Module 06 Notes

LINQ (Language-Integrated Queries)

Traditionally, data access code has been dependent on the type of data storage. This means that code used to manage text file data was very different from code that managed database data. To make data access code more general, Microsoft created an abstraction layer over the various data access technologies and called it LINQ.

In this module we will demonstrate how you can use LINQ to manage data.

## LINQ vs ADO.NET

“There are three separate ADO.NET Language-Integrated Query (LINQ) technologies: LINQ to DataSet, LINQ to SQL, and LINQ to Entities. LINQ to DataSet provides richer, optimized querying over the [DataSet](https://msdn.microsoft.com/en-us/library/system.data.dataset(v=vs.100).aspx) and LINQ to SQL enables you to directly query SQL Server database schemas, and LINQ to Entities allows you to query an Entity Data Model.

The following diagram provides an overview of how the ADO.NET LINQ technologies relate to high-level programming languages and LINQ-enabled data sources.” (<https://msdn.microsoft.com/en-us/library/bb399365(v=vs.100).aspx> )

# LINQ to ADO.NET overview

# LINQ

“**Language-Integrated Query (LINQ) is** a set of features introduced in Visual Studio 2008 that extends powerful query capabilities to the language syntax of C# and Visual Basic. LINQ introduces **standard**, easily-learned patterns **for querying and updating data**, and the technology can be extended to support potentially any kind of data store. Visual Studio includes LINQ provider assemblies that enable the use of LINQ with .NET Framework **collections**, SQL Server **databases**, ADO.NET **Datasets**, and **XML** **documents**.” (<https://msdn.microsoft.com/en-us/library/bb397926.aspx>)

## LINQ to Generic Types

**“**LINQ queries are based on generic types, which were introduced in version 2.0 of the .NET Framework. You do not need an in-depth knowledge of generics before you can start writing queries. However, you may want to understand two basic concepts:

1. When you **create an instance of a generic collection** class such as [List<T>](https://msdn.microsoft.com/en-us/library/6sh2ey19.aspx), you **replace the "T" with the type of objects that the list will hold**. For example, a list of strings is expressed as List<string>, and a list of Customer objects is expressed as List<Customer>. A generic list is strongly typed and provides many benefits over collections that store their elements as [Object](https://msdn.microsoft.com/en-us/library/system.object.aspx). If you try to add a Customer to a List<string>, you will get an error at compile time. It is easy to use generic collections because you do not have to perform run-time type-casting.
2. [IEnumerable<T>](https://msdn.microsoft.com/en-us/library/9eekhta0.aspx) is the interface that enables generic collection classes to be **enumerated by using the foreach statement**. Generic collection classes support [IEnumerable<T>](https://msdn.microsoft.com/en-us/library/9eekhta0.aspx) just as non-generic collection classes such as [ArrayList](https://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx) support [IEnumerable](https://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx). “

(<https://msdn.microsoft.com/en-us/library/bb546142.aspx>)

#### LINQ to Arrays

private void buttonLINQAndArrays\_Click(object sender, EventArgs e)

{

Console.WriteLine("\n\*\*\* LINQ with an Array \*\*\*");

// create an integer array

int[] **intValues** = { 4, 8, 5, 2, 9, 0, 3, 7, 1, 6 };

// Traditional approach has you specify how to process data from the array

Console.Write("\n Traditional approach: ");

int intTrdFilteredData = 0;

foreach (int element in **intValues**)

{

if (element == (4 + 2))

{

intTrdFilteredData = element;

}

}

Console.Write(intTrdFilteredData);

// LINQ approach, lets .Net create the filter code for you

Console.Write("\n LINQ approach: ");

var intLinqFiltered = // This returns an IEnumerable compatible object!

from value in intValues

where value == (4 + 2)

select value;

foreach (var element in intLinqFiltered)

{

Console.Write(" {0}", element);

}

/\*\*\* Additional Examples of that would take much more code to write without LINQ \*\*\*/

Console.Write("\n LINQ approach with ORDERBY: ");

var sorted =

from value in intValues

orderby value //Ordering results

select value;

foreach (var element in sorted)

Console.Write(" {0}", element);

Console.Write("\n LINQ approach with WHERE and ORDERBY: ");

var sortAndFilter =

from value in intValues

where value < 4

orderby value descending

select value;

foreach (var element in sortAndFilter)

Console.Write(" {0}", element);

Console.WriteLine("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

## LINQ to Objects

“In a basic sense, LINQ to Objects represents a new approach to collections. In the **old way**, you had to write **complex foreach loops that specified how to retrieve data from a collection**. In the LINQ approach, you write declarative code that describes what you want to retrieve.

