Cognizant | Academy

Business Focused. Learner Centric.

ADO.NET

Enabling Objectives

After completing this chapter, you will be able to explain about the basics of ADO.Net

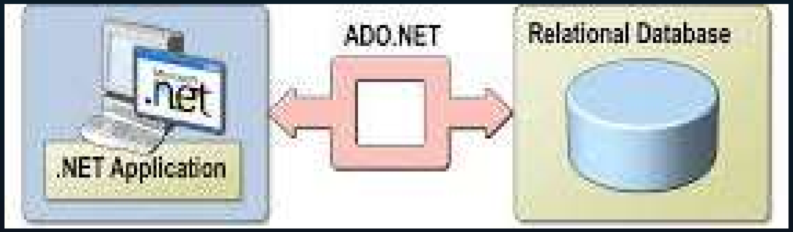
Key Topics

- Connected & Disconnected architecture
- Querying database
- Data binding
- Managing Local Transactions
- Usage of LINQ
- Integration with ASP.Net

What is ADO.Net and benefits of ADO.Net

ADO .Net : Overview

- ADO.Net is a set of classes that comes with the Microsoft .NET framework to facilitate data access from managed languages.
- ADO.Net has been in existence for a long time and it provides a comprehensive and complete set of libraries for data access.



Evolution of ADO.NET

- The first data access model, DAO (Data Access Model) was created for local databases with the built in Jet engine which had performance and functionality issues
- Next came RDO(Remote Data Object) and ADO (Active Data Object) which were designed for Client server Architectures but, soon ADO took over RDO.
- With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls.

Evolution of ADO.NET (Contd.)

- ADO was a connected data access, when a connection to the database is established the connection remains open until the application is closed.
- Leaving the connection open for the lifetime of the application raises concerns about database security and network traffic .
- Also, as database are becoming increasingly important and as they are serving more people, a connected data access model makes us think about its productivity.

Why ADO.Net?

- ADO.NET is ADO extended to work in cooperation with any software component , on any platform, that understands XML.
- The ADO.Net interface is readily accessible using any of the .NET complaint languages.
- Following are some of the Design Goal of ADO.NET
    - Scalability
    - Reliability
    - Performance
    - Statelessness
    - Robustness
    - Disconnected view of data (Dataset)
    - XML

Connected & Disconnected architecture

Connected Architecture in ADO.NET

- In the Connection Oriented Data Access Architecture the application makes a connection to the Data Source and then interact with it through SQL requests using the same connection.
- In these cases the application stays connected to the database system even when it is not using any Database Operation.

Connected Environment in ADO.NET

- Working with data directly via Open connection.
- Advantages:
    - Simple security realization
    - Work with real data
    - Simple Organization of distributed work
- Drawbacks:
    - Continual Connection
    - Not available Via Internet

Components of SQL connection

Creating a SqlConnection Object

- The SqlConnection object uses a constructor with a single argument of type string.
- This argument is called a connection string.

SqlConnection conn = new SqlConnection("Data Source=(local);Initial Catalog=Northwind; Integrated Security=SSPI");

Importance of ‘Using’ keyword

The ‘using’ keyword while acquiring SqlConnection helps to manage SqlConnection objects that are not managed by .Net runtime.

```
using System;
using System.Data.SqlClient;

class Program
{
    static void Main()
    {
        string connectionString =
            "Data Source=(local);Initial Catalog=Northwind; Integrated Security=SSPI";
        //
        // In a using statement, acquire the SqlConnection as a resource.
        //
        using (SqlConnection con = new SqlConnection(connectionString))
        {
            //
            // Open the SqlConnection.
            //
            con.Open();
             // Perform the required operation.
             // There is no need to explicitly close the connection on implementing "using"
         }
    }
}
```

Reference link: https://www.dotnetperls.com/sqlconnection

The SqlCommand Object

- A SqlCommand object allows you to specify what type of interaction you want to perform with a database.
- For example, you can do select, insert, modify, and delete commands on rows of data in a database table.
- The SqlCommand object can be used to support disconnected data management scenarios

Creating a SqlCommand Object

The SqlCommand class is at the heart of the System.SqlClient namespace.

