedure is used to illustrate the ability to seamlessly step from .NET code into PL/SQL and back out again.
Optimizing Data Access Performance with ODP.NET

Purpose

The Oracle database has numerous methods to optimize query execution and data retrieval from clients. ODP.NET enables these performance optimizations, allowing .NET developers to build more efficient database programs. This tutorial covers some of the most commonly used performance tuning methods for ODP.NET developers.

Time to Complete

Approximately 20 minutes

Using Database Change Notification With ODP.NET and Oracle Database

Purpose

The Oracle Data Provider for .NET provides a notification framework that supports Database Change Notification in the database server. Database Change Notification enables applications to receive notifications from a registered callback routine when there is a change in a query result set, schema objects, or the state of the database that could affect the results of a cached client result set.

Time to Complete

Approximately 20 minutes

Developing and Deploying a .NET Stored Function

Purpose

This tutorial shows how you can develop and deploy a .NET stored function into the Oracle database using Oracle Developer Tools for Visual Studio .NET. This gives you the ability to write stored procedures and functions for Oracle in the .NET language of your choice.

Time to Complete

Approximately 20 minutes
Building ASP.NET Web Applications with Oracle Developer Tools for Visual Studio .NET
Building ASP.NET Web Applications with Oracle Developer Tools for Visual Studio .NET

Purpose

This tutorial describes how you can build ASP.NET web applications that access Oracle Database using Oracle Developer Tools for Visual Studio .NET

Time to Complete

Approximately 20 minutes

Topics

This tutorial covers the following topics:

- Overview
- Prerequisites
- Creating a Web Site
- Creating the GridView
- Creating a Data Source
- Executing your Web Site
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

Oracle Developer Tools for Visual Studio .NET is a tightly integrated "add-in" for Visual Studio 2005. This tight integration makes it easy to create ASP.NET Web Applications that access Oracle database without requiring the developer write much code.

Prerequisites

Before starting this tutorial, you should first complete the following steps:
Creating a Web Site

To create an ASP.NET web site, perform the following steps.

1. From your Windows Start menu, start Visual Studio.

2. From File menu, select **New > Web Site**.
3. In the New Web Site window, select **ASP.NET Web Site** and click **OK**.

![New Web Site window](image)

Creating the GridView

To create a gridview, perform the following steps:

1. Click the **Design** tab.
2. From View menu, select Toolbox.
<table>
<thead>
<tr>
<th>View</th>
<th>Website</th>
<th>Build</th>
<th>Debug</th>
<th>Format</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Markup</td>
<td>Shift+F7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component Designer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open With...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visible Borders</td>
<td>Ctrl+Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Details</td>
<td>Ctrl+Shift+Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non Visual Controls</td>
<td>Ctrl+Alt+Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server Explorer</td>
<td>Ctrl+Alt+S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solution Explorer</td>
<td>Ctrl+Alt+L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bookmark Window</td>
<td>Ctrl+K, Ctrl+W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class View</td>
<td>Ctrl+Shift+C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Definition Window</td>
<td>Ctrl+, Ctrl+D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Browser</td>
<td>Ctrl+Alt+J</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error List</td>
<td>Ctrl+, Ctrl+E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>Ctrl+Alt+O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Properties Window</td>
<td>F4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task List</td>
<td>Ctrl+, Ctrl+T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toolbox</td>
<td>Ctrl+Alt+X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Find Results
- Other Windows
- Toolbars
- Full Screen Shift+Alt+Enter
3. In the toolbox, expand Data. Drag the **GridView** control to the Design area.

---

**Creating a Data Source**

To create a new data source, perform the following steps:
1. Select <New data source...> from the Choose Data Source drop down. A Data Source Configuration wizard starts.

2. Select Database. Enter HR as the ID for the data source. Click OK.
3. Click **New Connection**...
4. In Add Connection dialog box, first select the appropriate Data Source provider.

**Note:** If the Data source is already set to Oracle Database Server(Oracle ODP.NET), then move to next step.

Click **Change** to select the Data Source provider.
In the Change Data Source dialog box, select **Oracle Database Server** as the Data Source and **Oracle Data Provider for .NET** as the Data Provider. Click **OK**.
5. In the Data source name drop down, select the <SID> of your Oracle Database instance. Enter HR for the Username and Password. Click Test Connection.

6. Your test was successful. Click OK.
7. Click **OK**.
8. Click **Next**.
9. Click **Next** to save the connect string in the application configuration file.
10. Select the EMPLOYEES table from the Name drop down. Click the checkbox in front of *. Click **Next**.
11. Click Test Query.
12. The result of your query is shown. Click **Finish**.
13. Click the **Enable Paging** checkbox.
Executing your Web Site

To run the ASP.NET web application, perform the following steps:

1. Select **Build > Build Solution**.
2. Select **Debug > Start Debugging**.

3. Click **OK** to modify the web.config file to enable debugging.
4. A web browser will open showing the web application.

```
[Table showing employee data]
```

### Summary

In this tutorial, you learned how to:

- Create a ASP.NET web site
- Create a GridView
- Create a new Data Source
- Execute your web site
Getting Started with Oracle Data Provider for .NET (C# Version)

Purpose

This tutorial describes how you can build a .NET application that uses the Oracle Data Provider for .NET to access data in an Oracle Database.

Time to Complete

Approximately 30 minutes

Topics

This tutorial covers the following topics:

- Overview
- Prerequisites
- Creating a Project in Visual Studio .NET
- Adding a Reference
- Retrieving Data from the Database
- Incorporating Interactive Controls
- Adding Error Handling
- Retrieving Multiple Columns and Rows
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

In addition to basic Oracle client connectivity software, .NET applications require the use of what is known as a managed data provider (where "managed" refers to code managed by the .NET framework). The data provider is the layer between the .NET application code and the Oracle client connectivity software.

The Oracle Data Provider for .NET (ODP.NET) is Oracle's high performance ADO.NET 2.0 compliant data provider that exposes a complete set of Oracle specific features and tuning options including support for Real Application Clusters, XML DB, and advanced security. It is available for free download from the Oracle Technology Network website.

When ODP.NET and any required Oracle client connectivity software is installed, application development using Visual Studio can begin. It is a good idea to confirm client connectivity before starting development. If you can connect to Oracle
using SQL*Plus on the same machine as VS.NET, then you know that your Oracle client-side software is properly installed and configured.

If you are new to Oracle, see the section "Connecting to the Oracle Database" in the Oracle Data Provider for .NET Developer's Guide 10g for background information regarding ODP.NET specifically, or to the Oracle Database Administrator's Guide 10g for information about managing the Oracle Database generally. You can also consult the "Connect to an Oracle Database Using ODP.NET" sample code "how-to" document.

Prerequisites

Before you perform this tutorial, you should:

- Install and create an Oracle Database 11g or Oracle Database 10g database server
- Install Visual Studio .NET 2003 or above
- Install ODP.NET 11g from the Oracle Database Client or OTN

Creating a Project in Visual Studio .NET

To create a new project in Visual Studio, perform the following steps:

A New Project window appears.

2. From the Project types list, select Visual C#: Windows. Select the Template Windows Application. In the Name field, enter OraWinApp. In the Location field, enter the directory where you want to save the files. (Note: if the directory does not exist, it is created). Click OK.
3. The project has been created. From View menu, select Solution Explorer to open the Solution Explorer if not already open.
**Adding a Reference**

Because your project needs access to an Oracle database, it is necessary to add a reference to the dll containing the data provider. Perform the following steps:

1. From Project menu, select **Add Reference**...
An Add Reference dialog box appears.

2. Scroll down the list of Component Names and select **Oracle.DataAccess**. If there are multiple versions, choose the version "2.111.5.10". Click **OK**.
Retrieving Data From the Database

To retrieve data from the database, you can add some controls to your form. Perform the following steps:

1. From View menu, select Toolbox. The toolbox appears. Expand Common Controls.
2. Drag the **Button** control to the Form1 canvas.
The control is shown on the canvas. You also need to create a Label.
3. From the Toolbox, drag the **Label** control to the Form1 canvas.
4. Double-click **button1** to open the code window. Note, Form1.cs code window opens.
5. Add the following C# using statements before the Public Class declaration.

```csharp
using Oracle.DataAccess.Client; // ODP.NET Oracle managed provider
using Oracle.DataAccess.Types;
```
Add the following C# code in between the private void button1_Click(object sender, EventArgs e) { and } statements.

```csharp
string oradb = "Data Source=ORCL;User Id=hr;Password=hr;";
OracleConnection conn = new OracleConnection(oradb); // C#
conn.Open();
OracleCommand cmd = new OracleCommand();
cmd.Connection = conn;
cmd.CommandText = "select department_name from departments where department_id = 10";
cmd.CommandType = CommandType.Text;
OracleDataReader dr = cmd.ExecuteReader();
dr.Read();
label1.Text = dr.GetString(0);
conn.Dispose();
```
7. Select **Build > Build Solution**.

In the output window, ensure you did not get any errors. Now you can run the form.
8. Select **Debug > Start Without Debugging**.
9. The form appears. Click **button1**.

The result is shown in the label. Since the query in the code was to show the department name for department id of 10, the result is the Administration department. Close Form1.
Incorporating Interactive Controls

Instead of running the hard coded query, you can add a textbox control to accept user input for the department id. Perform the following steps:

1. Select the **Form1.cs [Design]** tab. From the Toolbox, drag the **TextBox** control to Form1 canvas.

   ![Toolbox](image)

   The textbox is shown. You can drag it to the desired location.
2. Drag a **Label** control on Form1 canvas.
3. The Label is shown. Right-click **Label2** and select **Properties**.
4. In the Properties sheet, change the Text property to **Enter Department ID:**
5. Select the **Form1.cs** tab.

6. You use a bind variable to include the value of the text box as part of the SELECT statement. Bind variables improve performance by preventing the database from having to reparse the SQL statement every time the text box value changes.

Add the following code right after the `conn.Open()` statement:

```csharp
OracleParameter parm = new OracleParameter();
parm.OracleDbType = OracleDbType.Decimal;
parm.Value = textBox1.Text;
```
7. **Add the cmd.Parameters statement right after the cmd.Connection statement:**

```csharp
    cmd.Parameters.Add(parm);
```
8. Change the code for the cmd.CommandText statement to the following:

```csharp
    cmd.CommandText = "select department_name from departments where department_id = :1";
```

```csharp
private void button1_Click(object sender, EventArgs e)
{
    string oradb = "Data Source=ORCL;User Id=hr;Password=hr;";

    OracleConnection conn = new OracleConnection(oradb); // C#
    conn.Open();

    OracleParameter parm = new OracleParameter();
    parm.OracleDbType = OracleDbType.Decimal;
    parm.Value = textBox1.Text;

    OracleCommand cmd = new OracleCommand();
    cmd.Connection = conn;
    cmd.Parameters.Add(parm);
    cmd.CommandText = "select department_name from departments where department_id = :1";
    cmd.CommandType = CommandType.Text;

    OracleDataReader dr = cmd.ExecuteReader();
    dr.Read();
    label1.Text = dr.GetString(0);

    conn.Dispose();
}
```

9. Select **Build > Rebuild Solution**.

Check in the output window for any errors reported. If no errors or warnings, proceed with the next step.
10. Select **Debug > Start Without Debugging**.

![Debug Menu](image)

11. Enter **20** for the Department ID and click **button1**.

![Form1](image)
12. Notice that the Label1 changes to the name of the department. This value depends on what is entered in the Department ID field.

13. Enter 300 for the Department ID and click button1.
14. This time you receive an error because there is no department with the id of 300. Click Quit. In the next topic you add some error handling so you know exactly why you received the error.

Adding Error Handling

Error handling is needed to instruct the user why something doesn't work properly. In this topic, you modify your code to prevent an error when an invalid Department ID is provided. In this tutorial, you add some very simple error handling code. For more robust error handling, Structured Exception Handling should be used. For more information on Structured Exception Handling with ODP.NET please refer to the ODP.NET online documentation. Perform the following steps:

1. In Form1.cs window, change the following code from:

```csharp
dr.Read();
label1.Text = dr.GetString(0);
```

   to

```csharp
if (dr.Read())
{
    label1.Text = dr.GetString(0);
}
else
{
    label1.Text = "Department ID not found";
}
```

Back to Topic List
2. Select **Build > Rebuild Solution**. Ensure that there are no errors reported in the output window.
3. Select **Debug > Start Without Debugging**.

