**Retrieving Data Using a DataReader (ADO.NET)**

Retrieving data using a **DataReader** involves creating an instance of the **Command** object and then creating a **DataReader** by calling **Command.ExecuteReader** to retrieve rows from a data source. The following example illustrates using a **DataReader** where reader represents a valid DataReader and command represents a valid Command object.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl01_code');" \o "Copy Code)

reader = command.ExecuteReader();

You use the **Read** method of the **DataReader** object to obtain a row from the results of the query. You can access each column of the returned row by passing the name or ordinal reference of the column to the **DataReader**. However, for best performance, the **DataReader** provides a series of methods that allow you to access column values in their native data types (**GetDateTime**, **GetDouble**, **GetGuid**, **GetInt32**, and so on). For a list of typed accessor methods for data provider-specific **DataReaders**, see [OleDbDataReader](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdatareader.aspx) and [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx). Using the typed accessor methods, assuming the underlying data type is known, reduces the amount of type conversion required when retrieving the column value.

|  |
| --- |
| **NoteNote** |
| The Windows Server 2003 release of the .NET Framework includes an additional property for the **DataReader**, **HasRows**, which enables you to determine if the **DataReader** has returned any results before reading from it. |

The following code example iterates through a **DataReader** object, and returns two columns from each row.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl05_code');" \o "Copy Code)

Private Sub HasRows(ByVal connection As SqlConnection)

Using connection

Dim command As SqlCommand = New SqlCommand( \_

"SELECT CategoryID, CategoryName FROM Categories;", \_

connection)

connection.Open()

Dim reader As SqlDataReader = command.ExecuteReader()

If reader.HasRows Then

Do While reader.Read()

Console.WriteLine(reader.GetInt32(0) \_

& vbTab & reader.GetString(1))

Loop

Else

Console.WriteLine("No rows found.")

End If

reader.Close()

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl06_code');" \o "Copy Code)

static void HasRows(SqlConnection connection)

{

using (connection)

{

SqlCommand command = new SqlCommand(

"SELECT CategoryID, CategoryName FROM Categories;",

connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

Console.WriteLine("{0}\t{1}", reader.GetInt32(0),

reader.GetString(1));

}

}

else

{

Console.WriteLine("No rows found.");

}

reader.Close();

}

}

The **DataReader** provides an unbuffered stream of data that allows procedural logic to efficiently process results from a data source sequentially. The **DataReader** is a good choice when retrieving large amounts of data because the data is not cached in memory.

http://i.msdn.microsoft.com/Global/Images/clear.gif Closing the DataReader

You should always call the **Close** method when you have finished using the **DataReader** object.

If your **Command** contains output parameters or return values, they will not be available until the **DataReader** is closed.

Note that while a **DataReader** is open, the **Connection** is in use exclusively by that **DataReader**. You cannot execute any commands for the **Connection**, including creating another **DataReader**, until the original **DataReader** is closed.

|  |
| --- |
| **NoteNote** |
| Do not call **Close** or **Dispose** on a **Connection**, a **DataReader**, or any other managed object in the **Finalize** method of your class. In a finalizer, only release unmanaged resources that your class owns directly. If your class does not own any unmanaged resources, do not include a **Finalize** method in your class definition. For more information, see [Garbage Collection](http://msdn.microsoft.com/en-us/library/0xy59wtx.aspx). |

http://i.msdn.microsoft.com/Global/Images/clear.gif Retrieving Multiple Result Sets using NextResult

If multiple result sets are returned, the **DataReader** provides the **NextResult** method to iterate through the result sets in order. The following example shows the [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) processing the results of two SELECT statements using the [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executereader.aspx) method.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl51_ctl00_ctl02_code');" \o "Copy Code)

Private Sub RetrieveMultipleResults(ByVal connection As SqlConnection)

Using connection

Dim command As SqlCommand = New SqlCommand( \_

"SELECT CategoryID, CategoryName FROM Categories;" & \_

"SELECT EmployeeID, LastName FROM Employees", connection)

connection.Open()

