

3

Retrieving Data from Oracle Using ODP.NET

We have several methodologies to retrieve information from Oracle using ODP.NET. Sometimes, we may have to use few of the ODP.NET classes together with few of the ADO.NET classes to develop .NET applications efficiently.

In this chapter, we will concentrate on the following:

- Executing queries with `OracleCommand`
- Retrieving data using `OracleDataReader`
- Retrieving data using `OracleDataAdapter`
- Working with `DataTable` and `DataSet` when offline (disconnected mode)
- Using `DataTableReader` with `DataTable`
- Bind variables using `OracleParameter`
- Performance techniques

If you would like to work with stored procedures to retrieve data, you should skip to Chapter 5 (provided you are familiar with all the concepts discussed here).

Fundamental ODP.NET Classes to Retrieve Data

To retrieve data from an Oracle database using ODP.NET, we need to work with a few of the ODP.NET classes. At this point, we will discuss the most fundamental classes available in ODP.NET for retrieving data.

The following is the list of fundamental ODP.NET classes:

- `OracleConnection`
- `OracleCommand`
- `OracleParameter`
- `OracleDataReader`
- `OracleDataAdapter`

The `OracleConnection` class provides the means to connect to the Oracle database. We have already used this class several number of times in the previous chapter. It connects to Oracle database and performs all the operations we need to carry out. Without this class, we would never be able to perform any database operation. It also manages transactions and connection pooling.

The `OracleCommand` class is mainly used to execute commands against Oracle database. It supports the execution of SQL commands (like `SELECT`, `INSERT`, and `CREATE`), stored procedures, etc. We can even specify table or view names (without even providing a `SELECT` statement) to retrieve the rows available through them. It works in conjunction with `OracleConnection` to connect to Oracle database.

The `OracleParameter` class is complementary to the `OracleCommand` class to provide run-time parameters along with their values to SQL queries or stored procedures. You can even work with different types of stored-procedure parameters like `IN`, `OUT`, or `IN OUT`. It is also mostly used whenever you want to execute the same SQL command frequently or continuously.

The `OracleDataReader` class is simply a read-only and forward-only result set. As the data retrieved using this class is non-updatable and only forward-navigable, this is the fastest retrieval mechanism available. The most important point to remember while using `OracleDataReader` is that it needs a dedicated connection to Oracle database while it retrieves information. It is best used to fill in drop-down lists, data grids, etc. It works in conjunction with `OracleCommand` to connect to and retrieve information from Oracle database.

The `OracleDataAdapter` class is mainly used to populate datasets or data tables for offline use (disconnected use). The `OracleDataAdapter` simply connects to the database, retrieves the information (or data), populates that information into datasets or data tables, and finally disconnects the connection to the database. It works with `OracleConnection` to connect to Oracle database. It can also work with `OracleCommand` if necessary.

A data table is very similar to a disconnected result set (or record set). A dataset is simply a set of data tables along with their relations (if available). A dataset is a kind of small scale in-memory RDBMS, which gets created on demand.

`DataTable` and `DataSet` are the two classes for these in ADO.NET that are used in combination with `OracleDataAdapter`. The data in a dataset (or data table) can be modified offline (in disconnected mode) and later can be updated back to the database using the same `OracleDataAdapter`. In simple words, `OracleDataAdapter` works as a bridge between offline data (or a dataset) and Oracle database.

Retrieving Data Using `OracleDataReader`

`OracleDataReader` is simply a read-only and forward-only result set. It works only if the database connection is open and it makes sure that the connection is open while you are retrieving data. As the data that it retrieves is read-only, it is a bit faster than any other method to retrieve data from Oracle.

You need to work with `OracleCommand` together with `OracleConnection` to get access to `OracleDataReader`. There is an `ExecuteReader` method in the `OracleCommand` class, which gives you the `OracleDataReader`.

Retrieving a Single Row of Information

Let us start by retrieving a single row from Oracle database using ODP.NET and populate the data into few textboxes on a WinForm.

To connect to and work with Oracle database, we need to start with `OracleConnection`. Once a connection to the database is established, we need to issue a `SELECT` statement to retrieve some information from the database. A query (or any SQL command) can be executed with the help of an `OracleCommand` object. Once the `SELECT` statement gets executed, we can use `OracleDataReader` to retrieve the information.

The following code accepts an employee number from the user and gives you the details of that employee:

```
Imports Oracle.DataAccess.Client

Public Class Form1

    Private Sub btnGetEmployee_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        btnGetEmployee.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x; _
                                         User Id=scott;Password=tiger")
    Try
        Dim SQL As String
        'build the SELECT statement
```

```
SQL = String.Format("SELECT ename, sal, job FROM
                    emp WHERE empno={0}", Me.txtEmpno.Text)
'create command object to work with SELECT
Dim cmd As New OracleCommand(SQL, cn)
'open the connection
cmd.Connection.Open()
'get the DataReader object from command object
Dim rdr As OracleDataReader = _
cmd.ExecuteReader(CommandBehavior.CloseConnection)
'check if it has any rows
If rdr.HasRows Then
    'read the first row
    rdr.Read()
    'extract the details
    Me.txtEname.Text = rdr("ename")
    Me.txtSal.Text = rdr("sal")
    Me.txtJob.Text = rdr("job")
Else
    'display message if no rows found
    MessageBox.Show("Not found")
End If
'clear up the resources
rdr.Close()
Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
'close the connection if it is still open
If cn.State = ConnectionState.Open Then
    cn.Close()
End If
End Try
End Sub

End Class
```

As explained earlier, the above program creates an `OracleConnection` object as follows:

```
Dim cn As New OracleConnection("Data Source=x; _
                                User Id=scott;Password=tiger")
```

Next, we need to create an `OracleCommand` object by providing a `SELECT` query and the connection object (through which it can connect to the database):

```
Dim SQL As String
SQL = String.Format("SELECT ename, sal, job FROM
                    emp WHERE empno={0}", Me.txtEmpno.Text)
Dim cmd As New OracleCommand(SQL, cn)
```

Once the `OracleCommand` object is created, it is time to open the connection and execute the `SELECT` query. The following does this:

```
cmd.Connection.Open()
Dim rdr As OracleDataReader = _
    cmd.ExecuteReader(CommandBehavior.CloseConnection)
```

You must observe that the query gets executed using the `ExecuteReader` method of `OracleCommand` object, which in turn returns an `OracleDataReader` object. In the above statement, the `ExecuteReader` method is specified with `CommandBehavior.CloseConnection`, which simply closes the database connection once the `OracleDataReader` and `OracleCommand` are disposed.