4. Enter **300** again for the Department ID and click **button1**.
5. This time, the message **Department ID not found** is displayed. Close Form1.

Retrieving Multiple Columns and Rows

In this topic, you want to retrieve more than one set of information using a listbox. Perform the following steps:

1. Click the **Form1.cs [Design]** tab. From the Toolbox, drag the **Listbox** control to the Form 1 canvas.
2. The new listbox appears on the form. You can position and re-size the listbox appropriately. Click **Form1.cs** tab.
3. Delete the `cmd.Parameters.Add(parm);` statement.

```
OracleConnection conn = new OracleConnection(oradb); // C#
conn.Open();

OracleParameter parm = new OracleParameter();
parm.OracleDbType = OracleDbType.Decimal;
parm.Value = textBox1.Text;

OracleCommand cmd = new OracleCommand();
cmd.Connection = conn;
cmd.CommandText = "select department_name from departments where department_id = :1";
cmd.CommandType = CommandType.Text;

OracleDataReader dr = cmd.ExecuteReader();
if (dr.Read())
{
    label1.Text = dr.GetString(0);
}
else
{
    label1.Text = "Department ID not found";
}
```
4. Change the cmd.CommandText code from:

```csharp
cmd.CommandText = "select department_name from departments where department_id = :1";
```

to

```csharp
cmd.CommandText = "select department_id, department_name, city"
   + " from departments d, locations l"
   + " where d.location_id = l.location_id";
```

```csharp
OracleConnection conn = new OracleConnection(oradb); // C#
conn.Open();

OracleParameter parm = new OracleParameter();
parm.OracleDbType = OracleDbType.Decimal;
parm.Value = textBox1.Text;

OracleCommand cmd = new OracleCommand();
cmd.Connection = conn;
cmd.CommandText = "select department_id, department_name, city"
   + " from departments d, locations l"
   + " where d.location_id = l.location_id";
cmd.CommandType = CommandType.Text;

OracleDataReader dr = cmd.ExecuteReader();
if (dr.Read())
{
   label1.Text = dr.GetString(0);
}
else
{
   label1.Text = "Department ID not found";
}
```

5. Also change the following lines of code from:

```csharp
if (dr.Read())
{
   label1.Text = dr.GetString(0);
}
else
{
   label1.Text = "Department ID not found";
}
```

to
while (dr.Read())
{
    listBox1.Items.Add("The " + dr.GetString(1) + " department is in " + dr.GetString(2));
}

OracleConnection conn = new OracleConnection(oradb); // C#
conn.Open();

OracleParameter parm = new OracleParameter();
parm.OracleDbType = OracleDbType.Decimal;
parm.Value = textBox1.Text;

OracleCommand cmd = new OracleCommand();
cmd.Connection = conn;
cmd.CommandText = "select department_id, department_name, city" + " from departments d, locations l" + " where d.location_id = l.location_id";
cmd.CommandType = CommandType.Text;

OracleDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
    listBox1.Items.Add("The " + dr.GetString(1) + " department is in " + dr.GetString(2));
}

conn.Dispose();

6. Select Build > Rebuild Solution.
7. Select **Debug > Start Without Debugging**.

8. Click **button1**.
9. The list of all the departments and their location is shown. Close Form1.

Summary

In this tutorial, you learned how to:

- Create a project and add a reference
- Retrieve data from the database
- Incorporate interactive controls and error handling
- Retrieve Multiple Columns and Rows
Getting Started with Oracle Data Provider for .NET (VB.NET Version)
Getting Started with Oracle Data Provider for .NET (VB Version)

Purpose

This tutorial describes how you can build a .NET application that uses the Oracle Data Provider for .NET to access data in an Oracle Database.

Time to Complete

Approximately 30 minutes

Topics

This tutorial covers the following topics:

- Overview
- Prerequisites
- Creating a Project in Visual Studio .NET
- Adding a Reference
- Retrieving Data from the Database
- Incorporating Interactive Controls
- Adding Error Handling
- Retrieving Multiple Columns and Rows
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

In addition to basic Oracle client connectivity software, .NET applications require the use of what is known as a managed data provider (where "managed" refers to code managed by the .NET framework). The data provider is the layer between the .NET application code and the Oracle client connectivity software.

The Oracle Data Provider for .NET (ODP.NET) is Oracle's high performance ADO.NET 2.0 compliant data provider that exposes a complete set of Oracle specific features and tuning options including support for Real Application Clusters, XML DB, and advanced security. It is available for free download from the Oracle Technology Network website.

When ODP.NET and any required Oracle client connectivity software is installed, application development using Visual Studio can begin. It's a good idea to confirm client connectivity before starting development. If you can connect to Oracle
If you are new to Oracle, see the section "Connecting to the Oracle Database" in the Oracle Data Provider for .NET Developer's Guide 10g for background information regarding ODP.NET specifically, or to the Oracle Database Administrator's Guide 10g for information about managing the Oracle Database generally. You can also consult the "Connect to an Oracle Database Using ODP.NET" sample code "how-to" document.

Prerequisites

Before you perform this tutorial, you should:

- Install and create an Oracle Database 11g or Oracle Database 10g database server
- Install Visual Studio .NET 2003 or above
- Install ODP.NET 11g from the Oracle Database Client or OTN

Creating a Project in Visual Studio .NET

To create a new project, perform the following steps:

2. From the Project types list, select **Visual Basic: Windows**. Select the Template **Windows Application**. In the Name field, enter **OraWinApp**. In the Location field, enter the directory where you want to save the files. (Note: if the directory does not exist, it is created). Click **OK**.
3. The project has been created. From View menu, select Solution Explorer to open the Solution Explorer if not already open.
Adding a Reference

Because your project needs access to an Oracle database, it is necessary to add a reference to the dll containing the data provider. Perform the following steps:
1. From Project menu, select **Add Reference**... 

An Add Reference dialog box appears.
2. Scroll down the list of Component Names and select **Oracle.DataAccess**. If there are multiple versions, choose the version "2.111.5.10". Click **OK**.

![Add Reference dialog](image)

---

**Retrieving Data From the Database**

To retrieve data from the database, you can add some controls to your form. Perform the following steps:

1. From **View** menu, select **Toolbox**. The toolbox appears. Expand **Common Controls**.
2. Drag the **Button** control to the Form1 canvas.
The control is shown on the canvas. You also need to create a Label.
3. From the Toolbox, drag the **Label** control to the Form1 canvas.
4. Double-click **Button1** to open the code window. Note, Form1.cs code window opens.
5. Add the following VB.NET `Imports` statements before the Public Class declaration.

```vbnet
Imports System.Data
Imports Oracle.DataAccess.Client ' ODP.NET Oracle managed provider
Imports Oracle.DataAccess.Types
```
6. Add the following VB.NET code between the Private Sub and End Sub statements.

```vbnet
Dim oradb As String = "Data Source=orcl;User Id=hr;Password=hr;"
Dim conn As New OracleConnection(oradb) conn.Open()
Dim cmd As New OracleCommand
    cmd.Connection = conn
    cmd.CommandText = "select department_name from departments where department_id = 10"
    cmd.CommandType = CommandType.Text
Dim dr As OracleDataReader = cmd.ExecuteReader()
    dr.Read()
    Label1.Text = dr.Item("department_name")
conn.Dispose()
```
7. Select **Build > Build Solution**.

In the output window, ensure you did not get any errors. Now you can run the form.
8. Select **Debug > Start Without Debugging.**

![Debug menu](image)

9. The form appears. Click **Button1.**

![Form1](image)

The result is shown in the label. Since the query in the code was to show the department name for department id of 10, the result is the Administration department. Close Form1.
Incorporating Interactive Controls

Instead of running the hard coded query, you can add a textbox control to accept user input for the department id. Perform the following steps:

1. Select the **Form1.vb [Design]** tab. From the Toolbox, drag the **TextBox** control to Form1 canvas.
The textbox is shown. You can drag it to the desired location.
2. Drag a **Label** control on Form1 canvas.
3. The Label is shown. Right-click **Label2** and select **Properties**.
4. In the Properties sheet, change the Text property to **Enter Department ID:**
5. Select the **Form1.vb** tab.

6. You use a bind variable to include the value of the text box as part of the SELECT statement. Bind variables improve performance by preventing the database from having to reparse the SQL statement every time the text box value changes.

Add the following code right after the `conn.Open()` statement:

```vbcn
Dim parm As New OracleParameter
parm.OracleDbType = OracleDbType.Decimal
parm.Value = TextBox1.Text
```
7. Add the `cmd.Parameters` statement right after the `cmd.Connection` statement:

```csharp
    cmd.Parameters.Add(parm)
```

```csharp
Dim oradb As String = "Data Source=ORCL;User Id=hr;Password=hr;"

Dim conn As New OracleConnection(oradb)
conn.Open()

Dim parm As New OracleParameter
parm.OracleDbType = DbType.Decimal
parm.Value = TextBox1.Text

Dim cmd As New OracleCommand
cmd.Connection = conn
cmd.CommandText = "select department_name from departments where department_id = 10"

Dim dr As OracleDataReader = cmd.ExecuteReader()
    dr.Read()
Labell1.Text = dr.Item("department_name")
```
8. Change the code for the cmd.CommandText statement to the following:

```vbnet
    cmd.CommandText = "select department_name from departments where department_id = :1"
```

9. Select **Build > Rebuild Solution**.

Check in the output window for any errors reported. If no errors or warnings, proceed with the next step.
10. Select **Debug > Start Without Debugging**.

11. Enter **20** for the Department ID and click **Button1**.
12. Notice that the Label1 changes to the name of the department. This value depends on what is entered in the Department ID field.

![Form1](image1)

13. Enter 300 for the Department ID and click **Button1**.

![Form1](image2)
14. This time you receive an error because there is no department with the id of 300. Click Quit. In the next topic you add some error handling so you know exactly why you received the error.

Adding Error Handling

Error handling is needed to instruct the user why something doesn't work properly. In this topic, you modify your code to prevent an error when an invalid Department ID is provided. In this tutorial, you add some very simple error handling code. For more robust error handling, Structured Exception Handling should be used. For more information on Structured Exception Handling with ODP.NET please refer to the ODP.NET online documentation. Perform the following steps:

1. In the Form1.vb code, change the following code from:

   ```vbnet
   dr.Read()
   label1.Text = dr.Item("department_name");
   ```

   to

   ```vbnet
   If dr.Read() Then
       Label1.Text = dr.Item("department_name")
   Else
       Label1.Text = "Department ID not found"
   End If
   ```
2. Select **Build > Rebuild Solution**.
3. Select **Debug > Start Without Debugging**.

4. Enter **300** again for the Department ID and click **Button1**.
5. This time, the message **Department ID not found** is displayed. Close Form1.

![Form1](image)

**Retrieving Multiple Columns and Rows**

In this topic, you want to retrieve more than one set of information using a listbox. Perform the following steps:

1. Click the **Form1.vb [Design]** tab. From the Toolbox, drag the **Listbox** control to the Form 1 canvas.
2. The new listbox appears on the form. You can position and re-size the listbox appropriately. Click Form1.vb tab.
3. Delete the `cmd.Parameters.Add(parm)` statement.

```vbnet
Dim conn As New OracleConnection(oradb)
conn.Open()

Dim parm As New OracleParameter
parm.OracleDbType = OracleDbType.Decimal
parm.Value = TextBox1.Text

Dim cmd As New OracleCommand
cmd.Connection = conn
cmd.CommandText = "select department_name from departments where department_id = :1"
cmd.CommandType = CommandType.Text

Dim dr As OracleDataReader = cmd.ExecuteReader()
If dr.Read() Then
    Label1.Text = dr.Item("department_name")
Else
    Label1.Text = "Department ID not found"
End If
```
4. Change the `cmd.CommandText` code from:

```vbnet
cmd.CommandText = "select department_name from departments where department_id = :1"
```

to

```vbnet
cmd.CommandText = "select department_id, department_name, city" _
+ " from departments d, locations l" _
+ " where d.location_id = l.location_id"
```

```vbnet
Dim conn As New OracleConnection(oradb)
conn.Open()

Dim parm As New OracleParameter
parm.OracleDbType = OracleDbType.Decimal
parm.Value = TextBox1.Text

Dim cmd As New OracleCommand
cmd.Connection = conn
cmd.CommandText = "select department_id, department_name, city" _
+ " from departments d, locations l" _
+ " where d.location_id = l.location_id"

Dim CommandType = CommandType.Text

Dim dr As OracleDataReader = cmd.ExecuteReader()
If dr.Read() Then
    Label1.Text = dr.Item("department_name")
Else
    Label1.Text = "Department ID not found"
End If
```

5. Also change the following lines of code from:

```vbnet
If dr.Read() Then
    Label1.Text = dr.Item("department_name")
Else
    Label1.Text = "Department ID not found"
End If
```

to

```vbnet
While dr.Read()
    ListBox1.Items.Add("The " + dr.Item(1) _
                        + " department is in " + dr.Item("city"))
End While
```
6. Select **Build > Rebuild Solution**.

```vbnet
Dim conn As New OracleConnection(oracleb)
conn.Open()

Dim parm As New OracleParameter
parm.OracleDbType = OracleDbType.Decimal
parm.Value = TextBox1.Text

Dim cmd As New OracleCommand
cmd.Connection = conn
cmd.CommandText = "select department_id, department_name, city" -
    + " from departments d, locations l" -
    + " where d.location_id = l.location_id"
cmd.CommandType = CommandType.Text

Dim dr As OracleDataReader = cmd.ExecuteReader()
While dr.Read()
    ListBox1.Items.Add("The " + dr.Item(1) -
    + " department is in " + dr.Item("city"))
End While
```
7. Select **Debug > Start Without Debugging**.