In addition, **LINQ queries offer three main advantages** over traditional **foreach** loops:

1. They are **more concise and readable**, especially when filtering multiple conditions.
2. They provide **powerful filtering, ordering, and grouping** capabilities with a minimum of application code.
3. They **can be ported to other data sources with little or no modification**.

In general, the more complex the operation you want to perform on the data, the more benefit you will realize by using LINQ instead of traditional iteration techniques.” ( <https://msdn.microsoft.com/en-us/library/bb397919.aspx> )

#### LINQ to Objects

/\* Make a class for the demo (See Customer.cs File)

\*

class Customer

{

private int intID;

private string strName;

public int ID

{

get { return intID; }

set { intID = value; }

}

public string Name

{

get { return strName; }

set { strName = value; }

}

public Customer(int ID, string Name) { this.ID = ID; this.Name = Name; }

public override string ToString() { return ID.ToString() + "," + Name; }

}

\*

\*/

//Use LINQ to work with objects made from the Customer class

private void buttonLINQAndObjects\_Click(object sender, EventArgs e)

{

Console.WriteLine("\n \*\*\* LINQ with Objects \*\*\*");

// Make an array of customers

Customer[] arrCustomers = { new Customer(1, "Bob"), new Customer(2, "Sue"), new Customer(3, "Tim") };

// Traditional approach has you specify how to process data from the array

Console.Write("\n Traditional approach: ");

foreach (var element in arrCustomers)

{ Console.Write(element.ToString() + " "); }

// LINQ approach

var objCust =

from c in arrCustomers

select c;

Console.Write("\n LINQ approach: ");

foreach (var element in objCust) { Console.Write(element + " "); }

/\*\*\* Additional Examples of that would take much more code to write \*\*\*/

// filter in a LINQ query

var objOnlySue =

from c in arrCustomers

where c.Name == "Sue"

select c;

Console.Write("\n LINQ approach Filtered Data: ");

foreach (var element in objOnlySue) { Console.Write(element); }

// order the customer by name with LINQ

var objNamesSorted =

from c in arrCustomers

orderby c.Name descending

select c;

Console.Write("\n LINQ approach Sorted Data: ");

foreach (var element in objNamesSorted) { Console.Write(element.ToString() + " "); }

//This example uses LINQ to select, or "Project", first and last names into an ANNOYMOUS type

// (Something that we see a lot in more advanced examples!)

var objIDsNamesAndMore =

from c in arrCustomers //The next line of code creates an ANNOYMOUS type, using properties you dictate.

select new { c.ID, ANameProperty = c.Name, StaticData = "a", DynamicData = DateTime.Now };

// Display ANNOYMOUS type data

Console.WriteLine("\n ANNOYMOUS type data:");

foreach (var element in objIDsNamesAndMore) { Console.Write(element.ToString() + "\n"); }

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

# **Demo**: Using LINQ to Arrays and Objects

Your instructor will now demo how to use LINQ with a simple integer array and an array of custom objects.

# Lab 06-1

In this lab, you will create an array of Product objects and use LINQ to display their data.

1. *Create a new Windows Form Application project in Visual Studio called Module06Labs.*
2. Create a new class called Product and another called Product**s** using the following code:

class Product

{

public int ID {get;set;}

public string Name{get;set;}

public float Price {get;set;}

public Product(int ID, string Name, float Price) { this.ID = ID; this.Name = Name; this.Price = Price; }

public override string ToString() { return ID.ToString() + "," + Name + "," + Price; }

}

static class Products

{

public static List<Product> ProductsList = new List<Product>();

static Products()

{

ProductsList.Add(new Product(1, "ProdA", 9.99F));