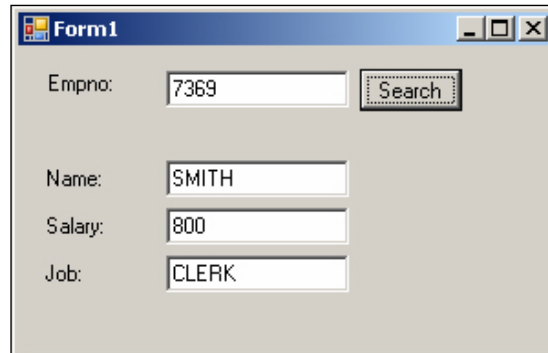
We can use the `HasRows` property of `OracleDataReader` to test whether the reader retrieved any rows or not. If any rows are retrieved, we can read each successive row using the `Read` method of `OracleDataReader`. The `Read` method returns a `Boolean` value to indicate whether it has successfully read a row or not. Once the `Read` succeeds, we can retrieve each value in the row with the column name as follows:

```
If rdr.HasRows Then
    'read the first row
    rdr.Read()
    'extract the details
    Me.txtEname.Text = rdr("ename")
    Me.txtSal.Text = rdr("sal")
    Me.txtJob.Text = rdr("job")
Else
    'display message if no rows found
    MessageBox.Show("Not found")
End If
```

Finally, we close the `OracleDataReader` object using the `Close` method as follows:

```
rdr.Close()
```

If it could read successfully, the output for this code would look similar to the following figure:



Using "Using" for Simplicity

The above program can be made simple by using the Using statement together with ODP.NET classes as follows:

```
Using cn As New OracleConnection("Data Source=x;
                                User Id=scott;Password=tiger")
Try
    cn.Open()
    Dim SQL As String
    SQL = String.Format("SELECT ename, sal,
                        job FROM emp WHERE empno={0}", Me.txtEmpno.Text)
    Using cmd As New OracleCommand(SQL, cn)
    Using rdr As OracleDataReader = cmd.ExecuteReader
        If rdr.HasRows Then
            'read the first row
            rdr.Read()
            'extract the details
            Me.txtEname.Text = rdr("ename")
            Me.txtSal.Text = rdr("sal")
            Me.txtJob.Text = rdr("job")
        Else
            'display message if no rows found
            MessageBox.Show("Not found")
        End If
    End Using
End Using
Catch ex As Exception
    MessageBox.Show("Error: " & ex.Message)
    If cn.State = ConnectionState.Open Then
```

```
        cn.Close()
    End If
End Try
End Using
```

The `Using` keyword is new in Visual Basic 2005, which internally generates `try` and `finally` blocks around the object being allocated and calls `Dispose()` for you saving you the hassle of manually creating it.

The objects created using the `Using` keyword are automatically erased (and respective resources would be automatically cleared) from the memory once it is out of using scope. Even though it is very flexible to use the `Using` statement, for the sake of clarity, we will go without using it in the examples of this book.

Retrieving Multiple Rows on to the Grid

In the previous section, we tried to retrieve only one row using `OracleDataReader`. In this section, we will try to retrieve more than one row (or a result set) and populate a `DataGridView` on a `WinForm`.

The following code lists out the details of all employees available in the `emp` table:

```
Imports Oracle.DataAccess.Client

Public Class Form2
    Private Sub btnGetEmployees_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        btnGetEmployees.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x;
                                        User Id=scott;Password=tiger")
    Try
        Dim SQL As String
        'build the SELECT statement
        SQL = String.Format("SELECT empno, ename, job,
                            mgr, hiredate, sal, comm, deptno FROM emp")
        'create command object to work with SELECT
        Dim cmd As New OracleCommand(SQL, cn)
        'open the connection
        cmd.Connection.Open()
        'get the DataReader object from command object
        Dim rdr As OracleDataReader = _
            cmd.ExecuteReader(CommandBehavior.CloseConnection)
        'check if it has any rows
        If rdr.HasRows Then
```

```
        With Me.DataGridView1
            'remove existing rows from grid
            .Rows.Clear()
            'get the number of columns
            Dim ColumnCount As Integer = rdr.FieldCount
            'add columns to the grid
            For i As Integer = 0 To ColumnCount - 1
                .Columns.Add(rdr.GetName(i), rdr.GetName(i))
            Next
            .AutoSizeColumnsMode =
                DataGridViewAutoSizeColumnsMode.ColumnHeader
            'loop through every row
            While rdr.Read
                'get all row values into an array
                Dim objCells(ColumnCount - 1) As Object
                rdr.GetValues(objCells)
                'add array as a row to grid
                .Rows.Add(objCells)
            End While
        End With
    Else
        'display message if no rows found
        MessageBox.Show("Not found")
        Me.DataGridView1.Rows.Clear()
    End If
    'clear up the resources
    rdr.Close()
Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
    'close the connection if it is still open
    If cn.State = ConnectionState.Open Then
        cn.Close()
    End If
End Try
End Sub
End Class
```

Except the highlighted section, the rest of the code is already explained as part of the previous section. You can observe that the SELECT statement now tries to retrieve all rows from emp as follows:

```
SQL = String.Format("SELECT empno, ename, job, mgr,
                    hiredate, sal, comm, deptno FROM emp")
```

Once the `OracleDataReader` is ready with rows, we need to start with clearing the rows already displayed in the `DataGridView` with the help of the following code:

```
With Me.DataGridView1
    'remove existing rows from grid
    .Rows.Clear()
```

Once the rows are cleared, the first issue is the header of the grid. The moment we add columns to the grid, the header row gets automatically populated (with the column names). Before adding columns to the header, we should know the number of columns being added (just for the loop iterations) with the `FieldCount` property of `DataGridView`. The following is the code fragment that finds the number of columns and adds the columns to `DataGridView`:

```
Dim ColumnCount As Integer = rdr.FieldCount
For i As Integer = 0 To ColumnCount - 1
    .Columns.Add(rdr.GetName(i), rdr.GetName(i))
Next
```

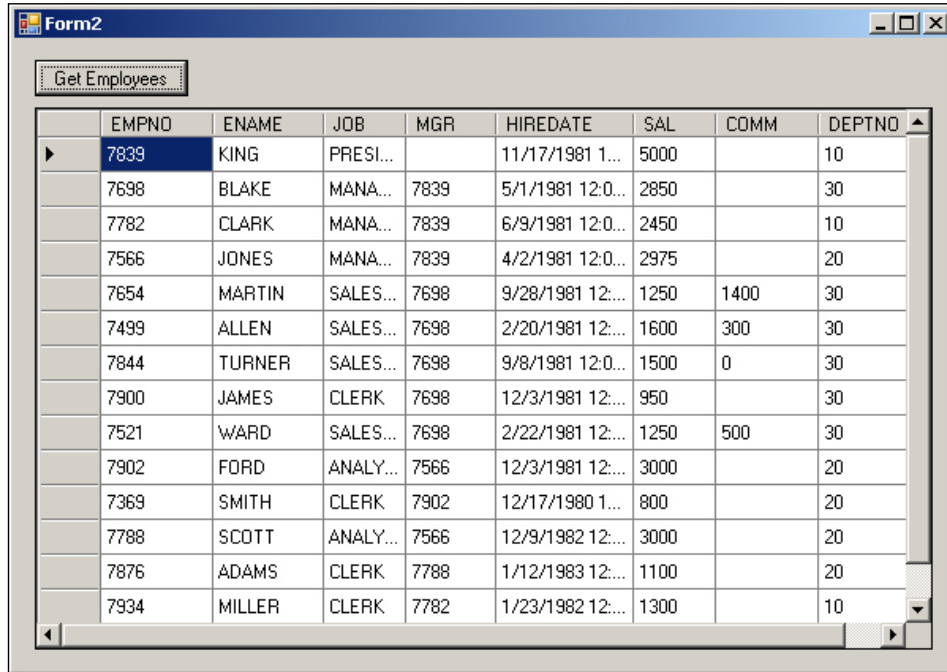
All the columns get auto-sized based on the column header with the following statement:

```
.AutoSizeColumnsMode =
    DataGridViewAutoSizeColumnsMode.ColumnHeader
```