8. Click **Button1**.
9. The list of all the departments with their locations is shown. Close Form1.

![Form1]

**Summary**

In this tutorial, you learned how to:

- Create a project and add a reference
- Retrieve data from the database
- Incorporate interactive controls and error handling
- Retrieve Multiple Columns and Rows

Move your mouse over this icon to hide all screenshots.
Building .NET Applications Using Oracle Developer Tools for Visual Studio .NET
Building .NET Applications Using Oracle Developer Tools for Visual Studio .NET

Purpose

This tutorial describes how to build .NET applications using Oracle Developer Tools for Visual Studio .NET.

Time to Complete

Approximately 30 minutes

Topics

This tutorial covers the following topics:

- Overview
- Prerequisites
- Creating an Oracle Connection
- Creating a Table
- Viewing Data in a Table
- Generating Code Automatically
- Using Stored Procedures
- Using the Query Window
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

The new Oracle Developer Tools for Visual Studio .NET is a tightly integrated "Add-in" for Visual Studio .NET that brings the power of the Oracle Database to .NET developers! Designed to be completely intuitive for database developers, your productivity will increase starting on day one. And if you are new to Oracle development, this is an easy way to learn!

Use the Server Explorer to browse your Oracle schema, launch one of the many powerful designers and wizards to create and alter schema objects, and drag and drop schema objects onto your form to automatically generate code.

There's also a PL/SQL editor, Adhoc SQL Query Window, and integrated context sensitive online help, including the Oracle SQL and PL/SQL Users Guides. And with the Oracle Data Window you won't have to leave the Visual Studio
environment for routine database tasks like inserting and updating Oracle data or testing stored procedures!
This and many other features greatly improve developer productivity and make developing for Oracle on Windows easy to learn!

**Prerequisites**

Before you perform this tutorial, you should:

1. Install Oracle Database 10g or above.
2. Install Oracle Developer Tools for Visual Studio .NET
3. Install Visual Studio 2005

**Creating an Oracle Connection**

Before you begin creating your application, you open Visual Studio and examine the interface. Perform the following steps:

1. Select **Start > Programs > Microsoft Visual Studio 2005> Microsoft Visual Studio 2005**.

2. Select **View > Server Explorer**.

![Server Explorer](image)
3. The Server Explorer panel opens. Right-click **Data Connections** and select **Add Connection...**

![Server Explorer Panel]

4. The Add Connection dialog box opens. First, you need to select the appropriate data source provider. If Data Source is already set to **Oracle Database Server (Oracle ODP.NET)** then move to the next step, otherwise click **Change**.

![Add Connection Dialog Box]

In the Change Data Source dialog box, select **Oracle Database Server** as the Data Source and **Oracle Data Provider for .NET** as the Data Provider. Click **OK**.
5. Enter your Data source name. This is the SID of the Oracle instance you want to use.

Enter hr for the User name and Password, click **Save password**, select Default as Role, and click **Test connection**.
Test connection was successful. Click OK.
6. Click OK.

7. The HR.ORCL connection has been created and added under Data Connections in the Server Explorer panel.
Creating a Table

Now you create a new table called DEPENDENTS which has a foreign key with the EMPLOYEES table.

- Create the Columns
- Create the Index
- Create the Foreign Key

Create the Columns

To create the columns in the table, perform the following:

1. Expand HR.ORCL. Right-click Tables and select New Relational Table...
2. Enter **DEPENDENTS** for Table name and click **Add** in the Columns section.

3. Enter **FIRSTNAME** for the Name, select **VARCHAR2** for the Data type and enter **30** for the Size. Then click **Add**.
4. Enter **LASTNAME** for the Name, select **VARCHAR2** for the Data type and enter **30** for the Size. Then click **Add**.
5. Enter **BIRTHDATE** for the Name, select **DATE** for the Data type. Then click **Add**.
6. Enter **relationship** for the Name, select **VARCHAR2** for the Data type and enter **20** for the Size. Then click **Add**.
7. Enter **EMPLOYEEID** for the Name, select **NUMBER** for the Data type, **deselect Allow null** and enter 6 for the Precision and 0 for the Scale. Then click **Add**.
8. Enter **DEPENDENTID** for the Name, select **NUMBER** for the Data type, deselect **Allow null** and enter 6 for the Precision and 0 for the Scale. Then click **Preview SQL**.
9. Here is the SQL that will execute when the table is created. Click OK.
10. Click **Save**.
11. The Output window shows execution messages. The DEPENDENTS table has been created. Close the Output window.
12. Click the + in front of DEPENDENTS table in the Server Explorer panel. You can view the columns that have been created in this topic. Now you are ready to create the index.

Create the Index

To create the index for the table, perform the following:

1. Click the Indexes tab.
2. Click **Add**.
3. Enter `DEPENDENTS_INDEX` for the Name and click **Add** in the Index Keys area.
4. Select **DEPENDENTID** for the Key and then click **Save**.
5. The Output window shows execution messages. The **DEPENDENTS_INDEX** index has been created. **Close** the Output window.

```
------------- Done -------------

------ Saving changes from DEPENDENTS to database ------

CREATE INDEX "HR"."DEPENDENTS_INDEX" ON "HR"."DEPENDENTS" ("DEPENDENTID"
Success: 0 errors 0 warnings

------------- Done -------------
```
Create the Foreign Key

To create the foreign key for the table, perform the following:

1. Click **Constraints** tab.

2. Click **Add**.
3. Enter `EMPLOYEES_FK` for the Name, select `Foreign Key` for the type, select `EMPLOYEES` for the table, and select the constraint `EMP_EMP_ID_PK`. Select `EMPLOYEE_ID` as the Referenced Column and `EMPLOREEID` as the local column. Then click **Add**.
4. Enter **DEPENDENTS_PK** for the Name, select **Primary Key** for the type, and click **Add** in the Primary key columns area.
5. Select **DEPENDENTID** for the primary key column. Then click **Save**.
The output window shows the execution messages. Close the Output window.
6. Click the + in front of **Indexes** and **Constraints** in the Server Explorer panel. You can see the table, its indexes and constraints.

**Viewing Data in a Table**

You can access the data in a table by performing the following
1. Right-click DEPENDENTS table and select Retrieve Data...

![Server Explorer]

2. Enter at least 4 lines of data as shown in the table below. Make sure you enter a valid Employee ID. Then click Save.

<table>
<thead>
<tr>
<th>FIRSTNAME</th>
<th>LASTNAME</th>
<th>BIRTHDATE</th>
<th>RELATIONSHIP</th>
<th>EMPLOYEEID</th>
<th>DEPENDENTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Martin</td>
<td>06-MAY-80</td>
<td>daughter</td>
<td>104</td>
<td>9999</td>
</tr>
<tr>
<td>Sue</td>
<td>Littlefield</td>
<td>12-JUL-68</td>
<td>daughter</td>
<td>130</td>
<td>8888</td>
</tr>
<tr>
<td>David</td>
<td>Griffiths</td>
<td>02-APR-97</td>
<td>son</td>
<td>104</td>
<td>7777</td>
</tr>
<tr>
<td>Aaron</td>
<td>Young</td>
<td>31-AUG-99</td>
<td>son</td>
<td>111</td>
<td>6666</td>
</tr>
</tbody>
</table>
Generating Code Automatically

In this topic, you perform the following:

- Create a Visual C# Windows Application
- Create a new Data Source
- Designing the Form
- Run the Application

Create a Visual C# Windows Application

To create a Visual C# Windows application, perform the following:
1. Assuming you did not exit Visual Studio, from File menu, select **New > Project**.

![Start Page - Microsoft Visual Studio](image)

2. From the Project types list, select **Visual C#: Windows**. Select the Template **Windows Application**.

You can enter an appropriate name for the application in the Name field. Optionally, in the Location field, enter the directory where you want to save the files. (Note: if the directory does not exist, it is created).

For now, accept the default name and click **OK**.
3. A Form1.cs [Design] tab is added.
Create a new Data Source

You use the DEPENDENTS table you created in the HR schema, for this application. To create a new data source, perform the following steps:
1. From Data menu, select **Show Data Sources**.

A Data Sources panel is added.
2. In the Data Sources panel, click **Add New Data Source...**

Data Source Configuration wizard opens.

3. Select **Database** as the Data Source type. Click **Next**.

142 of 306
4. In the previous topic, you already created a data connection, HR.ORCL.Select HR.ORCL.Select Yes, include sensitive data in the Connection string option. Click Next.
5. Accept the default selection for saving the connection string to the application configuration file. Click **Next**.
8. Now, you select the database object you want in your data set. Expand **Tables**. Expand **DEPENDENTS**. Click the checkbox beside **DEPENDENTS**. A check mark appears beside the table name as well as its attributes. Also note the name of the data set.
Click **Finish**. The Dependents table gets added in the Data Sources panel.
To design a form, perform the following steps:

1. Now, you work in the Form1.cs[Design] tab. Resize the form as required.

2. In the Data Sources panel, expand DEPENDENTS. Drag DEPENDENTS to the Form1 canvas.
A grid of column names of the DEPENDENTS table gets added. You can resize the grid as required.
3. Double click the Save icon ( ). Note: If the icon is not enabled, right-click the icon and select Enabled.

Form1.cs code window opens.

In the Form1.cs code window, examine the `private void dEPENDENTSBindingNavigatorSaveItem_Click (object sender, EventArgs e)` procedure.
4. Add a try {} and a catch {} block for the three lines of code. Also, add a MessageBox statement as shown below: (the statements in bold are the ones you need to add)

```csharp
try
{
    this.Validate();
    this.dEPENDENTSBindingSource.EndEdit();
    this.dEPENDENTSTableAdapter.Update(this.dataSet1.DEPENDENTS);
}
```

```csharp
MessageBox.Show("Update successful");
```

```csharp
}
```

```csharp
catch (System.Exception ex)
{
    MessageBox.Show("Update failed");
}
```
Running the Application

To run the application, perform the following steps:
1. From Build menu, select **Build Solution**.

Ensure there are no errors reported in the output window.

2. From Debug menu, select **Start Debugging**.

Form1 shows the Dependents tables data.
3. Modify the DependentID of one of the records to a unique value. Click the **Save** icon.
A Message box displays with an "Update successful" message. Click **OK**.

4. Delete an employee record by clicking the **Delete icon**.
Click the **Save icon**. A Message box displays with an "Update successful" message. Click **OK**.

Close Form1.

**Using Stored Procedures**

Now you create a stored procedure and run it.

- [Create a Stored Procedure](#)
- [Edit the Package Body](#)
- [Run the Stored Procedure](#)
Create a Stored Procedure

1. From the Server Explorer panel, right-click **Packages** and select **New Package...**
2. Accept the default Package name and click **Add** under Methods.

3. Enter **GETCURSORS** for the method name, select **Procedure** for the method type and click **Add** under Parameters.
4. Enter **MAXROWS** for the Name and click **Add**.
5. Enter **EMPLOYEESCUR** for the Name, select **OUT** for Direction, and select **SYS_REFCURSOR** for Data type. Then click **Add**.