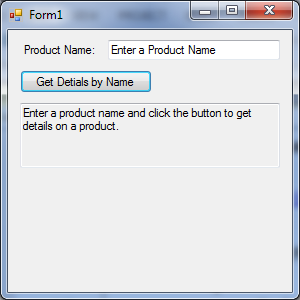
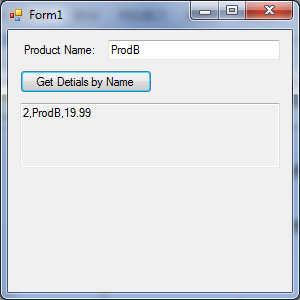
ProductsList.Add(new Product(2, "ProdB", 19.99F));

ProductsList.Add(new Product(3, "ProdC", 2.99F));

}

}

1. Create a Windows form that allows a user to display information about a particular product in the product list.

Estimated Time: 20 Minutes

## LINQ TO SQL

“In LINQ to SQL, the data model of a relational **database is mapped to an object model** expressed in the programming language of the developer. When the application runs, **LINQ to SQL translates into SQL** the language-integrated queries in the object model and sends them to the database for execution. When the database returns the results, **LINQ to SQL translates them back to objects** that you can work with in your own programming language.

Developers using Visual Studio typically use the Object Relational Designer, which provides a user interface for implementing many of the features of LINQ to SQL.” ( <https://msdn.microsoft.com/en-us/library/bb386976(v=vs.110).aspx> )

#### LINQ to Data Driven Objects

/\* You will **need to run this code in SQL Server** for the next demos. (See DatabaseCode.sql File)

Use TempDB

Create Table Customers (ID int, Name nVarChar(50));

Insert into Customers Values (1,'Bob'), (2,'Sue'),(3,'Tim')

\*/

private void buttonLINQToDataDrivenObjects\_Click(object sender, EventArgs e)

{

Console.WriteLine("\n \*\*\* LINQ to a Data Driven Object \*\*\*");

System.Data.SqlClient.SqlConnection objCon = null;

System.Data.SqlClient.SqlCommand objCmd = null;

System.Data.SqlClient.SqlDataReader objDR = null;

try

{

objCon = new System.Data.SqlClient.SqlConnection("Data Source=(local);Initial Catalog=tempdb;Integrated Security=True");

objCmd = new System.Data.SqlClient.SqlCommand("Select ID, Name From Customers", objCon);

objCon.Open();

objDR = objCmd.ExecuteReader();

List<Customer> lstCustomers = new List<Customer>();

while (objDR.Read() == true)

{

Customer c = new Customer((int)objDR["ID"], (string)objDR["Name"]);

lstCustomers.Add(c);

}

#region Same code as in the last demo!

// **Traditional** approach has you specify how to process data from the array

Console.Write("\n Traditional approach: ");

foreach (var element in lstCustomers)

{ Console.Write(element.ToString() + " "); }

// **LINQ** approach

var objCust =

from c in lstCustomers

select c;

Console.Write("\n LINQ approach: ");

foreach (var element in objCust) { Console.Write(element + " "); }

/\*\*\* Additional Examples of that would take much more code to write \*\*\*/

// filter in a LINQ query

var objOnlySue =

from c in lstCustomers

where c.Name == "Sue"

select c;

Console.Write("\n LINQ approach Filtered Data: ");

foreach (var element in objOnlySue) { Console.Write(element); }

// order the customer by name with LINQ

var objNamesSorted =

from c in lstCustomers

orderby c.Name descending

select c;

Console.Write("\n LINQ approach Sorted Data: ");

foreach (var element in objNamesSorted) { Console.Write(element.ToString() + " "); }

//This example uses LINQ to select, or "Project", first and last names into an ANNOYMOUS type

// (Something that we see a lot in more advanced examples!)

var objIDsNamesAndMore =

from c in lstCustomers //The next line of code creates an ANNOYMOUS type, using properties you dictate.

select new {c.ID, ANameProperty = c.Name, StaticData = "a", DynamicData = DateTime.Now};

// Display ANNOYMOUS type data

Console.WriteLine("\n ANNOYMOUS type data:");

foreach (var element in objIDsNamesAndMore) {Console.Write(element.ToString() + "\n");}