It is used to execute operations on a database and retrieve data.

SqlCommand object takes a string parameter that holds the command you want to execute and a reference to a SqlConnection object

Eg : SqlCommand cmd = new SqlCommand("select CategoryName from Categories", conn);

SqlCommand Properties

Item	Description
CommandText	Contains the text of a SQL query
CommandTimeout	Contains the length of the timeout of a query, in seconds
CommandType	Specifies the type of command to be executed
Connection	Specifies the connection to the database
Parameters	Specifies a collection of parameters for the SQL query
Transaction	Specifies a transaction object, which enables developers to run queries in a transaction

Preparing a SqlCommand Object for Parameters

- The first step in using parameters in SQL queries is to build a command string containing parameter placeholders.

- These placeholders are filled in with actual parameter values when the SqlCommand executes. Proper syntax of a parameter is to use an '@' symbol prefix on the parameter name as shown below:

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "DELETE FROM Empls WHERE
EmployeeID = @ID";
```

Preparing a SqlCommand Object for Parameters

// 1. declare command object with parameter

SqlCommand cmd = new SqlCommand("select * from Customers where city = @City", conn);

- In the SqlCommand constructor above, the first argument contains a parameter declaration, *@City*.
- This example used one parameter, but you can have as many parameters as needed to customize the query.
- Each parameter will match a SqlParameter object that must be assigned to this SqlCommand object.

Declaring a SqlParameter Object

- Each parameter in a SQL statement must be defined.

// 2. define parameters used in command object

SqlParameter param = new SqlParameter(); param.ParameterName = "@City";

param.Value = inputCity;

- Notice that the ParameterName property of the SqlParameter instance must be spelled exactly as the parameter that is used in the SqlCommand SQL command string.

- You must also specify a value for the command. When the SqlCommand object executes, the parameter will be replaced with this value.

Associate a SqlParameter Object with a SqlCommand Object

- For each parameter defined in the SQL command string argument to a SqlCommand object, you must define a SqlParameter.
- // 3. add new parameter to command object
  cmd.Parameters.Add("@City",SQLDBType.String);

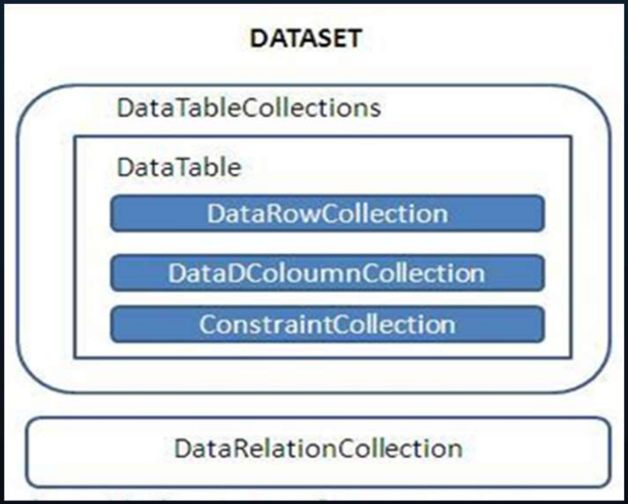
  cmd.Parameters["@City"].Value="Boston";

- The SqlParameter instance is the argument to the Add method of the Parameters property for the SqlCommand object above.

- You must add a unique SqlParameter for each parameter defined in the SqlCommand object's SQL command string.

ADO.NET DataSets

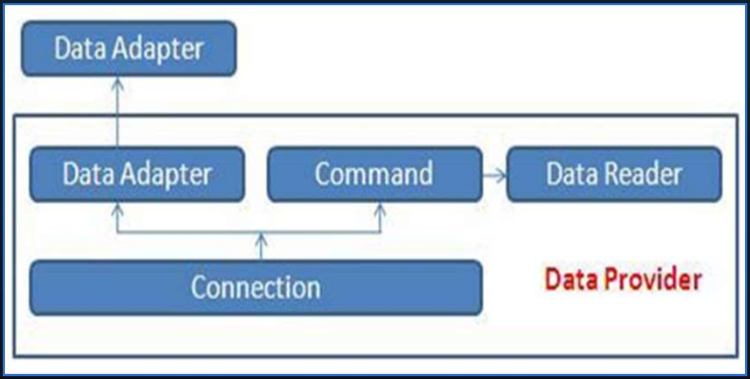
- The DataSet is a memory-resident representation of data that provides a consistent relational programming model regardless of the data source.
- The DataSet object supports disconnected, distributed data scenarios with ADO.NET.
- The DataSet represents a complete set of data, including related tables, constraints, and relationships among the tables.



The DataSet can also persist and reload its contents as XML, and its schema as XML schema definition language (XSD) schema.

ADO.NET Data Providers

- The two key components of ADO.NET are Data Providers and DataSet
- The .Net Framework includes mainly three Data Providers for ADO.NET. They are the following:
- Microsoft SQL Server Data Provider
- OLEDB Data Provider
- ODBC Data Provider



Components of data provider include:

- ➢ Connection
- ➢ Command
- ➢ DataReader
- ➢ DataAdapter

ADO.NET- .NET Data Providers

The following table lists the data providers that are included in the .NET Framework.

.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	Provides data access for Microsoft SQL Server.
.NET Framework Data Provider for OLE DB	For data sources exposed by using OLE DB.
.NET Framework Data Provider for ODBC	For data sources exposed by using ODBC.
.NET Framework Data Provider for Oracle	For Oracle data sources.
EntityClient Provider	Provides data access for Entity Data Model (EDM) applications.
.NET Framework Data Provider for SQL Server Compact 4.0.	Provides data access for Microsoft SQL Server Compact 4.0.

Components for querying

SqlCommand. ExecuteNonQuery()

Syntax

sqlCommand.ExecuteNonQuery()

Description

The ExecuteNonQuery() method executes the command text against the database specified in the Connection object.

This method is optimized for queries that do not return any information

(for example, DELETE and UPDATE queries).

Example