6. Enter **DEPENDENTSCUR** for the Name, select **OUT** for the Direction and select **SYS_REFCURSOR** for the Data type. Click **OK**.
7. Click **Preview SQL >>** to see the SQL that will be executed.
8. Review the SQL and click OK.
9. Click **OK** to create the Package.
The output window shows the execution messages. Close the **Output** window.
Edit the Package Body

1. The Package is created and the Package Body is opened. If you do not see the code that is generated for the Package Body, right-click PACKAGE1 Package in the Server Explorer and select Edit Package Body. Scroll down to find the line **NULL**;

```
BEGIN -- executable part starts here

-- Write PL/SQL and SQL statements to implement the processing logic
-- of subprogram. Example:
-- SELECT ENAME,
--   COMM
-- INTO varEname,
-- varComm
-- FROM EMP
-- WHERE EMPNO = 7369;

-- IF varComm IS NULL THEN
-- RAISE comm_missing;
-- END IF;

NULL;

-- EXCEPTION -- exception-handling part starts here
-- WHEN comm_missing THEN
-- dbms_output.put_line('Commision is NULL');

END "GETCURSORS";
END "PACKAGE1";
```

2. Replace the line **NULL**; with the following:

```
OPEN EMPLOYEESCUR FOR SELECT * FROM EMPLOYEES;
OPEN DEPENDENTSCUR FOR SELECT * FROM DEPENDENTS;
```
3. Right-click the window tab and select **Save**.
BEGIN -- executable logic
-- Write PL/SQL statements
-- of subprogram.
-- SELECT ENA
-- INTO var
-- FROM EMP
-- WHERE EMPNO = 7369;
--
-- IF varComn IS NULL THEN
-- RAISE comm_missing;
-- END IF;

OPEN EMPLOYEESCUR FOR SELECT * FROM EMPLOYEES;
OPEN DEPENDENTSCUR FOR SELECT * FROM DEPENDENTS;

-- EXCEPTION -- exception-handling part starts here
-- WHEN comm_missing THEN
-- dbms_output.put_line('Commision is NULL');

END "GETCURORS";
END "PACKAGE1";

The output window shows execution messages. **Close** the Output window.
Run the Stored Procedure

1. Expand the Package PACKAGE1 in the Server Explorer panel. Right-click GETCURSORS and select Run.

2. Enter 999 for the MAXROWS parameter value and click OK.
3. Select the value for EMPLOYEESCUR.

4. The details for the EMPLOYEESCUR are shown. Select the value for the DEPENDENTSCUR.
5. The DEPENDENTSCUR details are shown. Close the Output window.
Using the Query Window

The query window allows you to run one or more queries against data in your database. Perform the following steps:

1. Right-click your database connection **HR.ORCL** and select **Query Window**.
2. Enter the SQL command `SELECT * FROM DEPENDENTS;` and click **Execute**.

3. Your results are shown. To format the results, right-click in the query area, select **Set Query Output > Text** and click **Execute** again.
4. The results are shown in text format.
5. You can execute more than one SQL statement at a time. Add the SQL command `SELECT * FROM EMPLOYEES;` In order for both statements to be executed at the same time, you need to select both statements, then click **Execute**. Note: if both statements are not selected, only the last SQL statement in the list will run.
6. Scroll up to the top of your output. You see the data from both queries.

![SQL queries and table data]

**Summary**

In this lesson, you learned how to:

- Create a table, an index and constraints.
- View a tables data
- Automatically generate code
- Create and run a stored procedure
- Use a Query Window
Debugging Oracle PL/SQL from Visual Studio
Debugging Oracle PL/SQL from Visual Studio

Purpose

This tutorial describes how you can debug Oracle PL/SQL from Visual Studio.

Time to Complete

Approximately 30 minutes

Topics

This tutorial covers the following topics:

- Overview
- Adding a Connection
- Granting Privileges for Debugging
- Creating the PL/SQL Package and Package Body
- Creating a Project in Visual Studio .NET
- Configuring the Debugging Environment
- Setting Breakpoints
- Debugging PL/SQL
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

The new, integrated PL/SQL debugger in ODT 10.2.0.2 allows you to remain inside of Visual Studio for end-to-end debugging of your .NET and Oracle solution. You can now debug PL/SQL code such as procedures and functions (both stand-alone and packaged), object methods, and triggers from within the Visual Studio environment in the same way as you would debug your C# or VB.NET code. With the ODT integrated PL/SQL debugger, you can use traditional debugging features such as setting breakpoints, viewing and modifying variable values, and examining the call stack.

You use the PL/SQL debugger in one of three modes:.
<table>
<thead>
<tr>
<th><strong>Direct Database Debugging</strong></th>
<th>The Direct Database Debugging mode allows you to debug PL/SQL code directly from the Visual Studio environment. When you use the Direct Database Debugging mode, you do not need a Visual Studio solution or .NET code - you work directly with the PL/SQL code in the database. By right clicking on a procedure or function in Oracle Explorer, you can step into a PL/SQL procedure or function, or run to a breakpoint you have set. You can, of course, pass arguments to and receive return values from the PL/SQL code.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>External Application Debugging</strong></td>
<td>If you need to debug the PL/SQL code that is called by an application that is running outside of Visual Studio (for example an application that is running on a non-windows platform) use the External Application Debugging mode. This mode allows you to debug PL/SQL programs that are called by any application (built with Oracle client libraries 9.2 or later) located on any platform without requiring you to modify the application. It is most often used for debugging ASP.NET applications in the middle tier.</td>
</tr>
<tr>
<td><strong>Multi-tier Application Debugging</strong></td>
<td>The most powerful option during the development of a .NET application is the Multi-Tier Application Debugging mode. This mode allows you to seamlessly debug both .NET and PL/SQL code from within your Visual Studio solution. You can step directly from your .NET code into the PL/SQL code and back out again.</td>
</tr>
</tbody>
</table>

In this tutorial, you use the Multi-tier Application Debugging mode. This mode allows you, as a developer, to work with both .NET and PL/SQL code simultaneously during a debugging session.

### Creating an Oracle Connection

Before you begin creating your application, you open Visual Studio and create a connection. Perform the following steps:

1. Select **Start > Programs > Microsoft Visual Studio 2005 > Microsoft Visual Studio 2005**.

2. Select **View > Server Explorer**.

![Start Page - Microsoft Visual Studio](image)
3. Right-click **Data Connections** and select **Add Connection**.

![Server Explorer](image)

4. In Add Connection dialog box, first select the appropriate Data Source provider.

**Note:** If the Data source is already set to Oracle Database Server(Oracle ODP.NET), then move to next step or click **Change** to select the Data Source provider.

![Add Connection dialog box](image)

In the Change Data Source dialog box, select **Oracle Database Server** as the Data Source and **Oracle Data Provider for .NET** as the Data Provider. Click **OK**.
5. From the Data source name drop down, select the `<SID>` of your Oracle Database instance. Enter hr for the User name and Password, click **Save password** and click **Test connection**.
5. Test connection was successful. Click **OK**.

6. Click **OK**.
7. Your HR connection has been created. Similarly, you also need to create a SYS connection. Right-click **Data Connections** and select **Add Connection...**
8. The Data source provider should be set to Oracle Database Server (Oracle ODP.NET). From the Data source name drop down, select the `<SID>` of your Oracle Database instance.

Enter `sys` for the User name and enter your sys Password, click Save password, select SYSDBA for the Role and click Test connection.

![Add Connection dialog](image)

9. Test connection was successful. Click OK.
10. Click OK.

11. The SYS connection was created. Expand both hr.ORCL and SYS.ORCL to make the connection.
Granting Privileges for Debugging

Before you can use the debugger, you need to grant debug connect session and debug any procedure to the HR user. Perform the following steps:

1. In the Server Explorer panel, right-click SYS.ORCL connection and select **Query Window**.
2. Enter the following command and click **Execute**.

   **GRANT debug any procedure, debug connect session TO hr;**

   **Note:** If you are using Oracle Database Release 9.2, you should execute `grant debug any procedure to hr;`
3. The statement was executed successfully.

```
GRANT debug any procedure, debug connect session TO hr;
```

The command(s) successfully completed.

Creating the PL/SQL Package and Package Body

In this topic, you create a PL/SQL Package and Package Body that determines whether each number in a PL/SQL array is a prime number and then create a new row in the JOBS table using a PL/SQL record. Perform the following steps:

1. In the Server Explorer panel, right-click HR.ORCL and select Query Window.
2. Copy and paste the code in the PLSQL_Debug_Package.txt file in the codesnippets folder on your desktop into the Query Window and click **Execute**.

```plsql
CREATE OR REPLACE PACKAGE "HR"."OBE" IS
    -- types for associative arrays that client will pass as arguments
    TYPE "T_IN_VALUES" IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
    TYPE "T_OUT_VALUES" IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;

    -- procedure that accepts two associative arrays
    -- determines if an element is likely prime and
    -- sets value in output array
    PROCEDURE "DETERMINE_PRIMES" ("P_IN_VALUES" IN T_IN_VALUES,
        "P_OUT_VALUES" OUT T_OUT_VALUES);

    -- function that determines if a number is likely prime
    FUNCTION "IS_PRIME" ("P_NUMBER" IN NUMBER) RETURN NUMBER;

    -- constants used to return values from function
    IS_NOT_A_PRIME CONSTANT NUMBER DEFAULT 0;
    IS_A_PRIME CONSTANT NUMBER DEFAULT 1;

    -- pl/sql record type for the jobs table
    "JOBS_REC" jobs%rowtype;

    -- pl/sql procedure to add new job to jobs table
    PROCEDURE "ADD_NEW_JOB" ("P_JOB_ID" IN JOBS.JOB_ID%TYPE,
        "P_JOB_TITLE" IN JOBS.JOB_TITLE%TYPE,
        "P_MIN_SALARY" IN JOBS.MIN_SALARY%TYPE,
        "P_MAX_SALARY" IN JOBS.MAX_SALARY%TYPE);

END "OBE";
```
3. The PL/SQL Package was executed successfully.
CREATE OR REPLACE PACKAGE BODY "HR"."OBE" IS
    -- procedure that processes the incoming associative arrays
    -- calls the method IS_PRIME to determine if element is likely prime
    PROCEDURE "DETERMINE_PRIMES" ("P_IN_VALUES" IN T_IN_VALUES,
                                    "P_OUT_VALUES" OUT T_OUT_VALUES) IS
    BEGIN
        -- loop through each element in the incoming array
        -- and set the value for the corresponding element
        -- in the out array
        for i in p_in_values.first..p_in_values.last loop
            p_out_values(i) := is_prime(p_in_values(i));
        end loop;
    END "DETERMINE_PRIMES";

    -- private function to determine if a number is likely prime
    FUNCTION "IS_PRIME" ("P_NUMBER" IN NUMBER) RETURN NUMBER IS
    l_sqrt          number := 0;
    l_sqrt_ceil     number := 0;
    l_divisor      number := 0;
    l_divisor_squared number := 0;
    begin
        -- prime numbers must be >= 2
        if p_number < 2 then
            return IS_NOT_A_PRIME;
        end if;
        -- only integers can be prime

if p_number != ceil(p_number) then
    return IS_NOT_A_PRIME;
end if;
-- 2 is the only even prime, so it is a special case
if p_number = 2 then
    return IS_A_PRIME;
end if;
-- eliminate all other even numbers
if mod(p_number,2) = 0 then
    return IS_NOT_A_PRIME;
end if;
-- if the sqrt of the number is an integer, the number is not prime
l_sqrt := sqrt(p_number);
if l_sqrt = l_sqrt _ceil then
    return IS_NOT_A_PRIME;
end if;
-- the number has passed the basic elimination tests and may be prime
-- loop through set of odd divisors to determine if number is prime
l_divisor := 3;
for i in 1..l_sqrt _ceil loop
    l_divisor_squared := l_divisor * l_divisor;
    -- if l_divisor is a factor of p_number, then not a prime
    if mod(p_number,l_divisor) = 0 and l_divisor_squared < p_number then
        return IS_NOT_A_PRIME;
    end if;
    -- no factor found, therefore number is likely a prime
    if l_divisor_squared > p_number then
        return IS_A_PRIME;
    end if;
    l_divisor := l_divisor + 2;
end loop;
END "IS_PRIME";
-- pl/sql procedure to add new job to jobs table
PROCEDURE "ADD_NEW_JOB" ("P_JOB_ID" IN JOBS.JOB_ID%TYPE,
    "P_JOB_TITLE" IN JOBS.JOB_TITLE%TYPE,
    "P_MIN_SALARY" IN JOBS.MIN_SALARY%TYPE,
    "P_MAX_SALARY" IN JOBS.MAX_SALARY%TYPE) IS
BEGIN
    -- use the package variable JOBS_REC to create new record
    jobs_rec.job_id := p_job_id;
    jobs_rec.job_title := p_job_title;
    jobs_rec.min_salary := p_min_salary;
    jobs_rec.max_salary := p_max_salary;
    -- insert the job record into the table
    insert into jobs (job_id, job_title, min_salary, max_salary)
    values (jobs_rec.job_id, jobs_rec.job_title,
        jobs_rec.min_salary, jobs_rec.max_salary);
END "ADD_NEW_JOB";
END "OBE";
end if;
   -- no factor found, therefore number is likely a prime
   if l_divisor_squared > p_number then
      return IS_A_PRIME;
   end if;
   l_divisor := l_divisor + 2;
end loop;