Once the columns are added, we need to read every successive row from the `OracleDataReader` and add it to the `DataGridview`. To add all column values at a time, we make use of the `GetValues()` method of `OracleDataReader` to push all the values in to an array and finally add the array itself as a row to the `DataGridView`. The following code fragment accomplishes this.

```
While rdr.Read
    'get all row values into an array
    Dim objCells(ColumnCount - 1) As Object
    rdr.GetValues(objCells)
    'add array as a row to grid
    .Rows.Add(objCells)
End While
```

The output for this code would look similar to the following figure:



EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESI...		11/17/1981 1...	5000		10
7698	BLAKE	MANA...	7839	5/1/1981 12:0...	2850		30
7782	CLARK	MANA...	7839	6/9/1981 12:0...	2450		10
7566	JONES	MANA...	7839	4/2/1981 12:0...	2975		20
7654	MARTIN	SALES...	7698	9/28/1981 12:...	1250	1400	30
7499	ALLEN	SALES...	7698	2/20/1981 12:...	1600	300	30
7844	TURNER	SALES...	7698	9/8/1981 12:0...	1500	0	30
7900	JAMES	CLERK	7698	12/3/1981 12:...	950		30
7521	WARD	SALES...	7698	2/22/1981 12:...	1250	500	30
7902	FORD	ANALY...	7566	12/3/1981 12:...	3000		20
7369	SMITH	CLERK	7902	12/17/1980 1...	800		20
7788	SCOTT	ANALY...	7566	12/9/1982 12:...	3000		20
7876	ADAMS	CLERK	7788	1/12/1983 12:...	1100		20
7934	MILLER	CLERK	7782	1/23/1982 12:...	1300		10

Pulling Information Using Table Name

In all of the previous examples, the `SELECT` statement was used to retrieve a set of rows. The `SELECT` statement is a good choice if you would like to retrieve only specific columns or to include some complex combinations using sub-queries, joins etc. You can also retrieve a complete table (without using a `SELECT` statement) by setting the `CommandType` of `OracleCommand` to `TableDirect`. The following code demonstrates the use of `TableDirect`:

```
Imports Oracle.DataAccess.Client

Public Class Form2
    Private Sub btnGetEmployees_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        btnGetEmployees.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x; _
            User Id=scott;Password=tiger")
        Try
            Dim SQL As String
            'build the SELECT statement
```

```
Dim cmd As New OracleCommand("emp", cn)
cmd.CommandType = CommandType.TableDirect
cmd.Connection.Open()

...
End Sub
End Class
```

The default `CommandType` is `Text`, which accepts any SQL statement. When we change it to `TableDirect`, it accepts only a table name. Another command type available is `StoredProcedure`. It is mainly used when you want to execute stored procedures using an `OracleCommand` object. (Working with PL/SQL stored procedures is covered in Chapter 5.)

Retrieving Typed Data

While retrieving values from `OracleDataReader`, we can extract information available in individual columns (of a particular row) either by using column ordinal (position) values or column names.

Retrieving Typed Data Using Ordinals

ODP.NET provides data-specific enumerations through the namespace `oracle.DataAccess.types`. This is specially useful if you are trying to retrieve very specific data from the `OracleDataReader`.

For example, you can modify the code given previously to work with specific data types as following:

```
Me.txtEname.Text = rdr.GetOracleString(1)
Me.txtSal.Text = rdr.GetFloat(5)
Me.txtJob.Text = rdr.GetOracleString(2)
```

Here we provide ordinal values (column numbers starting from 0) to retrieve the data in a specific column. Apart from above data types, you also have the full support of every native data type existing in ODP.NET!

Retrieving Typed Data Using Column Names

The strategy of working with column ordinals will not be an issue as long as we know with what columns we are dealing with. But, sometimes, it is very dangerous to play with it. If the underlying table structure gets modified, our application becomes out of synch with the column ordinals. At the same time, using column ordinals can make your code very difficult to follow. It is always suggested not to go for column ordinals (unless we use it for looping purposes).

However, the typed methods only accept column ordinals as parameters. Fortunately, we can use the `GetOrdinal()` method to find the ordinal corresponding to a particular column name as demonstrated in the following:

```
Me.txtEname.Text =  
    rdr.GetOracleString(rdr.GetOrdinal("ename"))  
Me.txtSal.Text = rdr.GetFloat(rdr.GetOrdinal("sal"))  
Me.txtJob.Text =  
    rdr.GetOracleString(rdr.GetOrdinal("job"))
```

Working with Data Tables and Data Sets

The `OracleDataAdapter` class is mainly used to populate data sets or data tables for offline use. The `OracleDataAdapter` simply connects to the database, retrieves the information, populates that information into datasets or data tables, and finally disconnects the connection to the database. You can navigate through any of those rows in any manner. You can modify (add or delete) any of those rows in disconnected mode and finally update them back to the database using the same `OracleDataAdapter`.

A set of rows can be populated into a data table and a set of data tables can be grouped into a data set. Apart from grouping, a data set can also maintain offline relationships (using `DataRelation` between data tables existing in it).

`OracleDataAdapter` primarily works with `OracleConnection` to connect to Oracle database. It can also work with `OracleCommand` if necessary.

Retrieving Multiple Rows into a DataTable Using OracleDataAdapter

Now that we understand about `OracleDataAdapter`, let us try to use it to retrieve all the employees available in the `emp` table:

```
Imports Oracle.DataAccess.Client  
Public Class Form4  
  
    Private Sub btnGetEmployees_Click(ByVal sender As  
        System.Object, ByVal e As System.EventArgs) Handles  
        btnGetEmployees.Click  
        'create connection to db  
        Dim cn As New OracleConnection("Data Source=x; _  
                                         User Id=scott;Password=tiger")  
  
        Try  
            Dim SQL As String
```

```

'build the SELECT statement
SQL = String.Format("SELECT empno, ename, job,
mgr, hiredate, sal, comm, deptno FROM emp")
'create the dataadapter object
Dim adp As New OracleDataAdapter(SQL, cn)
'create the offline datatable
Dim dt As New DataTable
'fill the data table with rows
adp.Fill(dt)
'clear up the resources and work offline
adp.Dispose()
'check if it has any rows
If dt.Rows.Count > 0 Then
    'simply bind datatable to grid
    Me.DataGridView1.DataSource = dt
Else
    'display message if no rows found
    MessageBox.Show("Not found")
    Me.DataGridView1.Rows.Clear()
End If
Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
'close the connection if it is still open
If cn.State = ConnectionState.Open Then
    cn.Close()
End If
End Try
End Sub
End Class

```

Once the `OracleConnection` is established, we need to start with the `OracleDataAdapter` object as follows:

```

SQL = String.Format("SELECT empno, ename, job,
mgr, hiredate, sal, comm, deptno FROM emp")
Dim adp As New OracleDataAdapter(SQL, cn)

```

You can understand from the above that `OracleDataAdapter` can be used directly with a `SELECT` statement. You can also specify an `OracleCommand` object in place of a `SELECT` statement if necessary.