```
SqlCommand cmd = new SqlCommand("DELETE FROM Customers WHERE
LastName='Jones'", conn);
cmd.ExecuteNonQuery();
```

SqlCommand.ExecuteReader()

Syntax

```
SqlDataReader ExecuteReader()
```

Description

The ExecuteReader() method executes the command text against the database specified in the Connection object and returns a SqlDataReader object with the results of the query.

Example

```
SqlCommand cmd = new SqlCommand("SELECT Name, City FROM
Customers", conn);
SqlDataReader reader = cmd.ExecuteReader();
```

SqlCommand.ExecuteScalar()

Synatx

Object ExecuteScalar()

Description

The ExecuteScalar() method executes the command text against the database specified in the Connection object and returns a single object.

The ExecuteScalar() method exists because it is wasteful to return a dataset for a single value.

The overhead for the dataset would be much larger than the actual value being returned.

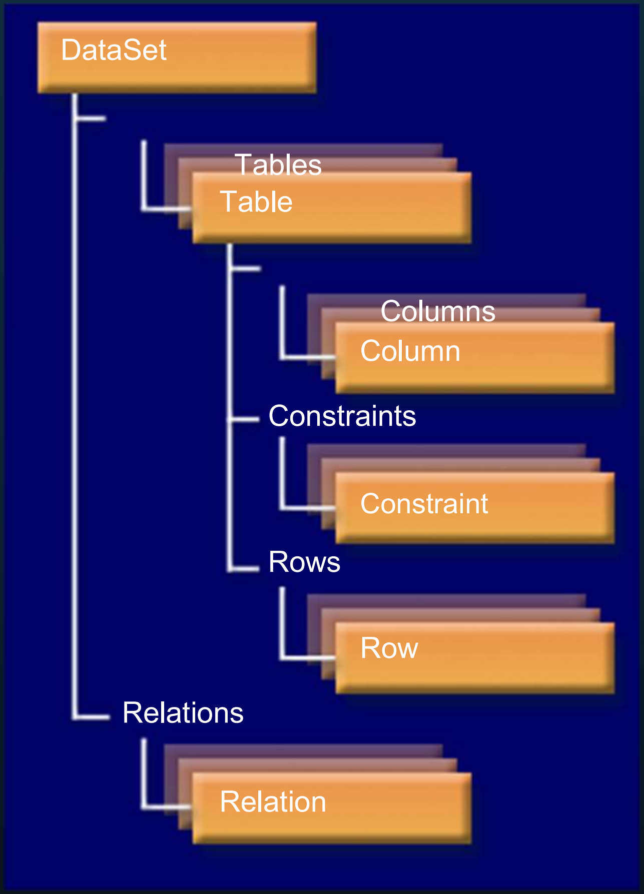
Example