END "IS_PRIME";

-- PL/SQL procedure to add new job to jobs table
PROCEDURE "ADD_NEW_JOB" ("P_JOB_ID" IN JOBS.JOB_ID%TYPE,
   "P_JOB_TITLE" IN JOBS.JOB_TITLE%TYPE,
   "P_MIN_SALARY" IN JOBS.MIN_SALARY%TYPE,
   "P_MAX_SALARY" IN JOBS.MAX_SALARY%TYPE) IS

BEGIN
   -- use the package variable JOBS_REC to create new record
   jobs_rec.job_id := p_job_id;
   jobs_rec.job_title := p_job_title;
   jobs_rec.min_salary := p_min_salary;
   jobs_rec.max_salary := p_max_salary;
   -- insert the job record into the table
   insert into jobs (job_id, job_title, min_salary, max_salary)
   values (jobs_rec.job_id, jobs_rec.job_title,
      jobs_rec.min_salary, jobs_rec.max_salary);

END "ADD_NEW_JOB";
END "OBE";

5. The PL/SQL Package Body was executed successfully.
6. In the Server Explorer for the HR.ORCL Connection, expand Packages > OBE to see the list of objects that were created.
7. Right-click the OBE Package and select **Compile Debug**. This action enables debugging of the package.
8. The package compiled successfully. Close the Output window. And then click the **Start Page** tab.

```
BEGIN
  -- use the package variable JOBS_REC to create new record
  jobs_rec.job_id := p_job_id;
  jobs_rec.job_title := p_job_title;
  jobs_rec.min_salary := p_min_salary;
  jobs_rec.max_salary := p_max_salary;
  -- insert the job record into the table
  insert into jobs (job_id, job_title, min_salary, max_salary)
  values (jobs_rec.job_id, jobs_rec.job_title,
          jobs_rec.min_salary, jobs_rec.max_salary);
END "ADD_NEW_JOB";
END "OBE";
```

The command(s) successfully completed.

```
------- Compiling Procedure HR.OBE -------
ALTER PACKAGE "HR"."OBE" COMPILDE DEBUG PACKAGE
Success: 0 errors 0 warnings
```
1. In the Recent Projects area next to Create: select Project....

![Start Page](image)

2. Accept the default Project Type Visual C#, select the Template Console Application, enter plsqldebugobe1 for the Name and enter the directory where you want the files stored in the location field (Note: if the directory does not exist, it is created). Then click OK.
3. The project has been created. Now you can add a reference.
4. Select **Project > Add Reference...**

![Add Reference dialog box](image)

5. Scroll down the list of References and select **Oracle.DataAccess** and click **OK**.

![Oracle.DataAccess in Add Reference dialog box](image)
using System;
using System.Data;
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;

namespace plsqldebugob1
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        /// <returns></returns>
        [STAThread]
        static void Main(string[] args)
        {
            // constants used to represent values returned
            // from the pl/sql procedure call
            const int IS_NOT_A_PRIME = 0;
            const int IS_A_PRIME = 1;

            // display progress message
            Console.WriteLine("Testing array for prime numbers...
\n");

            // connection string: adjust for your environment
            string constr = "User Id=hr; Password=hr; Data Source=ORCL; enlist=false; pooling=false";

            // create and open connection object
            OracleConnection con = new OracleConnection(constr);
            con.Open();

            // create command object for the function call
            OracleCommand cmd = new OracleCommand();
            cmd.Connection = con;
            cmd.CommandText = "OBE.determine_primes";

            // set the proper command type
            cmd.CommandType = CommandType.StoredProcedure;

            // parameter object for the input array
            OracleParameter p_in_values = new OracleParameter();
            p_in_values.OracleDbType = OracleDbType.Decimal;
            p_in_values.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
            p_in_values.Value = new decimal[10]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
            p_in_values.Size = 10;
            p_in_values.Direction = ParameterDirection.Input;

            // parameter object for the output array
            OracleParameter p_out_values = new OracleParameter();
            p_out_values.OracleDbType = OracleDbType.Decimal;
            p_out_values.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
            p_out_values.Value = null;
        }
    }
}
p_out_values.Size = 10;
p_out_values.Direction = ParameterDirection.Output;

// add parameters to the collection
// they must be added in the proper
// order when using bind by position (the default)
cmd.Parameters.Add(p_in_values);
cmd.Parameters.Add(p_out_values);

// execute the pl/sql procedure to populate output array
cmd.ExecuteNonQuery();

// display results to console window
for (int i = 0; i < p_in_values.Size; i++)
{
    foreach (OracleParameter p in cmd.Parameters)
    {
        // the input array is treated as System.Decimal[]
        // within the .NET code
        if (p.Value is System.Decimal[])
        {
            Console.Write("The Number {0} ", ((p.Value as System.Decimal[])[i]).ToString());
        }

        // the output array is treated as OracleDecimal[]
        // within the .NET code
        if (p.Value is OracleDecimal[])
        {
            if (((p.Value as OracleDecimal[])[i]).ToInt32() == IS_NOT_A_PRIME)
            {
                Console.WriteLine("is not a prime number!");
            }
            else if (((p.Value as OracleDecimal[])[i]).ToInt32() == IS_A_PRIME)
            {
                Console.WriteLine("is a prime number!");
            }
        }
    }
}

// display a separator line
Console.WriteLine();

// display progress message
Console.WriteLine("Using PL/SQL record...\n");

// remove parameters from command collection
// and set new command text
cmd.Parameters.Clear();
cmd.CommandText = "obe.add_new_job";

// parameter object for the job_id
OracleParameter p_job_id = new OracleParameter();
p_job_id.Value = "IT_DBA";

// parameter object for the job_title
OracleParameter p_job_title = new OracleParameter();
p_job_title.Value = "Database Administrator";

// parameter object for the min_salary
OracleParameter p_min_salary = new OracleParameter();
p_min_salary.OracleDbType = OracleDbType.Decimal;
p_min_salary.Value = 10000;

// parameter object for the max_salary
OracleParameter p_max_salary = new OracleParameter();
p_max_salary.OracleDbType = OracleDbType.Decimal;
p_max_salary.Value = 15000;

// add parameters to collection
cmd.Parameters.Add(p_job_id);
cmd.Parameters.Add(p_job_title);
cmd.Parameters.Add(p_min_salary);
cmd.Parameters.Add(p_max_salary);

// execute the pl/sql procedure to add new job
cmd.ExecuteNonQuery();

// display simple message to indicate procedure completed
Console.WriteLine("New job successfully created!");

// display a separator line
Console.WriteLine();

// Simple prompt to prevent the console from closing
// when running from the IDE
Console.WriteLine("Press ENTER to continue...");
Console.ReadLine();

// clean up objects
p_max_salary.Dispose();
p_min_salary.Dispose();
p_job_title.Dispose();
p_job_id.Dispose();
p_out_values.Dispose();
p_in_values.Dispose();
cmd.Dispose();
con.Dispose();
}
Configuring the Debugging Environment

There are a few properties and options that need to be configured before you can use the debugging environment. Perform the following steps:

```csharp
using System;
using System.Data;
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;

namespace plsqldebugoble1
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            // constants used to represent values returned
            // from the pl/sql procedure call
            const int IS_NOT_A_PRIME = 0;
            const int IS_A_PRIME = 1;

            // display progress message
            Console.WriteLine("Testing array for prime numbers...\n");

            // connection string: adjust for your environment
            string constr = "User Id=hr; Password=hr; Data Source=orcl; e:");

            // create and open connection object
            OracleConnection con = new OracleConnection(constr);
            con.Open();
        }
    }
}
```
1. Select **Project > plsqldebugobe1 Properties...**

2. Select **Debug**.
3. Deselect **Enable the Visual Studio hosting process** and right-click the **plsqldebugobe1** tab and select **Save Selected Items**.

4. Select **Tools > Options**....
5. Scroll down and expand **Oracle Developer Tools**. Select **PL/SQL Debugging**. In the Available database connections, click the checkbox in front of the **HR.ORCL** connection.

Notice that you see the TCP/IP port range. During PL/SQL debugging, the Oracle Database connects to Visual Studio via TCP/IP on a random port within this range. Make sure this range represents open ports on your machine and that they are not blocked by a firewall.

Click **OK**.
6. Select **Tools > Oracle Application Debugging**.

If you select **Tools > Oracle Application Debugging** again, you should see a checkmark in front of the menu item.
Setting Breakpoints

In this topic, you set some breakpoints in both your C# and PL/SQL code to stop the debugger during execution. Perform the following steps:

1. Click the **Program.cs** tab and locate the `cmd.ExecuteNonQuery()` statement.
2. In the Program.cs code, at the `cmd.Parameters.Add(p_out_values)` statement, you want to create a breakpoint. Click anywhere on the line, right-click and select Breakpoint, then select Insert Breakpoint. Alternatively, you can click on the grey leftmost edge of the code window at the location where you would like to set the breakpoint.

3. The breakpoint indicator is shown.
4. Create another breakpoint after the package is executed.

5. In the Oracle Explorer, double-click **DETERMINE_PRIMES** to open the code.
6. Create a breakpoint at the first statement after the BEGIN statement in the DETERMINE_PRIMES procedure.

7. From the list of procedures, select **ADD_NEW_JOB**.
8. Create a breakpoint at the first statement after the BEGIN statement.

9. Click the Program.cs tab.
Debugging PL/SQL

In this topic, you execute the program using the debugger. Perform the following steps:

1. Now you are ready to debug your program. Select **Debug > Start Debugging**...

   ![Debug Menu]

2. The debugger stops at the first breakpoint. To see the variables and their values, click the **Local** tab at the bottom of the window. Have a look at the code leading up to this breakpoint to understand how an array bind parameter is set up.
3. To move to the next line, click the **Step Over** icon.
4. You are at the statement that executes the PL/SQL Procedure. Click the Step Over icon.
5. The next breakpoint is reached in the DETERMINE_PRIMES procedure. Notice that the P_IN_VALUES is an array of length 10 so the loop will be performed 10 times. Click the **Continue** icon a couple of times to see the values in the Local window change.
6. Click on the + icon next to the P_IN_VALUES variable name in the **Locals** window to view the contents of the input array. This is the array of values that was passed into this stored procedure from the C# application. Continue stepping through the code for a while. You can also expand the P_OUT_VALUES array to watch as it gets filled with values that are eventually returned by this stored procedure.
7. Click the **Call Stack** tab in the lower right window. By examining the call stack, you can determine the code path to the current point in the execution of your program.
8. Disable this breakpoint so the program continues to the next breakpoint. Right-click the breakpoint and select Disable Breakpoint. You can also click directly on the red circle breakpoint icon to delete the breakpoint.
9. Click the **Continue** icon so that the program executes until the next Breakpoint.
10. The code performs the logic to produce what gets displayed to the user. Click **Step Over** a few more times.
11. Click the **Continue** icon again to go to the next Breakpoint.
12. The ADD_NEW_JOB procedure breakpoint is reached.
13. If you want to view the JOBS_REC PL/SQL record global variable, you need to create a watch. Select JOBS_REC and right-click, then select **Add Watch**. Then select the **Watch** tab at the bottom of the window.
14. The Watch window is a built-in window in Visual Studio that allows you to examine specific program variables. Expand the JOBS_REC Watch.
15. Click **Step Over** 4 times to see the Watch get populated. Then click **Continue**.
package body "hr"."OBE"

    end loop;
    end "IS_PRIME";
-- pl/sql procedure to add new job to jobs table
PROCEDURE "ADD_NEW_JOB" ("P_JOB_ID" IN JOBS.JOB_ID%TYPE,
    "P_JOB_TITLE" IN JOBS.JOB_TITLE%TYPE,
    "P_MIN_SALARY" IN JOBS.MIN_SALARY%TYPE,
    "P_MAX_SALARY" IN JOBS.MAX_SALARY%TYPE)

BEGIN
    -- use the package variable JOBS_REC to create new record
    jobs_rec.job_id := p_job_id;
    jobs_rec.job_title := p_job_title;
    jobs_rec.min_salary := p_min_salary;
    jobs_rec.max_salary := p_max_salary;
    -- insert the job record into the table
    insert into jobs (job_id, job_title, min_salary, max_salary)
        values (jobs_rec.job_id, jobs_rec.job_title,
            jobs_rec.min_salary, jobs_rec.max_salary);

    end "ADD_NEW_JOB";
    end "OBE";

Watch 1

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobs_rec</td>
<td>{HR.OBE.JOBS.Rowtype}</td>
<td>HR.OBE.JOBS</td>
<td></td>
</tr>
<tr>
<td>JOB_ID</td>
<td>&quot;IT_DBA&quot;</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>JOB_TITLE</td>
<td>&quot;Database Administrator&quot;</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>MIN_SALARY</td>
<td>10000</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>MAX_SALARY</td>
<td>15000</td>
<td>NUMBER</td>
<td></td>
</tr>
</tbody>
</table>
When the program finishes executing the results are shown.