Dim reader As SqlDataReader = command.ExecuteReader()

Do While reader.HasRows

Console.WriteLine(vbTab & reader.GetName(0) \_

& vbTab & reader.GetName(1))

Do While reader.Read()

Console.WriteLine(vbTab & reader.GetInt32(0) \_

& vbTab & reader.GetString(1))

Loop

reader.NextResult()

Loop

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl51_ctl00_ctl03_code');" \o "Copy Code)

static void RetrieveMultipleResults(SqlConnection connection)

{

using (connection)

{

SqlCommand command = new SqlCommand(

"SELECT CategoryID, CategoryName FROM dbo.Categories;" +

"SELECT EmployeeID, LastName FROM dbo.Employees",

connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.HasRows)

{

Console.WriteLine("\t{0}\t{1}", reader.GetName(0),

reader.GetName(1));

while (reader.Read())

{

Console.WriteLine("\t{0}\t{1}", reader.GetInt32(0),

reader.GetString(1));

}

reader.NextResult();

}

}

}

http://i.msdn.microsoft.com/Global/Images/clear.gif Getting Schema Information from the DataReader

While a **DataReader** is open, you can retrieve schema information about the current result set using the **GetSchemaTable** method. **GetSchemaTable** returns a [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) object populated with rows and columns that contain the schema information for the current result set. The **DataTable** contains one row for each column of the result set. Each column of the schema table row maps to a property of the column returned in the result set, where the **ColumnName** is the name of the property and the value of the column is the value of the property. The following code example writes out the schema information for **DataReader**.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl01_code');" \o "Copy Code)

Private Sub GetSchemaInfo(ByVal connection As SqlConnection)

Using connection

Dim command As SqlCommand = New SqlCommand( \_

"SELECT CategoryID, CategoryName FROM Categories;", \_

connection)

connection.Open()

Dim reader As SqlDataReader = command.ExecuteReader()

Dim schemaTable As DataTable = reader.GetSchemaTable()

Dim row As DataRow

Dim column As DataColumn

For Each row In schemaTable.Rows

For Each column In schemaTable.Columns

Console.WriteLine(String.Format("{0} = {1}", \_

column.ColumnName, row(column)))

Next

Console.WriteLine()

Next

reader.Close()

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl02_code');" \o "Copy Code)

static void GetSchemaInfo(SqlConnection connection)

{

using (connection)

{

SqlCommand command = new SqlCommand(

"SELECT CategoryID, CategoryName FROM Categories;",

connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

DataTable schemaTable = reader.GetSchemaTable();

foreach (DataRow row in schemaTable.Rows)

{

foreach (DataColumn column in schemaTable.Columns)

{

Console.WriteLine(String.Format("{0} = {1}",

column.ColumnName, row[column]));

}

}

}

}

http://i.msdn.microsoft.com/Global/Images/clear.gif Working with OLE DB Chapters

Hierarchical rowsets, or chapters (OLE DB type **DBTYPE\_HCHAPTER**, ADO type **adChapter**) can be retrieved using the [OleDbDataReader](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdatareader.aspx). When a query that includes a chapter is returned as a **DataReader**, the chapter is returned as a column in that **DataReader** and is exposed as a **DataReader** object.

The ADO.NET **DataSet** can also be used to represent hierarchical rowsets using parent-child relationships between tables. For more information, see [DataSets, DataTables, and DataViews (ADO.NET)](http://msdn.microsoft.com/en-us/library/ss7fbaez.aspx).