```
SqlCommand cmd = new SqlCommand("SELECT count(*) FROM
Customers", conn);
Int customerCount = (Int)cmd.ExecuteScalar();
msg.Text = "There are "+customerCount.ToString()+" customers in the
database.";
```

Querying database

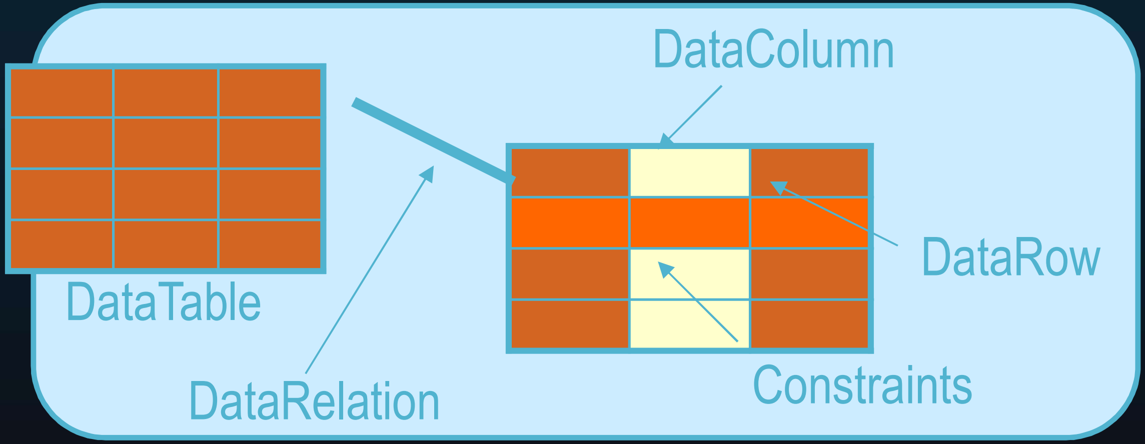
- A Dataset is an in-memory data store that can hold numerous tables.
- Datasets only hold data and do not interact with a data source.

Eg : DataSet dsCustomers = new DataSet();



The DataSet Object Model

- Common collections
  - Tables (collection of DataTable objects)
  - Relations (collection of DataRelation objects)



- It is the SqlDataAdapter that manages connections with the data source and gives us disconnected behavior.

- The SqlDataAdapter opens a connection only when required and closes it as soon as it has performed its task.

Eg : SqlDataAdapter daCustomers = new SqlDataAdapter("select CustomerID, Company Name from Customers", conn);

- For example, the SqlDataAdapter performs the following tasks when filling a Dataset with data
- Open connection
- Retrieve data into Dataset (***Fill***)
- Close connection

Filling the DataSet

Syntax:

Dataadapter.fill(dataset,”datatablename”);

Eg :

daCustomers.Fill(dsCustomers,"Customers");

Recall

- Recall the importance of 'Using' statement in opening and closing SQL connection

Data binding

SqlCommand. ExecuteNonQuery()

Syntax