To place data offline, we need to either work with `DataSet` or `DataTable` objects. In this scenario, we will deal with a `DataTable` object, and it is created as follows:

```

Dim dt As New DataTable

```

Once the `DataTable` object is created, we need to fill up all the rows using the `OracleDataAdapter` object as follows:

```
adp.Fill(dt)
```

Once all the rows are available in the `DataTable` object (which will always be in memory), we can close (`dispose`) the `OracleDataAdapter` using the following statement:

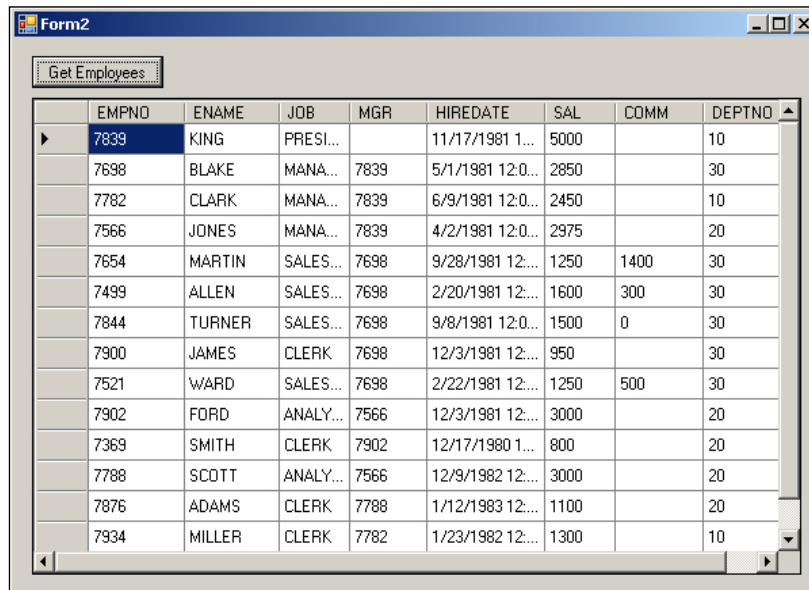
```
adp.Dispose()
```

The `DataTable` object contains a collection of `DataRow` objects corresponding to each row populated into it. We can retrieve the number of rows available in the `DataTable` object using the `DataTable.Rows.Count` property as follows:

```
If dt.Rows.Count > 0 Then
    'simply bind datatable to grid
    Me.DataGridView1.DataSource = dt
Else
    'display message if no rows found
    MessageBox.Show("Not found")
    Me.DataGridView1.Rows.Clear()
End If
```

In the above code fragment, we are assigning the `DataTable` object as `DataSource` to `DataGridView`. This would automatically populate entire `DataGridView` with all the column names (as part of the header) and all rows.

The output for the above code would look similar to the following figure:



The screenshot shows a Windows form titled "Form2" with a button labeled "Get Employees". Below the button is a `DataGridView` displaying a list of employees. The data is as follows:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESI...		11/17/1981 1...	5000		10
7698	BLAKE	MANA...	7839	5/1/1981 12:0...	2850		30
7782	CLARK	MANA...	7839	6/9/1981 12:0...	2450		10
7566	JONES	MANA...	7839	4/2/1981 12:0...	2975		20
7654	MARTIN	SALES...	7698	9/28/1981 12:...	1250	1400	30
7499	ALLEN	SALES...	7698	2/20/1981 12:...	1600	300	30
7844	TURNER	SALES...	7698	9/8/1981 12:0...	1500	0	30
7900	JAMES	CLERK	7698	12/3/1981 12:...	950		30
7521	WARD	SALES...	7698	2/22/1981 12:...	1250	500	30
7902	FORD	ANALY...	7566	12/3/1981 12:...	3000		20
7369	SMITH	CLERK	7902	12/17/1980 1...	800		20
7788	SCOTT	ANALY...	7566	12/9/1982 12:...	3000		20
7876	ADAMS	CLERK	7788	1/12/1983 12:...	1100		20
7934	MILLER	CLERK	7782	1/23/1982 12:...	1300		10

Filling a DataTable Using OracleDataReader

So far, we have been filling data tables using `OracleDataAdapter`. ADO.NET 2.0 gives us the flexibility to fill a data table using `OracleDataReader` as well. The following code gives you the details of all employees available in the `emp` table by filling a data table using an `OracleDataReader`:

```
Dim cn As New OracleConnection("Data Source=x; _
                               User Id=scott;Password=tiger")
Try
    Dim SQL As String
    Dim dt As New DataTable
    'build the SELECT statement
    SQL = String.Format("SELECT empno, ename, job,
                        mgr, hiredate, sal, comm, deptno FROM emp")
    'create command object to work with SELECT
    Dim cmd As New OracleCommand(SQL, cn)
    'open the connection
    cmd.Connection.Open()
    'get the DataReader object from command object
    Dim rdr As OracleDataReader = _
        cmd.ExecuteReader(CommandBehavior.CloseConnection)
    'check if it has any rows
    If rdr.HasRows Then
        'simply bind datatable to grid
        dt.Load(rdr, LoadOption.OverwriteChanges)
        Me.DataGridView1.DataSource = dt
    Else
        'display message if no rows found
        MessageBox.Show("Not found")
        Me.DataGridView1.Rows.Clear()
    End If
    rdr.Close()
Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
    'close the connection if it is still open
    If cn.State = ConnectionState.Open Then
        cn.Close()
    End If
End Try
```

Once the `OracleConnection` and `OracleDataReader` are created, we need to create and fill a `DataTable` object using `OracleDataReader` itself. The following is the statement that creates a `DataTable` object:

```
Dim dt As New DataTable
```

To fill the above `DataTable` object with respect to `OracleDataReader`, we can directly use the `Load` method of `DataTable`, which accepts a `DataReader` object and the type of `LoadOption`. The following statement loads the content of an `OracleDataReader` into a `DataTable` object with a `LoadOption` as `OverwriteChanges` (overwrites all the modifications that are available as part of the `DataTable` object):

```
dt.Load(rdr, LoadOption.OverwriteChanges)
```

Retrieving a Single Row of Information Using OracleDataAdapter

In the previous example, we worked with a set of rows in the `DataTable` object. Now, we shall work with a particular row using the `DataTable` object. The following code accepts an employee number from the user and gives you the details of that employee:

```
Imports Oracle.DataAccess.Client

Public Class Form3

    Private Sub btnGetEmployee_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnGetEmployee.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x; _
            User Id=scott;Password=tiger")
        Try
            Dim SQL As String
            'build the SELECT statement
            SQL = String.Format("SELECT ename, sal, job FROM
                emp WHERE empno={0}", Me.txtEmpno.Text)
            'create the dataadapter object
            Dim adp As New OracleDataAdapter(SQL, cn)
            'create the offline datatable
            Dim dt As New DataTable
            'fill the data table with rows
            adp.Fill(dt)
            'clear up the resources and work offline
            adp.Dispose()

            'check if it has any rows
```