The following code example uses the MSDataShape Provider to generate a chapter column of orders for each customer in a list of customers.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl02_code');" \o "Copy Code)

Using connection As OleDbConnection = New OleDbConnection( \_

"Provider=MSDataShape;Data Provider=SQLOLEDB;" & \_

"Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind")

Dim custCMD As OleDbCommand = New OleDbCommand( \_

"SHAPE {SELECT CustomerID, CompanyName FROM Customers} " & \_

"APPEND ({SELECT CustomerID, OrderID FROM Orders} AS CustomerOrders " & \_

"RELATE CustomerID TO CustomerID)", connection)

connection.Open()

Dim custReader As OleDbDataReader = custCMD.ExecuteReader()

Dim orderReader As OleDbDataReader

Do While custReader.Read()

Console.WriteLine("Orders for " & custReader.GetString(1))

' custReader.GetString(1) = CompanyName

orderReader = custReader.GetValue(2)

' custReader.GetValue(2) = Orders chapter as DataReader

Do While orderReader.Read()

Console.WriteLine(vbTab & orderReader.GetInt32(1))

' orderReader.GetInt32(1) = OrderID

Loop

orderReader.Close()

Loop

' Make sure to always close readers and connections.

custReader.Close()

End Using

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl03_code');" \o "Copy Code)

Using (OleDbConnection connection = new OleDbConnection(

"Provider=MSDataShape;Data Provider=SQLOLEDB;" +

"Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind"));

{

OleDbCommand custCMD = new OleDbCommand(

"SHAPE {SELECT CustomerID, CompanyName FROM Customers} " +

"APPEND ({SELECT CustomerID, OrderID FROM Orders} AS CustomerOrders " +

"RELATE CustomerID TO CustomerID)", connection);

connection.Open();

OleDbDataReader custReader = custCMD.ExecuteReader();

OleDbDataReader orderReader;

while (custReader.Read())

{

Console.WriteLine("Orders for " + custReader.GetString(1));

// custReader.GetString(1) = CompanyName

orderReader = (OleDbDataReader)custReader.GetValue(2);

// custReader.GetValue(2) = Orders chapter as DataReader

while (orderReader.Read())

Console.WriteLine("\t" + orderReader.GetInt32(1));

// orderReader.GetInt32(1) = OrderID

orderReader.Close();

}

// Make sure to always close readers and connections.

custReader.Close();

}

http://i.msdn.microsoft.com/Global/Images/clear.gif Returning Results with Oracle REF CURSORs

The .NET Framework Data Provider for Oracle supports the use of Oracle REF CURSORs to return a query result. An Oracle REF CURSOR is returned as an [OracleDataReader](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracledatareader.aspx).

You can retrieve an **OracleDataReader** object, that represents an Oracle REF CURSOR using the [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oraclecommand.executereader.aspx) method, and you can also specify an [OracleCommand](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oraclecommand.aspx) that returns one or more Oracle REF CURSORs as the **SelectCommand** for an [OracleDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracledataadapter.aspx) used to fill a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx).

To access a REF CURSOR returned from an Oracle data source, create an **OracleCommand** for your query and add an output parameter that references the REF CURSOR to the **Parameters** collection of your **OracleCommand**. The name of the parameter must match the name of the REF CURSOR parameter in your query. Set the type of the parameter to **OracleType.Cursor**. The **ExecuteReader** method of your **OracleCommand** will return an **OracleDataReader** for the REF CURSOR.

If your **OracleCommand** returns multiple REF CURSORS, add multiple output parameters. You can access the different REF CURSORs by calling the **OracleCommand.ExecuteReader** method. The call to **ExecuteReader** returns an **OracleDataReader** referencing the first REF CURSOR. You can then call the **OracleDataReader.NextResult** method to access subsequent REF CURSORs. Although the parameters in your **OracleCommand.Parameters** collection match the REF CURSOR output parameters by name, the **OracleDataReader** accesses them in the order that they were added to the **Parameters** collection.