The Number 1 is not a prime number!
The Number 2 is a prime number!
The Number 3 is a prime number!
The Number 4 is not a prime number!
The Number 5 is a prime number!
The Number 6 is not a prime number!
The Number 7 is a prime number!
The Number 8 is not a prime number!
The Number 9 is not a prime number!
The Number 10 is not a prime number!

Using PL/SQL record...
New job successfully created!
Press ENTER to continue...

Summary

In this tutorial, you learned how to:

- Add a connection to the database and grant privileges for debugging
- Create a package and package body
- Create a project and configure the debug environment
- Debug PL/SQL

Move your mouse over this icon to hide all screenshots.
Optimizing Data Access
Performance with ODP.NET
Optimizing Data Access Performance with ODP.NET

Purpose

This tutorial describes how to optimize the performance of your .NET application using ODP.NET.

Time to Complete

Approximately 20 minutes

Topics

This tutorial covers the following topics:

- Overview
- Prerequisites
- Using Statement Caching
- Using FetchSize and RowSize
- Passing Array Parameters
- Using Batch SQL, REF Cursors, and Multiple Active Result Sets (MARS)
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

The Oracle database has numerous methods to optimize query execution and data retrieval from clients. ODP.NET enables these performance optimizations, allowing .NET developers to build more efficient database programs. This tutorial covers some of the most commonly used performance tuning methods for ODP.NET developers.

The first section discusses statement caching. Statement caching eliminates the need to re-parse each executed SQL or PL/SQL statement by caching server cursors created during the initial statement execution. Subsequent executions of the same statement can reuse the parsed information from the cursor, and then execute the statement without re-parsing, for better performance. In order to see performance gains from statement caching, Oracle recommends caching only those statements that will be repeatedly executed. Furthermore, SQL or PL/SQL statements should use parameters rather than literal values. Doing so takes full advantage of statement caching since parsed information from parameterized statements can be reused even if the parameter values change in subsequent executions.

The second section shows how to control the amount of data retrieved per database round trip by using the FetchSize and RowSize properties in ODP.NET. It can be inefficient to make multiple round trips to retrieve one distinct set of data. These two properties allow developers to fine-tune the amount of data to retrieve from the database server so that fewer database
round trips need to be used.

The third section demonstrates how to pass a PL/SQL associative arrays between ODP.NET and the Oracle database. PL/SQL associative arrays allow Oracle to pass large amounts of data of the same datatype to .NET arrays. This method provides a flexible and easy way to batch data together in a single parameter to pass between the two tiers as needed.

The fourth section shows how to batch SQL, use the Oracle REF Cursor, and demonstrates multiple active result sets (MARS). Batching SQL allows developers to execute multiple SQL statements in a single database round trip. The Oracle REF Cursor is a data type unique to the Oracle database. It provides unique result set handling capabilities that allow for flexible data handling and tuning. MARS has been a feature that Oracle has always supported. This allows for a single connection to have more than one result set active at the same time.

Note: Performance gains seen in this demonstration may not necessarily reflect results seen in production environments. At times, real-world results may be better or worse than the demonstration's results, depending on differences in set up, hardware, database configuration, etc. For example, if you run this demonstration with the client and server on the same machine, improvements gained from reducing the number of database round trips will be greater in a production environment where the client and server are generally on two different machines.

Prerequisites

Before you perform this tutorial, you should:

- Install and create an Oracle Database 11g or Oracle Database 10g database server
- Install Visual Studio .NET 2003 or above
- Install ODP.NET from the Oracle Database Client or OTN

Using Statement Caching

Statement caching is beneficial when you need to repeatedly execute the same SQL or PL/SQL statement. ODP.NET will cache the most recently used statements. The developer decides the number and which of the most recently statements to cache. Cached statements improve performance by retaining the statement's parse tree and allowing it to be quickly looked up. To use ODP.NET statement caching, perform the following steps:
1. Open **Visual Studio**. Select **File > Open > Project/Solution**. Navigate to the **codesnippets\OptimizingDotNet** folder on your Desktop, select the **ODP.NET Performance Best Practices** solution and click **Open**.

2. Select **View > Solution Explorer**.
3. Double-click **ODP.NET Best Practices.cs** to open the code for this tutorial.

4. Review your connection string to allow the ODP.NET application to connect to Oracle's sample HR schema. You may have to modify the Password and Data Source values for the `conString` String variable.

   Note: If you are accessing a remote Oracle Database instance, then you need to specify the Data Source in `//<hostname>/<SID>` format.

```csharp
namespace ConsoleApplication1
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            string conString = "User Id=hr; Password=hr; Data Source=orcl;";
        }
    }
}
```

Scroll down to the Demo 1 section of the code. Note, the code file is interspersed with comments to mark areas where you need to enter code in the following steps. So, make sure you enter the relevant code under the relevant comment section. This is done to ensure the code is entered in proper sequence and to avoid mixing up of code in the file.
5. To see how statement caching works, you first want to execute a statement 10000 times with Statement Cache Size set to 0.

Set the Statement Cache Size to 0. Append the following code to the con1.ConnectionString statement, as shown in the screenshot below:

"Statement Cache Size=0";

```csharp
// Demo 1: Statement Caching
// Let's see the performance improvement when we repeatedly
// execute the same SQL statement with statement caching
// first off and then on.
// You should see significant performance improvement as
// statements repeatedly executed do not need to be re-parsed.

// Let's create a connection and command object to test
// performance *without* statement caching.
try
{
    OracleConnection con1 = new OracleConnection();
    con1.ConnectionString = connectionString + "Statement Cache Size=0";

    con1.Open();

    OracleCommand cmd1 = con1.CreateCommand();

    //SELECT on the EMP table will be executed repeatedly and timed.
    cmd1.CommandText = "select * from employees";
}
```

6. Further down in the same file, the same code is executed, but with statement caching on. Set Statement Cache Size to 1 and re-execute the statement 10000 times. Scroll down and append the following code to the con2.ConnectionString statement, as shown in the screenshot below:

"Statement Cache Size=1";

The application calculates the performance percentage improvement with caching enabled for the statement.
7. Now that you have reviewed the code, you can execute the application to see the results. Select **Build > Build Solution**.

The output window should not show any errors or warnings. The code should compile successfully.

8. Select **Debug > Start Debugging** or press **F5**.
A command window opens. Wait for the output. Do not close the window.

9. Note that your results may differ slightly from the results in the screenshot.

![Image showing command window output]

As you can see, there can be a substantial performance improvement when using statement caching for SQL or PL/SQL statements that are repeatedly executed. To enable statement caching across all your ODP.NET applications, you can enable statement caching in the Windows Registry as well.

Do not press Enter. To continue program execution, you add code in the program in subsequent topics. Close the command window.

Using FetchSize and RowSize

Controlling the amount of data fetched per database round trip can optimize application performance. For example, it is inefficient to retrieve only half a row of data per round trip when the end user needs to use the entire row. Instead of making two round trips, one round trip should have been made for optimum performance. ODP.NET allows developers to discover automatically the query row size and then specify the number of rows to be retrieved per round trip. This feature makes optimizing data retrieval much simpler for .NET programmers.

In this topic, you run a query that fetches one row at a time and then run the same query that fetches 100 rows at a time to see the performance improvement from making fewer round trips. Perform the following steps:
1. You first execute a query with FetchSize set to 1 row on the OracleDataReader.

Note: When ODP.NET executes an OracleCommand, it first retrieves metadata on how large the row size is and populates the RowSize property. A developer can then set the number of rows to retrieve from the database per round trip by multiplying RowSize by that number. In this example, you are only retrieving one row per round trip.

Scroll down to the Demo 2 section of the code. Locate the for loop. Enter the following code:

```csharp
reader.FetchSize = cmd.RowSize * 1;
```

```csharp
// Demo 2: Fetch Size and Row Size
// Let's see the performance difference when we set the FetchSize
// differently for the same SQL statement
// You should see significant performance improvement as number
// of round trips decrease with increased FetchSize.

// Create a connection to Oracle
OracleConnection con = new OracleConnection();
con.ConnectionString = conString;
con.Open();
// Create a command within the context of the connection
// Use the command to display employee names and salary from
// the EMP table
OracleCommand cmd = con.CreateCommand();
cmd.CommandText = "select * from all_objects where rownum < 100";

// Execute the command and set the FetchSize to 1 row on the DataReader
start_time = DateTime.Now;

for (int i = 0; i < 100; i++)
{
    OracleDataReader reader = cmd.ExecuteReader();

    // Add "reader.FetchSize = cmd.RowSize * 1;" here
    reader.FetchSize = cmd.RowSize * 1;
}
```
2. Since RowSize is determined at runtime, the query or underlying schema can change, but the same number of rows will still be retrieved per round trip. You can compare time difference between when FetchSize is set to 100 rows in this step and the time when FetchSize was set to one row.

Scroll down to the next for loop. Enter the following code:

```csharp
reader.FetchSize = cmd.RowSize * 100;
```

```csharp
//Add "reader.FetchSize = cmd.RowSize * 100;" here
reader.FetchSize = cmd.RowSize * 100;
while (reader.Read())
{
    reader.Dispose();
}
end_time = DateTime.Now;
ts = end_time - start_time;
ts2 = Math.Round(ts.TotalSeconds, 3);
Console.WriteLine("Fetch Size = 100: "+ ts2 + " seconds");
Console.WriteLine();
```

3. **Note:** You can comment out the for loops in the Demo 1 section you ran in the previous topic to avoid spending time on the demo you already executed. Commenting out just the loops will not affect the results for the remaining tutorial topics. It saves you time from repeatedly executing the loops again.

If you do not comment the for loops in the Demo 1 section, then press Enter after you get Demo 1 section's results to continue program execution for this section.

Now that you have reviewed the code, you can execute the application to see the results. Select **Build > Build Solution**.

4. Select **Debug > Start Debugging** or press **F5**.
5. Note that your results may differ slightly from the results in the screenshot.

Do not press Enter. To continue program execution, you add code in subsequent topics. Close the command window.

Passing Array Parameters

In this topic, you learn how to pass array parameters between .NET and Oracle. Array parameters allow large amounts of data to be passed as a single parameter. This example uses a stored procedure that takes an input array parameter and returns an output array parameter.