```
sqlCommand.ExecuteNonQuery()
```

Description

The ExecuteNonQuery() method executes the command text against the database specified in the Connection object.

This method is optimized for queries that do not return any information (for example, DELETE and UPDATE queries).

Example

```
SqlCommand cmd = new SqlCommand("DELETE FROM Customers WHERE
LastName='Jones'", conn);
cmd.ExecuteNonQuery();
```

Managing Local Transactions

TransactionScope

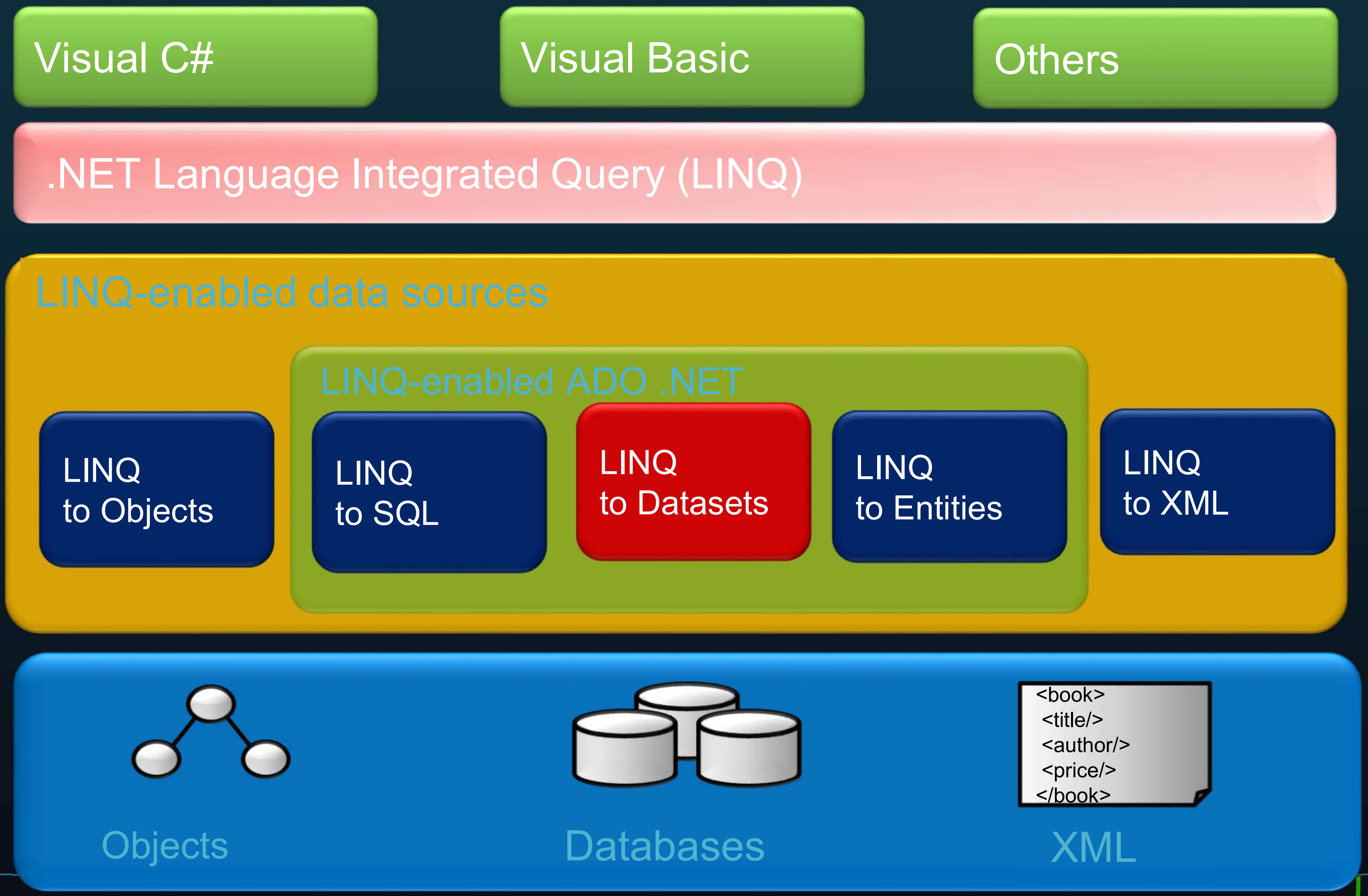
- Feature of ADO.Net to ensure transactional operations complete successfully
- On error occurrence the data operation is rolled back so that the previous stable state of the data and database is restored

- TransactionScope of System.Transactions
- TransactionScope complete commits a successful transaction
- TransactionScope enclosed within Using statement ensures that the rollback happens on occurrence of error

- Reference link:
- https://docs.microsoft.com/en-us/dotnet/framework/data/transactions/implementing-an-implicit-transaction-using-transaction-scope

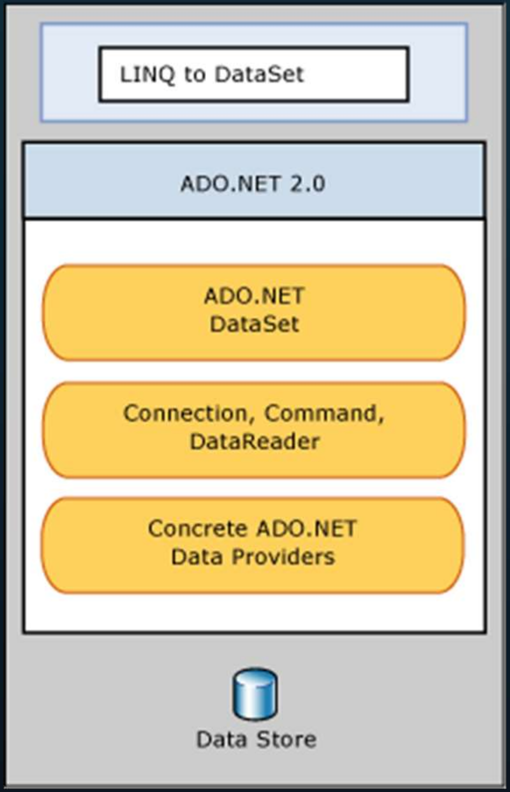
Usage of LINQ

LINQ Architecture



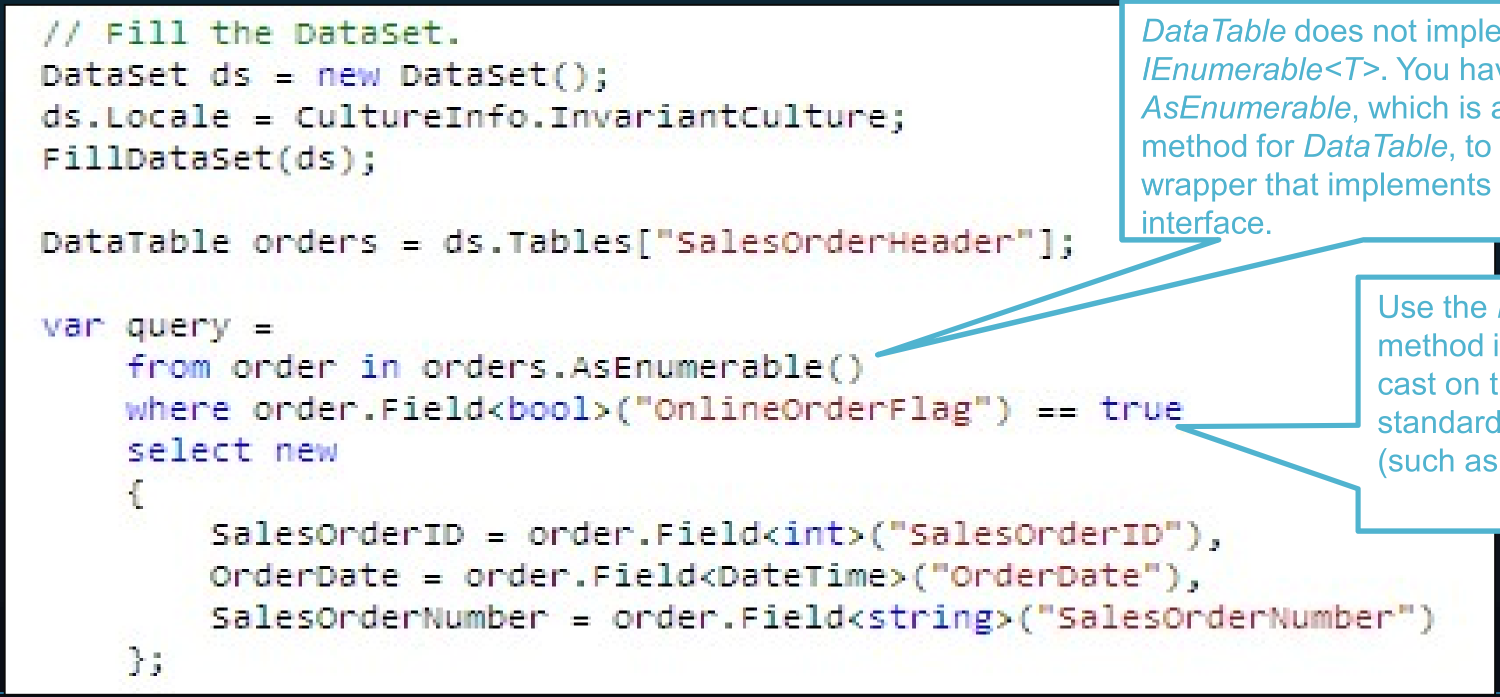
LINQ to DataSet

- LINQ to DataSet makes it easier and faster to query over data cached in a DataSet object.
- LINQ to DataSet simplifies querying by enabling developers to write queries from the programming language itself, instead of by using a separate query language.


LINQ to DataSet
ADO.NET 2.0
ADO.NET DataSet
Connection, Command, DataReader
Concrete ADO.NET Data Providers
Data Store

LINQ to DataSet – Querying DataSets

- Formulating queries with LINQ to DataSet is similar to using LINQ against other LINQ-enabled data sources.
- When using LINQ queries over a DataSet object, the developers are actually querying an enumeration of DataRow objects, instead of an enumeration of a custom type.