#endregion

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

finally

{

objDR.Close();

objCon.Close();

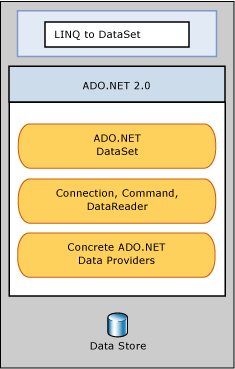
}

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

## LINQ to DataSet

“LINQ to DataSet builds on and uses the existing ADO.NET 2.0 architecture, and is not meant to replace ADO.NET 2.0 in application code. Existing ADO.NET 2.0 code will continue to function in a LINQ to DataSet application. The relationship of LINQ to DataSet to ADO.NET 2.0 and the data store is illustrated in the following diagram.” ( <https://msdn.microsoft.com/en-us/library/bb386977(v=vs.110).aspx> )



#### LINQ to DataSet Objects

/\* You will need to run code in SQL Server for the next demos. (See DatabaseCode.sql File) \*/

private void buttonLINQToDataSet\_Click(object sender, EventArgs e)

{

Console.WriteLine("\n \*\*\* LINQ To a DataSet Object \*\*\*");

System.Data.SqlClient.SqlConnection objCon = null;

System.Data.SqlClient.SqlCommand objCmd = null;

//System.Data.SqlClient.SqlDataReader objDR = null;

System.Data.SqlClient.SqlDataAdapter objDA = null;

System.Data.DataSet objDS = null;

try

{

objCon = new System.Data.SqlClient.SqlConnection("Data Source=(local);Initial Catalog=tempdb;Integrated Security=True");

objCmd = new System.Data.SqlClient.SqlCommand("Select ID, Name From Customers", objCon);

objDA = new System.Data.SqlClient.**SqlDataAdapter**(objCmd);

objDS = new **DataSet**();

//We do not need this code now...

//objCon.Open();

//objDR = objCmd.ExecuteReader();

//List<Customer> lstCustomers = new List<Customer>();

//while (objDR.Read() == true)

//{

// Customer objRow = new Customer((int)objDR["ID"], (string)objDR["Name"]);

// lstCustomers.Add(objRow);

//}

//instead we use this code...

objDA.Fill(objDS, "Customers");

DataTable objDT = objDS.Tables["Customers"];

Console.Write("\n Traditional approach: ");

// Traditional approach has you specify how to process data from the array

foreach (var element in objDT.AsEnumerable())

{

Console.Write(element["ID"].ToString() + "," + element["Name"].ToString() + " ");

}

// LINQ approach

var objCustomers = from c in objDT.AsEnumerable()

select c;

Console.Write("\n LINQ approach: ");

foreach (var element in objCustomers)

{

Console.Write(element.Field<int>("ID").ToString() + "," + element.Field<string>("Name").ToString() + " ");

}

/\*\*\* Additional Examples of that would take much more code to write \*\*\*/

// filter in a LINQ query

var objOnlySue = from c in objDT.AsEnumerable()

where c.Field<string>("Name") == "Sue"

select new { ID = c.Field<int>("ID"), Name = c.Field<string>("Name") };

Console.Write("\n LINQ approach Filtered Data: ");

foreach (var element in objOnlySue)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// order the customer by name with LINQ

var objNamesSorted = from c in objDT.AsEnumerable()

orderby c.Field<string>("Name") descending

select new { ID = c.Field<int>("ID"), Name = c.Field<string>("Name") };

Console.Write("\n LINQ approach Sorted Data: ");

foreach (var element in objNamesSorted)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// LINQ approach (using an ANNOYMOUS type)

var objIDsNamesAndMore = from c in objDT.AsEnumerable()

select new { ID = c.Field<int>("ID"), Name = c.Field<string>("Name"), StaticData = "a", DynamicData = DateTime.Now };

Console.WriteLine("\n ANNOYMOUS type data:");

foreach (var element in objIDsNamesAndMore) { Console.Write(element.ToString() + "\n"); }

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

finally

{

//objDR.Close();

//objCon.Close();

}

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

## LINQ to SQL with a Database Model (.dbml)

“LINQ to SQL offers an infrastructure (run-time) for the management of relational data as objects. It is a component of version 3.5 of the .NET Framework and ably does the translation of language-integrated queries of the object model into SQL. These queries are then sent to the database for the purpose of execution. After obtaining the results from the database, LINQ to SQL again translates them to objects.”