1. In the Solution Explorer, right-click array.sql and select Open.

2. Review the procedure. Note that a PL/SQL package MYPACK is created with a PL/SQL stored procedure MYSP. PL/SQL must be used since a PL/SQL associative array is used. The stored procedure has a PL/SQL associative array input parameter that just passes its values back as an output PL/SQL associative array. Select all the code in the file and select ctrl+c to copy it to the clipboard.
3. Select **View > Server Explorer.**
4. You need to create a data connection for the HR user. You can review how to do this in Oracle Developer Tools for Visual Studio .NET tutorial. Right-click the HR.ORCL connection and select Query Window. In this example, ORCL is the database alias, which may be different from the database alias on your system.

![Server Explorer](image)

5. Press [Ctrl]+V to paste the code in the Query Window. Select all the text in the Query Window and click Execute.

![Query Editor](image)
6. Once the stored procedure has executed successfully, switch back to **ODP.NET Best Practices.cs** code to review the code.

![Query Editor](image)

BEGIN
  Param3(1) := Param1(1);
  Param3(2) := Param1(2);
  Param3(3) := Param1(3);
END MYSP;
END MYPACK;

Query Output

The command(s) successfully completed.
The command(s) successfully completed.

7. Scroll down to Demo 3 section. Enter the following code:

```csharp
    cmd = new OracleCommand("MYPACK.MYSP", con);
    cmd.CommandType = CommandType.StoredProcedure;

    OracleParameter param1 = cmd.Parameters.Add("param1", OracleDbType.Varchar2);
    OracleParameter param2 = cmd.Parameters.Add("param2", OracleDbType.Varchar2);
    param1.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
    param2.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
```

Note: ODP.NET sets up the call to the MYPACK.MYSP stored procedure and binds its array parameters. The arrays' elements contain values of data type Varchar2. To indicate that arrays are passed in and out of the stored procedure, ODP.NET sets the parameters as PL/SQL associative arrays, which is an Oracle database collection data type.
8. Now, setup the parameter direction (input or output) and then specify the value for each parameter. Set the param2 to NULL.

Enter the following code:

```csharp
param1.Direction = ParameterDirection.Input;
param2.Direction = ParameterDirection.Output;
param2.Value = null;
```

```csharp
//Setup the parameter direction
//Note that param2 is NULL

param1.Direction = ParameterDirection.Input;
param2.Direction = ParameterDirection.Output;
param2.Value = null;
```

9. Then you specify the maximum number of elements in the arrays (in this case, 3) and the maximum size of the varchar2 (in this case, 20 characters).

Enter the following code:

```csharp
param1.Size = 3;
param2.Size = 3;
```

To execute the stored procedure and output the results, enter the following code:

```csharp
cmd.ExecuteNonQuery();
```
for(int i=0; i<3; i++)
{
    Console.Write((param2.Value as OracleString[])[i]);
    Console.WriteLine();
}

//Specify the maximum number of elements in the arrays
// and the maximum size of the varchar2
param1.Size = 3;
param2.Size = 3;

//Execute the statement and output the results

cmd.ExecuteNonQuery();
for (int i = 0; i < 3; i++)
{
    Console.Write((param2.Value as OracleString[])[i]);
    Console.WriteLine();
}

10. Note: You can comment out the for loops in the Demo 1 and Demo 2 sections you ran in the previous topics to avoid spending time on the demos you already executed. Commenting out just the loops will not affect the results for the remaining tutorial topics. It saves you time from repeatedly executing the loops again.

If you do not comment the for loops in the Demo 1 and Demo 2 sections, then press Enter twice (after results for previous demos are shown) to continue program execution for this section.

Now that you have reviewed the code, you can execute the application to see the results. Select Build > Build Solution.

11. Select Debug > Start Debugging or press F5.
Using Batch SQL, REF Cursors, and Multiple Active Result Sets (MARS)

Many times you want to execute multiple SQL statements in batches to save the number of database round trips. In ODP.NET this is accomplished via anonymous PL/SQL. Anonymous PL/SQL can be used to batch any number of queries, updates, inserts, and, deletes.

In this topic, you use anonymous PL/SQL to execute three queries in a single database round trip. To retrieve the results, you use Oracle REF Cursors to show how data can be flexibly retrieved from the database. With REF Cursors, you can choose to retrieve only data you require from the result set. Lastly, you see that ODP.NET can maintain multiple active result sets open for the same connection. Perform the following steps:

1. For ODP.NET to batch SQL, anonymous PL/SQL is used.

   **Note:** Anonymous PL/SQL is just a string that starts with a "BEGIN" and terminates with an "END;". In between lies the SQL that needs to be executed in a single round trip.

In this example, you batch three queries to return three result sets. Parameters are bound so that ODP.NET can return the query results to the client. Oracle executes anonymous PL/SQL like any other text command. As such, the command is bound as CommandType.Text.

Scroll down to the Demo 4 section of the code. Enter the following code:

```csharp
string cmdtxt = "BEGIN " +
"OPEN :1 for select FIRST_NAME, DEPARTMENT_ID from EMPLOYEES where DEPARTMENT_ID = 10; " +
"OPEN :2 for select FIRST_NAME, DEPARTMENT_ID from EMPLOYEES where DEPARTMENT_ID = 20; " +
"OPEN :3 for select FIRST_NAME, DEPARTMENT_ID from EMPLOYEES where DEPARTMENT_ID = 30; " +
"END;";

cmd = new OracleCommand(cmdtxt, con);
cmd.CommandType = CommandType.Text;
```
2. You then bind three output REF Cursors to the command. In this case you select employees in department 10, 20 and 30 each in a different REF Cursor.

Enter the following code:

```csharp
OracleParameter p1 = cmd.Parameters.Add("refcursor1", OracleDbType.RefCursor);
p1.Direction = ParameterDirection.Output;

OracleParameter p2 = cmd.Parameters.Add("refcursor2", OracleDbType.RefCursor);
p2.Direction = ParameterDirection.Output;

OracleParameter p3 = cmd.Parameters.Add("refcursor3", OracleDbType.RefCursor);
p3.Direction = ParameterDirection.Output;
```

```csharp
// ODP.NET has native Oracle data types, such as Oracle REF
// Cursors, which can be mapped to .NET data types
// Bind REF Cursor Parameters for each department

OracleParameter p1 = cmd.Parameters.Add("refcursor1", OracleDbType.RefCursor);
p1.Direction = ParameterDirection.Output;

OracleParameter p2 = cmd.Parameters.Add("refcursor2", OracleDbType.RefCursor);
p2.Direction = ParameterDirection.Output;

OracleParameter p3 = cmd.Parameters.Add("refcursor3", OracleDbType.RefCursor);
p3.Direction = ParameterDirection.Output;
```
3. Execute the batched statements and then retrieve the data from the 2nd and 3rd parameter without having to fetch the results from the first parameter. This is one of the benefits of using REF Cursors whereby you can retrieve data only when the client requires it. With REF Cursors, you can selectively choose data within a result set to fetch as well. With MARS, you can observe that both OracleDataReaders are actively reading data at the same time.

Enter the following code after the relevant comments:

```csharp
cmd.ExecuteNonQuery();

OracleDataReader dr1 = ((OracleRefCursor)cmd.Parameters[2].Value).GetDataReader();
OracleDataReader dr2 = ((OracleRefCursor)cmd.Parameters[1].Value).GetDataReader();
```

Add a while loop to retrieve both DataReaders at one time to test if MARS works. Enter the following code to output the results of the two REF Cursors:

```csharp
while (dr1.Read() && dr2.Read())
{
    Console.WriteLine("Employee Name: " + dr1.GetString(0) + ", " +
                      "Employee Dept:" + dr1.GetDecimal(1));
    Console.WriteLine("Employee Name: " + dr2.GetString(0) + ", " +
                      "Employee Dept:" + dr2.GetDecimal(1));
    Console.WriteLine();
}
```

// Execute batched statement
cmd.ExecuteNonQuery();

// Let's retrieve data from the 2nd and 3rd parameter without
// having to fetch results from the first parameter
// At the same time, we'll test MARS with Oracle
OracleDataReader dr1 =
    ((OracleRefCursor)cmd.Parameters[2].Value).GetDataReader();
OracleDataReader dr2 =
    ((OracleRefCursor)cmd.Parameters[1].Value).GetDataReader();

// Let's retrieve both DataReaders at one time to test if
// MARS works
while (dr1.Read() && dr2.Read())
{
    Console.WriteLine("Employee Name: " + dr1.GetString(0) + ", " +
                      "Employee Dept:" + dr1.GetDecimal(1));
    Console.WriteLine("Employee Name: " + dr2.GetString(0) + ", " +
                      "Employee Dept:" + dr2.GetDecimal(1));
    Console.WriteLine();
}
4. Now that you have reviewed the code, you can execute the application to see the results. Select **Build > Build Solution**.

5. Select **Debug > Start Debugging** or press **F5**.

6. Examine the results.

   - **Employee Name**: Den, **Employee Dept**: 30
   - **Employee Name**: Michael, **Employee Dept**: 20
   - **Employee Name**: Alexander, **Employee Dept**: 30
   - **Employee Name**: Pat, **Employee Dept**: 20

   Press 'Enter' to continue

---

**Summary**

In this tutorial, you learned how to:

- Use statement caching
- Use fetchsize and rowsize
- Pass array parameters
- Use batch SQL, REF Cursors, and Multiple Active Result Sets (MARS)
Using Database Change Notification With ODP.NET and Oracle Database
Using Database Change Notification With ODP.NET

Purpose

This tutorial introduces the Database Change Notification feature supported by the Oracle Data Provider for .NET and Oracle database.

Time to Complete

Approximately 30 minutes

Topics

This tutorial covers the following topics:

- Overview
- Prerequisites
- Granting Privileges to HR
- Loading and Running the Change Notification Demo Application
- Modifying the Employees Table and Watching a Change Notification Event Occur
- Adding a Callback Routine to Update Client Data After a Change Notification
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

The Oracle Data Provider for .NET provides a notification framework that supports Database Change Notification in the database server. Database Change Notification enables applications to receive notifications from a registered callback routine when there is a change in a query result set, schema objects, or the state of the database that could affect the results of a cached client result set.

Using this feature in ODP.NET, an application can maintain the validity of the client-side cache (for example, the ADO.NET DataSet) easily. The client does not need to maintain a connection to the database. It will receive notifications even when no longer connected to the database.

This feature is intended for queries whose results change infrequently over time and for those who cannot maintain a large number of active database connections.
In this tutorial, an application will display a DataSet grid containing the values of the HR.EMPLOYEES table. After populating the grid, the application will disconnect from the database and await notification that data has changed. Once the change is detected, the application will reconnect to the database and refresh the data.

This tutorial demonstrates the flexibility of Database Change Notification feature, including persisting notification registrations even after numerous changes and distinguishing at the row level whether a change would affect the client's query results. This feature is called Database Change Notification (DCN) in Oracle Database 10g Release 2. In Oracle Database 11g, this feature was renamed to Continuous Query Notification (CQN). Both these names refer to the same feature.

**Prerequisites**

Before you perform this tutorial, you should:

- Install and create an Oracle Database 11g or Oracle Database 10g Release 2 database server
- Install Visual Studio 2005 with Visual C#
- Install ODP.NET 11g with the Oracle Database 11g Client or from OTN

**Granting Privileges to HR**

A user or role must have CHANGE NOTIFICATION privileges prior to being able to run applications that accept Change Notification callbacks.

1. From your windows program menu, start Visual Studio. In Visual Studio, from **View** menu select **Server Explorer**.

   ![Start Page - Microsoft Visual Studio](image)

   A Server Explorer panel opens.
2. Right click **Data Connections** and select **Add Connection** from the menu.

An Add Connection dialog box opens.
3. In the Add Connection dialog box, click "Change" to select the appropriate data source and the data source provider. In the Change Data Source dialog box, select Oracle Database Server from the Data Source list and select Oracle Data Provider for .NET from the Data Provider drop down list. Click OK.

4. In the Add Connection dialog box, make sure the Connection Details tab is selected. From the Data Source drop down list, select the Oracle Database instance SID you are using.

Enter user name as SYS, password as oracle and select the SYSDBA role from the dropdown list. Click OK to connect.

Note: Your sys password may not be oracle. Check with your database administrator for the sys password.
5. Similarly, create a connection for HR schema. In Server Explorer, right click **Data Connections** and select **Add Connection**.

In the Add Connection dialog box, the appropriate data source and data source provider is already selected.

In the Connection Details tab, enter user name as **HR** and password as **hr**. Click **OK**.