```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable orders = ds.Tables["SalesOrderHeader"];

var query =
    from order in orders.AsEnumerable()
    where order.Field<bool>("OnlineOrderFlag") == true
    select new
    {
        SalesOrderID = order.Field<int>("SalesOrderID"),
        OrderDate = order.Field<DateTime>("OrderDate"),
        SalesOrderNumber = order.Field<string>("SalesOrderNumber")
    };
```

DataTable does not implement *IEnumerable<T>*. You have to call *AsEnumerable*, which is an extension method for *DataTable*, to obtain a wrapper that implements that interface.

Use the *Field<T>* accessor method instead of using a direct cast on the result of the standard *DataRow* accessor (such as *o["OnlineOrderFlag"]*).

Integration with ASP.Net

Integration requirement

- Create a Class Library project
- Create a Business entity class
- Create class with a method to fetch data from database using ExecuteDataSet
- Fill the data in the business entity class
- Return the data
- Reference ADO.Net project in ASP.Net application
- Invoke the ADO.Net project class method to fetch the data returned by it
- Display the data returned by ADO.Net project on Gridview

Test Your Understanding

- Practice Check
- Final Check

Recap

In this chapter, we have learnt about:

- Connected & Disconnected architecture
- Querying database
- Data binding
- Managing Local Transactions
- Usage of LINQ
- Integration with ASP.Net

ADO.Net
You have successfully completed –

Learning on basics of ADO.Net