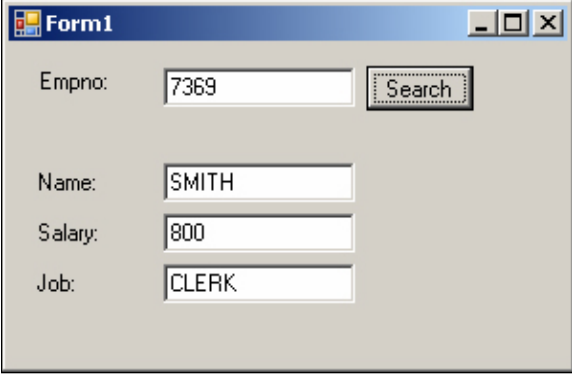
```
If dt.Rows.Count > 0 Then
    'extract the details
    Me.txtEname.Text = dt.Rows(0)("ename")
    Me.txtSal.Text = dt.Rows(0)("sal")
    Me.txtJob.Text = dt.Rows(0)("job")
Else
    'display message if no rows found
    MessageBox.Show("Not found")
End If

Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
    'close the connection if it is still open
    If cn.State = ConnectionState.Open Then
        cn.Close()
    End If
End Try
End Sub
End Class
```

Once the `DataTable` object is filled using `OracleDataAdapter`, we can directly retrieve a particular row using the row index. Once the row is fetched, we extract column values by providing column names for the rows as follows:

```
Me.txtEname.Text = dt.Rows(0)("ename")
Me.txtSal.Text = dt.Rows(0)("sal")
Me.txtJob.Text = dt.Rows(0)("job")
```

The output for the above code would look similar to the following figure:



The screenshot shows a Windows form titled "Form1" with a search interface. It contains four text boxes and a search button. The text boxes are labeled "Empno:", "Name:", "Salary:", and "Job:". The "Empno:" box contains the value "7369". The "Name:" box contains "SMITH". The "Salary:" box contains "800". The "Job:" box contains "CLERK". A "Search" button is located to the right of the "Empno:" box.

Field	Value
Empno:	7369
Name:	SMITH
Salary:	800
Job:	CLERK

Working with DataTableReader

DataTableReader is complementary to a DataTable object, and is mainly used as a type of *Data Reader* in the disconnected mode. The following is the modified code:

```
'create connection to db
Dim cn As New OracleConnection("Data Source=xe; _
                               User Id=scott;Password=tiger")
Try
    Dim SQL As String
    'build the SELECT statement
    SQL = String.Format("SELECT ename, sal, job FROM emp
                        WHERE empno={0}", Me.txtEmpno.Text)
    'create the DataAdapter object
    Dim adp As New OracleDataAdapter(SQL, cn)
    'create the offline datatable
    Dim dt As New DataTable
    'fill the data table with rows
    adp.Fill(dt)
    'clear up the resources and work offline
    adp.Dispose()
    Dim dtr As DataTableReader = dt.CreateDataReader

    'check if it has any rows
    If dtr.HasRows Then
        'read the first row
        dtr.Read()
        'extract the details
        Me.txtEname.Text = dtr("ename")
        Me.txtSal.Text = dtr("sal")
        Me.txtJob.Text = dtr("job")
    Else
        'display message if no rows found
        MessageBox.Show("Not found")
    End If

Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
    'close the connection if it is still open
    If cn.State = ConnectionState.Open Then
        cn.Close()
    End If
End Try
```

You can observe the highlighted code, which creates a `DataTableReader` object by calling the `CreateDataReader` method related to the `DataTable` object. Once the `DataTableReader` is created, we can directly retrieve the column values with the specified column names as follows:

```
Me.txtEname.Text = dtr("ename")
Me.txtSal.Text = dtr("sal")
Me.txtJob.Text = dtr("job")
```

Populating a Dataset with a Single Data Table

A dataset is simply a group of data tables. These data tables can be identified with their own unique names within a dataset. You can also add relations between data tables available in a dataset.

The following code gives you the details of all employees available in the `emp` table by populating a dataset with only a single data table using `OracleDataAdapter`:

```
Imports Oracle.DataAccess.Client
Public Class Form6

    Private Sub btnGetEmployees_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnGetEmployees.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x; _
            User Id=scott;Password=tiger")
    Try
        Dim SQL As String
        'build the SELECT statement
        SQL = String.Format("SELECT empno, ename, job,
            mgr, hiredate, sal, comm, deptno FROM emp")
        'create the dataadapter object
        Dim adp As New OracleDataAdapter(SQL, cn)
        'create the offline datatable
        Dim ds As New DataSet
        'fill the data set with a data table named emp
        adp.Fill(ds, "emp")
        'clear up the resources and work offline
        adp.Dispose()
        'check if it has any rows
        If ds.Tables("emp").Rows.Count > 0 Then
            'simply bind datatable to grid
            Me.DataGridView1.DataSource = ds.Tables("emp")
        End If
    End Try
End Sub
End Class
```

```
Else
    'display message if no rows found
    MessageBox.Show("Not found")
    Me.DataGridView1.Rows.Clear()
End If
Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
    'close the connection if it is still open
    If cn.State = ConnectionState.Open Then
        cn.Close()
    End If
End Try
End Sub
End Class
```

If you can observe the highlighted code in the above script, we are creating a new DataSet object, populating it with a DataTable named "emp" (which contains all the rows) and finally assigning the same DataTable to the grid. The output for the above code would look similar to the figure in the section *Retrieving Multiple Rows into a Data Table Using OracleDataAdapter*.

Populating a Dataset with Multiple Data Tables

Now, let us add more than one data table into a dataset. The following code retrieves a list of department details into a data table named `Departments` and another list of employee details into a data table named `Employees`:

```
Imports Oracle.DataAccess.Client
Public Class Form7

    Private Sub btnData_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnData.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x; _
            User Id=scott;Password=tiger")
    Try
        Dim ds As New DataSet
        Dim adp As OracleDataAdapter

        adp = New OracleDataAdapter("SELECT deptno,
            dname, loc FROM Dept", cn)
        adp.Fill(ds, "Departments")
    End Try
End Sub
End Class
```

```

    adp.Dispose()
    adp = New OracleDataAdapter("SELECT empno, ename,
                                job, mgr, hiredate, sal, comm, deptno FROM
                                Emp", cn)
    adp.Fill(ds, "Employees")
    adp.Dispose()

    Me.DataGridView1.DataSource = ds
    Me.DataGridView1.DataMember = "Departments"

    Me.DataGridView2.DataSource =
                                ds.Tables("Employees")
Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
    'close the connection if it is still open
    If cn.State = ConnectionState.Open Then
        cn.Close()
    End If
End Try
End Sub
End Class

```

From the above highlighted code, you can easily observe that we are retrieving two different result sets (identified by Departments and Employees) into the same dataset. The following code fragment creates the Departments data table:

```

    adp = New OracleDataAdapter("SELECT deptno, dname,
                                loc FROM Dept", cn)
    adp.Fill(ds, "Departments")
    adp.Dispose()

```

The following code fragment creates the Employees data table:

```

    adp = New OracleDataAdapter("SELECT empno, ename, job,
                                mgr, hiredate, sal, comm, deptno FROM Emp", cn)
    adp.Fill(ds, "Employees")
    adp.Dispose()