( <http://www.tutorialspoint.com/linq/linq_sql.htm> )

To add an LINQ to SQL Database Model, follow these steps:

1. Create a database table.
2. Create a LINQ to SQL Data Model.
3. Open Server Explorer.
4. Make a new connection to the database with the products table.
5. Drag and Drop the Products table unto the SQL Data Model window. (**Note**, that this represents ONE row of data!)
6. Verify the name of the Context Object.
7. Modify the code to connect to the database through the context object.

#### LINQ to Microsoft SQL Server with a database model

private void buttonLINQToSQL\_Click(object sender, EventArgs e)

{

Console.WriteLine("\n \*\*\* LINQ To a SQL \*\*\*");

LinqToSQLDemoDataContext objDataContext = new **LinqToSQLDemoDataContext**();

Console.Write("\n Traditional approach: ");

// Traditional approach has you specify how to process data from the array

foreach (var element in objDataContext.CustomerEntities.AsEnumerable())

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// LINQ approach

var objCustomers = from c in objDataContext.CustomerEntities.AsEnumerable()

select c;

Console.Write("\n LINQ approach: ");

foreach (var element in objCustomers)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

/\*\*\* Additional Examples of that would take much more code to write \*\*\*/

// filter in a LINQ query

var objOnlySue = from c in objDataContext.CustomerEntities.AsEnumerable()

where c.Name == "Sue"

select new { ID = c.ID, Name = c.Name };

Console.Write("\n LINQ approach Filtered Data: ");

foreach (var element in objOnlySue)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// order the customer by name with LINQ

var objNamesSorted = from c in objDataContext.CustomerEntities.AsEnumerable()

orderby c.Name descending

select new { ID = c.ID, Name = c.Name };

Console.Write("\n LINQ approach Sorted Data: ");

foreach (var element in objNamesSorted)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// LINQ approach (using an ANNOYMOUS type)

var objIDsNamesAndMore = from c in objDataContext.CustomerEntities.AsEnumerable()

select new { ID = c.ID, Name = c.Name, StaticData = "a", DynamicData = DateTime.Now };

Console.WriteLine("\n ANNOYMOUS type data:");

foreach (var element in objIDsNamesAndMore) { Console.Write(element.ToString() + "\n"); }

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

# **Demo**: Using LINQ to Arrays and Objects

Your instructor will now demo how to use LINQ to SQL.

# Lab 06-2

In this lab, you will create a list of Project objects and use LINQ to display their data.

1. Create a database table.

Use TempDB

Go

Create Table Products

(ID int Primary Key, Name nVarChar(50), Price float);

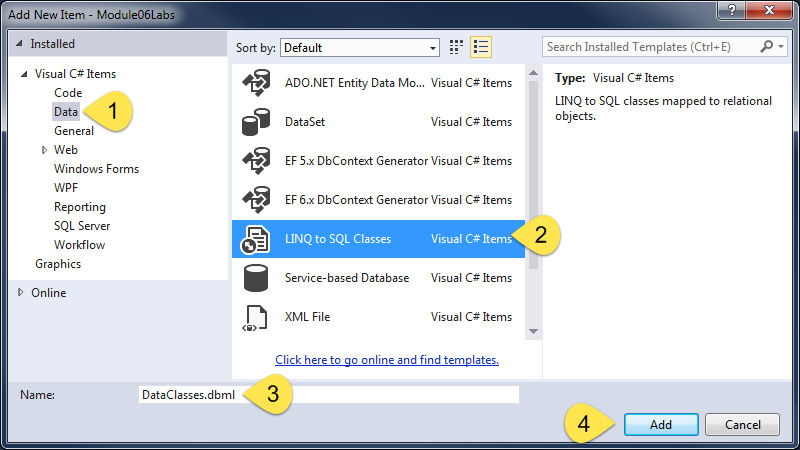
Insert into Products

Values (1,'ProdA', 9.99),

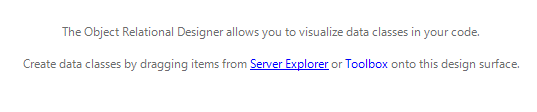
(2,'ProdB', 19.99),

(3,'ProdC', 2.99);