**Note:** Your **HR** schema's password may not be **hr**. Check with your database administrator for the HR schema's password.
The Server Explorer shows the two connections.
6. Right click **SYS.ORCL** connection node in Server Explorer and select **Query Window**. This will open the Query Window.

7. In the Query Window, enter the following grant statement:

   ```sql
   grant change notification to hr
   ```

   Click the **Execute Query** button.

   You get a successfully completed message in the query output window. Close the Query Editor.
Loading and Running the Change Notification Demo Application

A Change Notification Visual Studio solution has been precreated for you to load and run.

This program will select all records in the EMPLOYEES table in the HR schema where the EMPLOYEE_ID value is less than 200, then place the results in an ADO.NET DataSet. The DataSet does not maintain a connection to the database server, so it no longer holds onto any database resources, allowing for better scalability.

Before the SELECT statement is executed, an OracleDependency object is bound to an OracleCommand. When the SELECT statement is executed, this OracleDependency will create a notification registration on the database server.

Even though the original DataSet is disconnected, when another user modifies the EMPLOYEES table, the notification handler will inform the .NET application a change to the source data has occurred. On the client side, this will trigger an event handler that refreshes the DataSet results.
1. In Visual Studio, from File menu, select Open > Project/Solution. In the Open project dialog box, navigate to the codesnippets\ChangeNotification folder on your Desktop, select DB Change Notification solution file. Click Open.

2. From View menu, select Solution Explorer. In the Solution Explorer, right click Form1.cs and select View Code.
3. You need to modify connection string variables to connect to Oracle. If your data source is a single local instance with the SID as `orcl` then you need not change the Data Source connection string, `constr`.

If you need to use a data source alias, modify the Data Source value in the connection string variable, "`constr`".

**Note:** You will connect to the HR schema, so check that User Id is set to HR and Password is set correctly. In the example code, HR's password is "hr". If this is not your HR schema's password, check with your DBA for the correct password.
Add the following code under their respective comments. The comments explain what each line of code does.

```csharp
4. AddRowid = true;
OracleDependency dep = new OracleDependency(cmd);
cmd.Notification.IsNotifiedOnce = false;
dep.OnChange += new OnChangeEventHandler(OnMyNotification);
```

// Add ROWID to the query to identify the specific rows that 
// have been changed. ROWID is a pseudo-column in every Oracle 
// table that uniquely identifies each row.

```
cmd.AddRowid = true;
```

// Register notification with command object if result changes. 
// When an OracleDependency instance is bound to an OracleCommand 
// instance, an OracleNotificationRequest is created and is set in the 
// OracleCommand's Notification property. This indicates subsequent 
// execution of command will register the notification.

```
OracleDependency dep = new OracleDependency(cmd);
```

// Allow the change notification handler in the database to persist 
// even after the first database change

```
cmd.Notification.IsNotifiedOnce = false;
```

// Add the event handler to handle the notification. 
// The OnMyNotification method will be invoked when a notification 
// message is sent from the database

```
dep.OnChange +=
    new OnChangeEventHandler(OnMyNotification);
```
5. In the Visual Studio, from **Debug** menu, select **Start Debugging**. This will run the demo. If any break points are hit, press **F5** to continue running until a grid containing the values of the HR. EMPLOYEES table appears.

![Employee Table](image)

**Note:** When you run this program for the first time, you may receive the message as shown in the screenshot below from Windows Firewall. Click on **Unblock** to allow the change notifications to be sent from the database server to the client.

![Windows Security Alert](image)

Do not exit the running demo.
Now the Change Notification application is running and ready to receive notifications that data in the HR.EMPLOYEES table has been changed. Let's modify the EMPLOYEES table data.

1. Click the (+) plus sign next to the HR.ORCL node in Server Explorer to expand it. Expand the Tables node.

Right click the EMPLOYEES table and select Retrieve Data... to show its data in Visual Studio.

A window opens showing the employees table data.
2. In the newly opened Data Window in Visual Studio (NOT in the Change Notification application grid), modify one of the SALARY values. Click Save to cause the value to be committed to Oracle Database.
3. In a second or two, the Change Notification application opens a Notification Alert window, informing that the result set has changed.

Note: See the taskbar at the bottom of the desktop if you don't see the window. It may be hidden. (If a breakpoint fires before seeing this window, press F5 to continue)

Click OK. Close the Oracle Database Change Notification Demo.
Adding a Callback Routine to Update Client Data After a Change Notification

Right now, our application receives the Change Notification callback, but our event handler contains nothing more than the dialog box you saw in the previous section. Let's add code to connect to the database and fetch the rows that have been modified.

For single table SELECTs, the Change Notification callback will return the ROWID information for all modified rows in the table. The ROWID will be appended to the original query so that you can retrieve only the rows that were changed, not the entire data set which would be less optimal. You will then refresh the DataSet with just the new data.

1. Click the `Form1.cs` tab. In the dropdown box select `OnMyNotificaton(..`.

This will take you to the `OnMyNotificaton` callback routine.

```csharp
// Receiving Change Notification indicates results are now invalid.
// Refresh the results.
public static void OnMyNotificaton(object src, OracleNotificationEventArgs args)
{
    MessageBox.Show("Result set has changed.", "Notification Alert",
    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

    // Append on to the sqlUpdate statement if there are additional
    // updated rows

    // Refresh changed data
```
2. Add following lines of code under their respective comments. The comments explain what each line of code does.

```csharp
string sqlUpdate = sql;

for (int i = 1; i < args.Details.Rows.Count; i++)
{
    DataRow detailRow = args.Details.Rows[i];
    string rowid = detailRow["Rowid"].ToString();
    sqlUpdate = sqlUpdate + " or rowid = \'' + rowid + '\'';
}
```

3. Add following lines of code to refresh the data.

```csharp
OracleConnection con2 = new OracleConnection(constr);
OracleCommand cmd2 = new OracleCommand(sqlUpdate, con2);
con2.Open();
OracleDataAdapter da2 = new OracleDataAdapter(cmd2);
da2.Fill(ds, tablename);
```
4. From **Build** menu, select **Build Solution**.

Ensure there are no errors or warnings reported.
5. From **Debug** menu, select **Start Debugging**. A grid containing the values of the HR. EMPLOYEES table appears.

6. You will now change multiple rows in the database and see ODP.NET update only those rows that were changed.

   In the Solution Explorer, double click **multiple_row_change.sql**.
7. Select the following code from the `multiple_row_change.sql`. Press CTRL+C to copy the code.

```
update employees set salary=salary+10 where employee_id=100 or employee_id=101;
commit;
```

Right-click HR.ORCL and select Query Window. In the Query Window, paste the code and make sure both the statements are selected. Click **Execute Query** to run the query.

```
2 row(s) affected.
The command(s) successfully completed.
```
8. You get the Notification Alert. Click OK.

Check the modified rows refreshed in the application grid.

Note: The Oracle database identifies the changed rows via their ROWIDs. The OracleNotificationEventArgs instance passes the ROWID information back to ODP.NET. Only the rows that have changed are retrieved, not the entire table. This saves the application from refreshing data that is still valid.

In addition, you will notice that for each distinct database change, you receive one notification. Even though two rows were modified, the modification occurred as one discrete event. As such, only one notification was delivered.

Close the Oracle Database Change Notification Demo window.

Summary

In this tutorial, you learned how to:
- Grant privileges to HR for Change Notification
- Load and run the Change Notification Demo Application
- Modify the Employees table and watch a Change Notification event occur
- Add a callback routine to handle the Change Notification Event
Developing and Deploying a .NET Stored Function
Developing and Deploying a .NET Stored Function

Purpose

This tutorial describes how to deploy a .NET stored function using Oracle Developer Tools for Visual Studio 2005.

Time to Complete

Approximately 30 minutes

Topics

This tutorial covers the following topics:

- Overview
- Prerequisites
- Creating an Oracle Connection
- Creating a .NET Stored Function
- Changing the ODP.NET Reference
- Deploying the .NET Stored Function
- Running the .NET Stored Function
- Summary

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

With Oracle Database 10g Release 2 on Windows, Oracle now offers Oracle Database Extensions for .NET. This database option makes it possible to deploy stored procedures and functions written in a .NET managed language into the Oracle database on Windows.

Prerequisites

Before you perform this tutorial, you should:

1. Perform the Installing Oracle Database 10g on Windows tutorial.
2. Perform the [Installing Oracle Developer Tools for VS .NET](#) tutorial.

3. Perform the [Configuring the .NET Stored Procedures Environment](#) tutorial.

## Creating an Oracle Connection

Before you begin creating your application, you open Visual Studio and examine the interface. Perform the following steps:

1. Select **Start > Programs > Microsoft Visual Studio 2005 > Microsoft Visual Studio 2005**.

2. Select **View > Server Explorer**.
3. Right-click **Data Connections** and select **Add Connection**.

4. Enter your Data source name. Enter hr for the User name and Password. Click **Save password**, select **Default** as the Role, and click **Test Connection**.
5. Test connection succeeded. Click **OK**.
6. Click OK.

7. Your connection has been created. Expand HR.ORCL. Now you can create your Stored Function.
Creating a .NET Stored Function

In this topic, you create an Oracle project to store the function in the database. Perform the following steps:

1. From the **Start Page** tab, click **Project...** adjacent to Create: in the Recent Projects area.
2. Expand the Visual C# Project type, click Database and select the Oracle Project template. Enter Project1 in the Name field and select the directory where you want the files stored (Note: if the directory does not exist, it is created). Then click OK.

The Project1 class object is shown.
3. Delete `public static void StoredProcedure1()` and its curly braces ({})) from the code and copy and paste the text located in file `getdepartmentno.txt`. This file is located in the codesnippets folder on the Desktop.

```csharp
public static int getDepartmentno(int employee_id)
{
    int department_id = 0;

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

    // Create and execute a command
    OracleCommand cmd = con.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;

}
4. Right-click the **Class1.cs** tab and select **Save Class1.cs**.

![Class1.cs](Image)

---

**Changing the ODP.NET Reference**

Since you are using Oracle Database 10g in this Hands-on Lab, you need to use ODP.NET 10.2 in your stored procedure. The Oracle Project will default to using ODP.NET version 11. You need to change the reference to use ODP.NET 10.2. Perform the following steps:

1. In the Solution Explorer window, expand **References**. (Note: If you can not see the Solution Explorer window, select **View > Solution Explorer**.)

![Solution Explorer](Image)
2. Right-click **Oracle.DataAccess** and select **Remove**.

3. Right-click **References** and select **Add Reference**...
4. In the Add Reference window, select the **Browse** tab.

5. Browse to `e:\oracle\product\10.2.0\db_1\odp.net\bin\2.x` to select `Oracle.DataAccess.Dll` and click **OK**.
Deploying a .NET Stored Function

In this topic, you deploy a .NET Stored Function into the Database. Perform the following steps:

1. Select **Build > Deploy Solution**. This task compiles the code and then launches the .NET deployment wizard.
2. The Oracle Deployment Wizard for .NET appears. Click **Next**.
3. You need to establish a connection with SYSDBA credentials. Click **New Connection**.
4. Select **ORCL** as the Data source name, enter **sys** for the User name and **<syspassword>** for the Password. Select **Save password** and click **Test Connection**.
5. Test connection succeeded. Click **OK**.
6. Click OK to save your connection.

7. Click Next.
8. Make sure **Copy assembly and generate stored procedures** is selected and click **Next**.
9. Accept the defaults and click **Next**.
10. Accept the defaults and click **Next**.
11. Expand **Project1 > Class1** and select the check box next to **Int32 getDepartmentno**. Then select the down arrow next to schema.
12. Select HR from the list of Schemas and click Next.
13. At the Summary window, click **Finish**.
14. Your .NET Stored Function has been deployed successfully.
Running the .NET Stored Function

In this topic, you execute the .NET Stored Function you just deployed. In this case, the function retrieves the department number for the employee id given. Perform the following steps:
1. Expand HR.ORCL > Functions. Right-click GETDEPARTMENTNO and select Run. If you do not see GETDEPARTMENTNO, right-click Functions and click Refresh.

2. Enter a value of 100 (or any other valid Employee ID) and click OK.
Note: If you receive an error, stop and start the OracleOraDb10g_home1ClrAgent. Then try again.

3. The Department id 90 of the Employee 100 was selected.

Summary

In this tutorial, you learned how to create, deploy and run a .NET Stored Function.

Place the cursor over this icon to hide all screenshots.