```

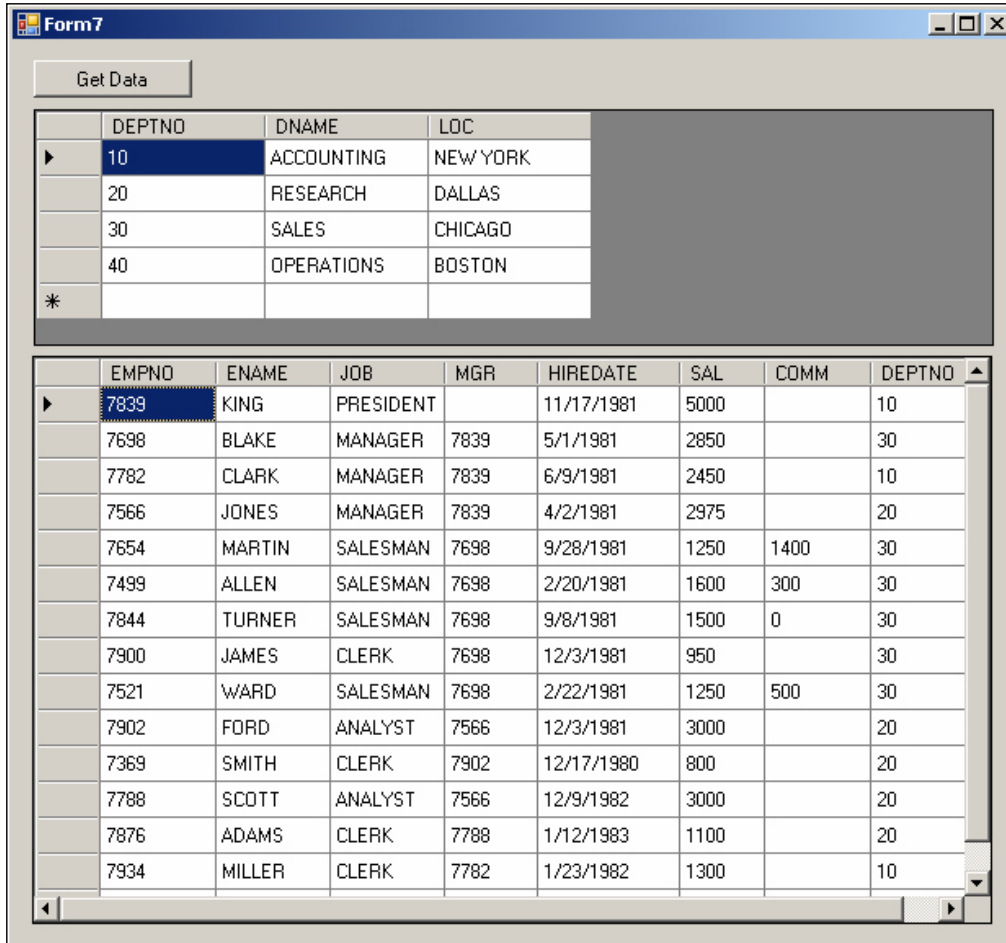
Those two result sets are automatically created as two data tables within the same dataset. Once the dataset is populated, we can present them with two different grids (two different methods) as follows:

```

    Me.DataGridView1.DataSource = ds
    Me.DataGridView1.DataMember = "Departments"
    Me.DataGridView2.DataSource = ds.Tables("Employees")

```

The output for this code would look similar to the following figure:



Presenting Master-Detail Information Using a Dataset

As mentioned before, a `DataSet` object can have its own relations between data tables existing in it. We can add these relations dynamically at the client side (within an application), to represent master-detail (or hierarchical) information. The following code gives the list of employees (in the bottom grid) based on the department you choose in the top grid:

```
Imports Oracle.DataAccess.Client
Public Class Form8

    Private Sub btnData_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnData.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=xe; _
            User Id=scott;Password=tiger")
    Try
        Dim ds As New DataSet
        Dim adp As OracleDataAdapter

        adp = New OracleDataAdapter("SELECT deptno,
            dname, loc FROM Dept", cn)
        adp.Fill(ds, "Departments")
        adp.Dispose()

        adp = New OracleDataAdapter("SELECT empno, ename,
            job, mgr, hiredate, sal, comm, deptno FROM
            Emp", cn)
        adp.Fill(ds, "Employees")
        adp.Dispose()

        ds.Relations.Add(New DataRelation("FK_Emp_Dept",
            ds.Tables("Departments").Columns("Deptno"),
            ds.Tables("Employees").Columns("Deptno")))
        Dim bsMaster As New BindingSource(ds, _
            "Departments")
        Dim bsChild As New BindingSource(bsMaster, _
            "FK_Emp_Dept")
        Me.DataGridView1.DataSource = bsMaster
        Me.DataGridView2.DataSource = bsChild

    Catch ex As Exception
        'display if any error occurs
        MessageBox.Show("Error: " & ex.Message)
        'close the connection if it is still open
        If cn.State = ConnectionState.Open Then
            cn.Close()
        End If
    End Try
End Sub
End Class
```

Once the DataSet is filled with data tables (Departments and Employees), we can add an in-memory relation using the following statement:

```
ds.Relations.Add(New DataRelation("FK_Emp_Dept",  
    ds.Tables("Departments").Columns("Deptno"),  
    ds.Tables("Employees").Columns("Deptno")))
```

The above statement simply adds a new relation (named FK_Emp_Dept) between two DataTable objects (Departments and Employees) based on the column Deptno (available in both DataTable objects).

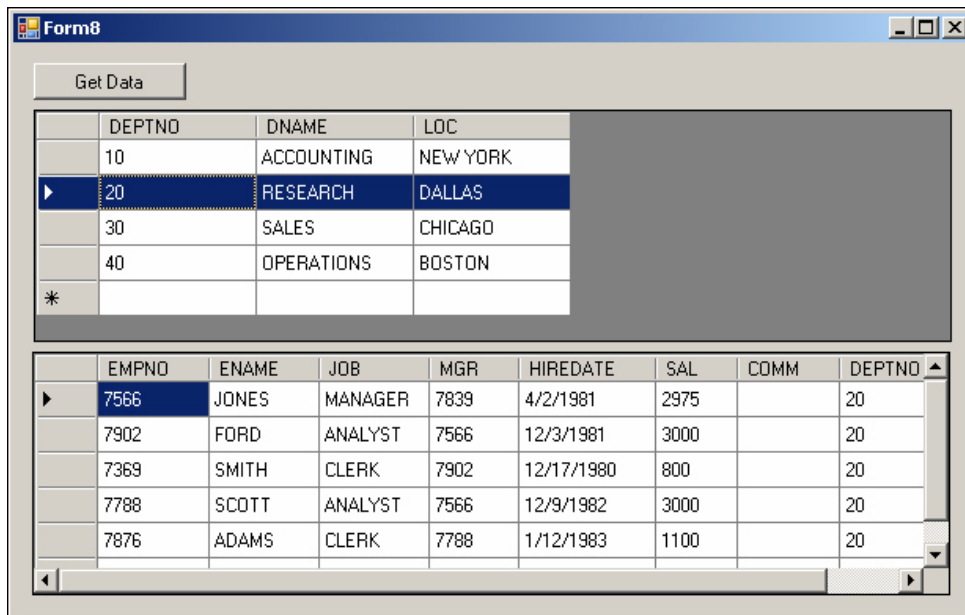
To present the information in a master-detail fashion, we can make use of the BindingSource object as follows:

```
Dim bsMaster As New BindingSource(ds, "Departments")  
Dim bsChild As New BindingSource(bsMaster, "FK_Emp_Dept")
```

In the above code fragment, we used two BindingSource objects corresponding to master and child data tables respectively. The child BindingSource object is created based on the master BindingSource object together with the specification of DataRelation. Once the BindingSource objects are ready, we can assign them as data sources to the DataGridView controls as following:

```
Me.DataGridView1.DataSource = bsMaster  
Me.DataGridView2.DataSource = bsChild
```

The output for the above code would look similar to the following figure:



You can observe that this screen displays only the employees working in department number 20 as that is selected in the top grid.