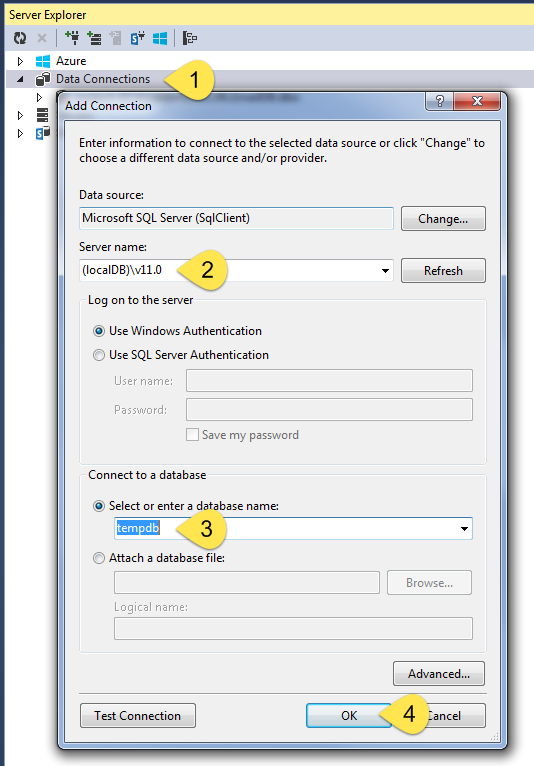
1. Create a LINQ to SQL Data Model.



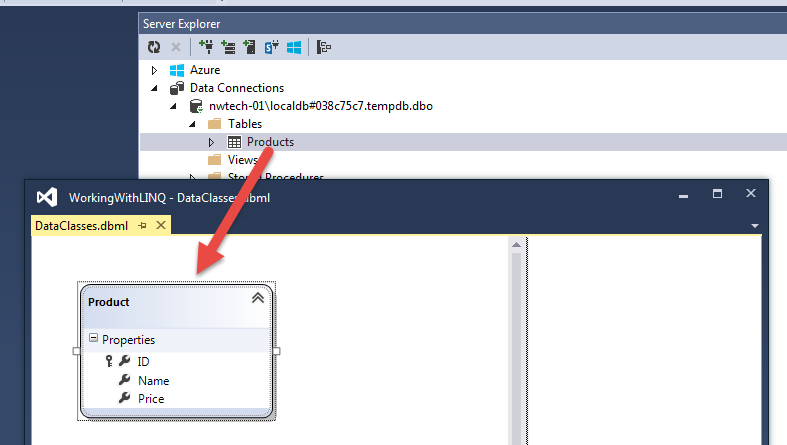
1. Open Server Explorer.



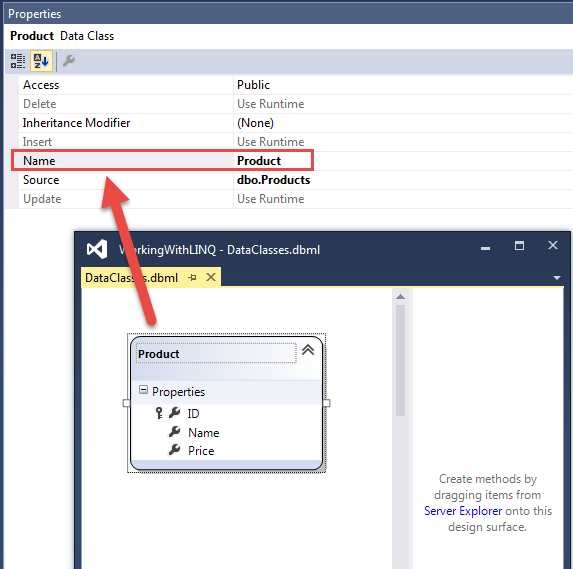
1. Make a new connection to the database with the products table.



1. Drag and Drop the Products table onto the SQL Data Model window. (Note, that this represents ONE row of Product data!)