For example, consider the following Oracle package and package body.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl05_code');" \o "Copy Code)

CREATE OR REPLACE PACKAGE CURSPKG AS

TYPE T\_CURSOR IS REF CURSOR;

PROCEDURE OPEN\_TWO\_CURSORS (EMPCURSOR OUT T\_CURSOR,

DEPTCURSOR OUT T\_CURSOR);

END CURSPKG;

CREATE OR REPLACE PACKAGE BODY CURSPKG AS

PROCEDURE OPEN\_TWO\_CURSORS (EMPCURSOR OUT T\_CURSOR,

DEPTCURSOR OUT T\_CURSOR)

IS

BEGIN

OPEN EMPCURSOR FOR SELECT \* FROM DEMO.EMPLOYEE;

OPEN DEPTCURSOR FOR SELECT \* FROM DEMO.DEPARTMENT;

END OPEN\_TWO\_CURSORS;

END CURSPKG;

The following code creates an **OracleCommand** that returns the REF CURSORs from the previous Oracle package by adding two parameters of type **OracleType.Cursor** to the **Parameters** collection.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl06_code');" \o "Copy Code)

Dim cursCmd As OracleCommand = New OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn)

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl07_code');" \o "Copy Code)

OracleCommand cursCmd = new OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn);

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

The following code returns the results of the previous command using the **Read** and **NextResult** methods of the **OracleDataReader**. The REF CURSOR parameters are returned in order.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl08_code');" \o "Copy Code)

oraConn.Open()

Dim cursCmd As OracleCommand = New OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn)

cursCmd.CommandType = CommandType.StoredProcedure

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

Dim reader As OracleDataReader = cursCmd.ExecuteReader()

Console.WriteLine(vbCrLf & "Emp ID" & vbTab & "Name")

Do While reader.Read()

Console.WriteLine("{0}" & vbTab & "{1}, {2}", reader.GetOracleNumber(0), reader.GetString(1), reader.GetString(2))

Loop

reader.NextResult()

Console.WriteLine(vbCrLf & "Dept ID" & vbTab & "Name")

Do While reader.Read()

Console.WriteLine("{0}" & vbTab & "{1}", reader.GetOracleNumber(0), reader.GetString(1))

Loop

' Make sure to always close readers and connections.

reader.Close()

oraConn.Close()

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl09_code');" \o "Copy Code)

oraConn.Open();

OracleCommand cursCmd = new OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn);

cursCmd.CommandType = CommandType.StoredProcedure;

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

OracleDataReader reader = cursCmd.ExecuteReader();

Console.WriteLine("\nEmp ID\tName");

while (reader.Read())

Console.WriteLine("{0}\t{1}, {2}", reader.GetOracleNumber(0), reader.GetString(1), reader.GetString(2));

reader.NextResult();

Console.WriteLine("\nDept ID\tName");

while (reader.Read())

Console.WriteLine("{0}\t{1}", reader.GetOracleNumber(0), reader.GetString(1));

// Make sure to always close readers and connections.

reader.Close();

oraConn.Close();

The following example uses the previous command to populate a **DataSet** with the results of the Oracle package.

|  |
| --- |
| **NoteNote** |
| To avoid an **OverflowException**, we recommend that you also handle any conversion from the Oracle NUMBER type to a valid .NET Framework type before storing the value in a **DataRow**. You can use the **FillError** event to determine if an **OverflowException** has occurred. For more information on the **FillError** event, see [Handling DataAdapter Events (ADO.NET)](http://msdn.microsoft.com/en-us/library/6d1wk41s.aspx). |

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl12_code');" \o "Copy Code)

Dim ds As DataSet = New DataSet()

Dim adapter As OracleDataAdapter = New OracleDataAdapter(cursCmd)

adapter.TableMappings.Add("Table", "Employees")

adapter.TableMappings.Add("Table1", "Departments")

adapter.Fill(ds)

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl13_code');" \o "Copy Code)

DataSet ds = new DataSet();

OracleDataAdapter adapter = new OracleDataAdapter(cursCmd);

adapter.TableMappings.Add("Table", "Employees");

adapter.TableMappings.Add("Table1", "Departments");

adapter.Fill(ds);