More About the OracleCommand Object

Till now, we have seen `OracleCommand` working with `OracleDataReader`. `OracleCommand` is not simply meant for `OracleDataReader`. It has got a lot of functionality for itself. Let us see few of the most commonly used features of `OracleCommand` in this section. We will further go into depth in subsequent sections and chapters.

Retrieving a Single Value from the Database

As we already covered working with single or multiple rows, we need to work on retrieving a single value from database very effectively. We have already retrieved row values in our previous examples, but those examples are more suitable when you are trying to deal with entire rows.

`OracleCommand` is equipped with a method called `ExecuteScalar`, which is mainly used to retrieve single values from the database very efficiently thus improving the performance. The following example focuses on this:

```
Imports Oracle.DataAccess.Client

Public Class Form9
    Private Sub btnEmployeeCount_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs) Handles
        btnEmployeeCount.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x; _
            User Id=scott;Password=tiger")
        Try
            'create the command object
            Dim cmd As New OracleCommand("SELECT COUNT(*) _
                FROM emp", cn)

            'open the connection from command
            cmd.Connection.Open()
            'execute the command and get the single value
            'result
            Dim result As String = cmd.ExecuteScalar
            'clear the resources
            cmd.Connection.Close()
            cmd.Dispose()
            'display the output
        End Try
    End Sub
End Class
```

```
        MessageBox.Show("No. of Employees: " & result)
    Catch ex As Exception
        'display if any error occurs
        MessageBox.Show("Error: " & ex.Message)
        'close the connection if it is still open
        If cn.State = ConnectionState.Open Then
            cn.Close()
        End If
    End Try
End Sub
End Class
```

The highlighted line in the above code simply executes the `SELECT` command, which retrieves the number of rows from the `emp` table and assigns this value to the result variable.

Handling Nulls when Executing with `ExecuteScalar`

The most important issue to remember is that `ExecuteScalar` simply returns an object type of data. The object refers to any data type within .NET. If the data type of your variable matches with the type of object returned by `ExecuteScalar`, an implicit (automatic) conversion takes place. There would not be a problem as long as the data types match. However, it would be a problem if the result is `NULL`. Let us have an example that accepts an employee number from the user and gives his or her commission:

```
Imports Oracle.DataAccess.Client

Public Class Form12

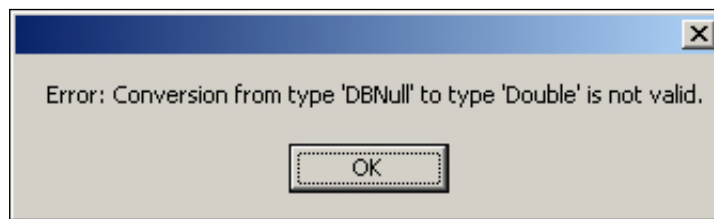
    Private Sub btnGetCommission_Click(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
    btnGetCommission.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=xe; _
            User Id=scott;Password=tiger")
    Try
        'create the command object
        Dim cmd As New OracleCommand("SELECT comm FROM _
            emp WHERE empno=" & Me.txtEmpno.Text, cn)
        'open the connection from command
        cmd.Connection.Open()
        'execute the command and get the single value
        'result
        Dim result As Double = cmd.ExecuteScalar
        cmd.Connection.Close()
    End Try
End Sub
End Class
```

```

        cmd.Dispose()
        'display the output
        MessageBox.Show("Commission: " & result)
    Catch ex As Exception
        'display if any error occurs
        MessageBox.Show("Error: " & ex.Message)
        'close the connection if it is still open
        If cn.State = ConnectionState.Open Then
            cn.Close()
        End If
    End Try
End Sub

```

In the highlighted statement above, we are expecting a numeric (or double) value as the result. If the `ExecuteScalar` returns a double value, it would never be a problem. What if it returns a NULL? The following is the error you would receive:



To deal with the above error, we may have to include our own condition to test against nulls in the output. Just replace the highlighted code above with the following two statements and it should work fine now:

```

Dim result As Object = cmd.ExecuteScalar
If IsDBNull(result) Then result = 0

```

You can observe from the above two lines that we are receiving the value in the form of an object and assigning a value zero if it is null.

Handling Nulls when Working with OracleDataReader

When we work with `OracleDataReader` (or for that matter, even with data rows in a data table), we may come across nulls. The following is the efficient way to deal in with such scenarios:

```

'create connection to db
Dim cn As New OracleConnection("Data Source=x; _
                                User Id=scott;Password=tiger")
Try

```

```
'create the command object
Dim cmd As New OracleCommand("SELECT comm FROM _
    emp WHERE empno=" & Me.txtEmpno.Text, cn)
'open the connection from command
cmd.Connection.Open()
'create the data reader
Dim rdr As OracleDataReader = _
    cmd.ExecuteReader(CommandBehavior.CloseConnection)
'check if it has any rows
If rdr.HasRows Then
    'read the first row
    rdr.Read()
    'extract the details
    Dim result As Double = IIf(IsDBNull(rdr("comm")), _
        0, rdr("comm"))
    MessageBox.Show("Commission: " & result)
Else
    'display message if no rows found
    MessageBox.Show("Not found")
End If
rdr.Dispose()
Catch ex As Exception
    'display if any error occurs
    MessageBox.Show("Error: " & ex.Message)
'close the connection if it is still open
If cn.State = ConnectionState.Open Then
    cn.Close()
End If
End Try
```

You can observe that we are making use of the IIF function in Visual Basic.NET to make the inline comparison. We can also use the `rdr.IsDBNull` method to achieve the same.

Working with Bind Variables together with OracleParameter

With the help of `OracleParameter`, you can include bind variables within any SQL statement. These bind variables are nothing but run-time query parameters. The values in the SQL statement are bound at run time when we use bind variables.

If the same SQL statement is being continuously used (with different values), it is recommended to work with bind variables. When you use bind variables in SQL statements, the statements would automatically cache at server level to improve performance during repeated database operations of the same type.

Following is a simple example that includes a bind variable in a SELECT statement followed by OracleParameter, which fills the bind variable with a value:

```
Imports Oracle.DataAccess.Client

Public Class Form11

    Private Sub btnGetEmployee_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnGetEmployee.Click
        'create connection to db
        Dim cn As New OracleConnection("Data Source=x; _
            User Id=scott;Password=tiger")
        Try
            'create command object to work with SELECT

            Dim cmd As New OracleCommand("SELECT empno, _
                ename, sal, job FROM emp WHERE empno=:empno", cn)
            cmd.Parameters.Add(New OracleParameter(":empno",
                Me.txtEmpno.Text))

            'open the connection
            cmd.Connection.Open()
            'get the DataReader object from command object
            Dim rdr As OracleDataReader = _
            cmd.ExecuteReader(CommandBehavior.CloseConnection)
            'check if it has any rows
            If rdr.HasRows Then
                'read the first row
                rdr.Read()
                'extract the details
                Me.txtEmpno.Text = rdr("empno")
                Me.txtEname.Text = rdr("ename")
                Me.txtSal.Text = rdr("sal")
                Me.txtJob.Text = rdr("job")
            Else
                'display message if no rows found
                MessageBox.Show("Not found")
            End If
            'clear up the resources
            rdr.Close()
        Catch ex As Exception
            'display if any error occurs
            MessageBox.Show("Error: " & ex.Message)
            'close the connection if it is still open
        End Try
    End Sub
End Class
```

```
        If cn.State = ConnectionState.Open Then
            cn.Close()
        End If
    End Try
End Sub
End Class
```

Within the above highlighted code, `:empno` is the bind variable. We are placing (or assigning) a value into that bind variable using `OracleParameter`.

If you want to provide a very clear `OracleParameter`, you can even write something like the following code:

```
Dim cmd As New OracleCommand("SELECT empno, ename, _
    sal, deptno FROM emp WHERE ename=:ename", cn)
Dim pEmpno As New OracleParameter
With pEmpno
    .ParameterName = ":ename"
    .OracleDbType = OracleDbType.Varchar2
    .Size = 20
    .Value = Me.txtEname.Text
End With
cmd.Parameters.Add(pEmpno)
```

In the above code fragment, we are working with a bind variable `:ename`, which is of type `VARCHAR2` and size 20. We will deal with `OracleParameter` in more detail in subsequent chapters.

Working with OracleDataAdapter together with OracleCommand

In the previous examples, we worked with `OracleDataAdapter` by directly specifying SQL statements. You can also pass `OracleCommand` to `OracleDataAdapter`. This is very useful if you deal with stored procedures (covered in Chapter 5) or bind variables together with `OracleDataAdapter`.

The following is a simple example that uses `OracleCommand` together with `OracleDataAdapter`:

```
Imports Oracle.DataAccess.Client

Public Class Form10

    Private Sub btnGetEmployees_Click_1(ByVal sender As
```

```

System.Object, ByVal e As System.EventArgs) Handles
btnGetEmployees.Click
    'create connection to db
    Dim cn As New OracleConnection("Data Source=x; _
                                   User Id=scott;Password=tiger")
    Try
        'create command object to work with SELECT
        Dim cmd As New OracleCommand("SELECT empno, _
                                     ename, job, mgr, hiredate, sal, comm, deptno _
                                     FROM emp", cn)
        'create DataAdapter from command
        Dim adp As New OracleDataAdapter(cmd)
        'create the offline data table
        Dim dt As New DataTable
        'fill the data table with data and clear resources
        adp.Fill(dt)
        adp.Dispose()
        'display the data
        Me.DataGridView1.DataSource = dt
    Catch ex As Exception
        'display if any error occurs
        MessageBox.Show("Error: " & ex.Message)
        'close the connection if it is still open
        If cn.State = ConnectionState.Open Then
            cn.Close()
        End If
    End Try
End Sub
End Class

```

You can observe from the above highlighted code that we created an `OracleCommand` object, and the `OracleDataAdapter` can accept `OracleCommand` as a parameter.

Techniques to Improve Performance while Retrieving Data

Performance tuning is a great subject in Oracle. Volumes of books would not be enough to cover every aspect of performance tuning in Oracle. However, in this section, we will only discuss the fundamental performance techniques while working with ODP.NET.

Some of the frequently used techniques to achieve greater performance with ODP.NET are as follows:

- Connection pooling
- Choosing a proper retrieval methodology for every data retrieval task
- Choosing a proper `CommandType` (when using an `OracleCommand` object)
- Controlling the amount of data returned to the client (or middle tier)
- SQL statement caching
- Developing object pooling components (like COM+ etc.)

We have already mentioned **Connection Pooling** earlier in this chapter. Working with a physical database connection for every SQL statement could be very expensive in terms of performance. Try to figure out the best strategy to implement connection pooling in your applications based on factors like heavy data consumption, server resources utilization, frequent access to database, continuous (or long) operations on data, mission-critical scenarios, etc.

As discussed previously, the only way to retrieve data from Oracle in ODP.NET is by using the core `OracleCommand`, `OracleDataReader`, or `OracleDataAdapter`. An application would be made with several simple to complex tasks. Be wise and select the best option between those three, based on every respective task and its complexity. Do not try to take a decision on using only one of them throughout the application, which really kills performance in several scenarios. For example, to retrieve a single value from the database, it is always the best to use `ExecuteScalar` (of the `OracleCommand` object) directly, rather than using the other two.

Never retrieve a whole table unnecessarily. Never use "SELECT *"; always fully qualify an SQL statement. Using "SELECT *" would not only slow down your application performance but also can be a bit dangerous. Imagine a few more new columns are added to the table. All those columns would also be retrieved automatically in the .NET application (whether required or not).

Try to be selective when choosing `CommandType`. It is suggested to use the `StoredProcedure` command type (if you implement stored procedures) or `Text` rather than `TableDirect`. Working with PL/SQL stored procedures is covered in Chapter 5.

Another very common mistake is retrieving too many rows unnecessarily. Imagine a table exists with one million rows and you are trying to retrieve all of them for the user. Any user would never want to view million rows in his or her life time. Not only that, pulling one million of rows from the server really consumes huge memory resources and also makes the network too busy.

In any case, ODP.NET by default fetches only 64K at a time. So, even though you try to execute a `SELECT` statement that retrieves all rows in a table, it retrieves only chunks of 64K based on demand. You can customize this fetch size by issuing the following statement:

```
cmd.FetchSize = cmd.RowSize * 25
```

The above makes sure that it retrieves a maximum of 25 rows per round-trip to the server. You can observe that the `FetchSize` is completely based on `RowSize` and not simply on the number of rows. Apart from modifying the `FetchSize`, try to provide filters in your user interface to minimize the data fetching from server.

If you are working continuously with a similar set of SQL statements (like `INSERT` in a loop etc.) in a routine, it is always suggested to take advantage of statement caching. A cache is nothing but some high-performance memory at server. If you cache the frequently used SQL statements, a copy of such SQL statements gets stored at that high-performance memory and gets executed (with different values) every time you issue the same SQL statement. This removes the burden at the server of parsing and preparing an execution plan for every SQL statement and improves the performance tremendously. Generally, when you use the concept of bind variables together with `OracleParameter`, the statement caching automatically takes place.

Finally, when developing business logic, it is suggested to design scalable business components, which can take advantage of features like automatic object pooling, loosely coupled behavior, caching, persistence, accessibility permissions (security), transactions etc. Designing and implementing business components (like COM+, MSMQ, Windows Services, Web Services, .NET Remoting, etc.) are very common in enterprise applications. Selecting a proper approach for implementing a business component is the main backbone at the middle tier (if you are developing multi-tier applications).

Summary

In this chapter, we have seen several methods to retrieve data from Oracle database. We worked with the core ODP.NET classes like `OracleCommand`, `OracleDataReader`, `OracleDataAdapter`, `OracleParameter`, etc., and the most important ADO.NET classes like `Dataset`, `DataTable`, `DataRow`, etc.