1. Verify the name of the Context Object.



1. Modify your existing code to use the new LINQ to SQL data model.

## LINQ to SQL vs the Entity Framework

“Question #2: **Where does Microsoft stand on LINQ to SQL?**

Answer: We would like to be very transparent with our customers about our intentions for future innovation **with respect to LINQ to SQL and the Entity Framework.**

In .NET 4.0, we continue to invest in both technologies. Within LINQ to SQL, we made a number of performance and usability enhancements, as well as updates to the class designer and code generation. Within the Entity Framework, we listened to a great deal to customer feedback and responded with significant investments including better foreign key support, T4 code generation, and POCO support.

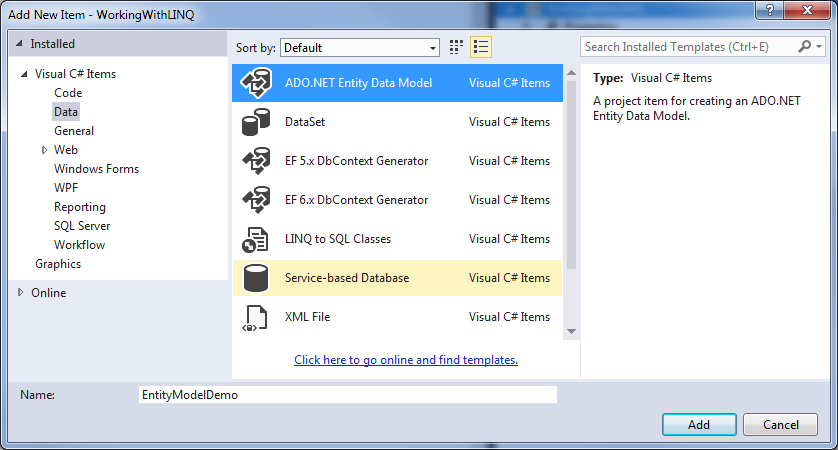
Moving forward, **Microsoft is committing to supporting both technologies** as important parts of the .NET Framework, adding new features that meet customer requirements. We do, however, **expect that the bulk of our overall investment will be in the Entity Framework**, as this framework is built around the Entity Data Model (EDM). **EDM** represents a key strategic direction for Microsoft that **spans many of our products**, including SQL Server, .NET, and Visual Studio. EDM-based tools, languages and frameworks are important technologies that enable our customers and partners to increase productivity across the development lifecycle and enable better integration across applications and data sources.” ( <https://msdn.microsoft.com/en-us/data/bb525059.aspx#Q3> )

## The Entity Framework

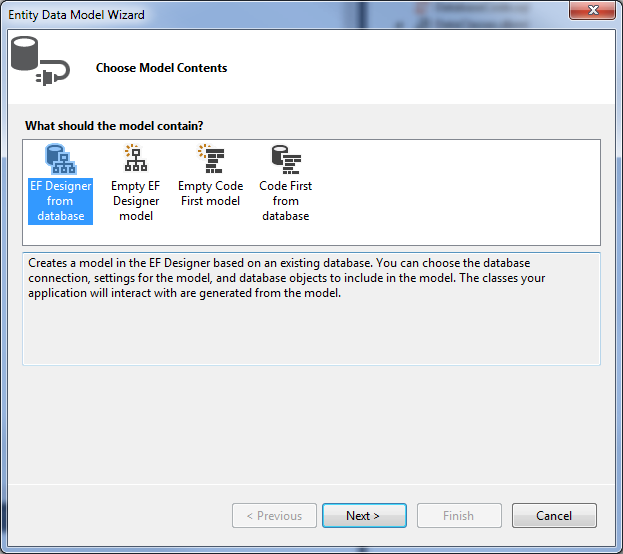
“A part of the ADO.NET Entity Framework, LINQ to Entities is more flexible than LINQ to SQL, but is not much popular because of its complexity and lack of key features. However, it does not have the limitations of LINQ to SQL that allows data query only in SQL server database as LINQ to Entities facilitates data query in a large number of data providers like Oracle, MySQL, etc.” (<http://www.tutorialspoint.com/linq/linq_entities.htm> )

To add an Entity Framework Database Model, follow these steps:

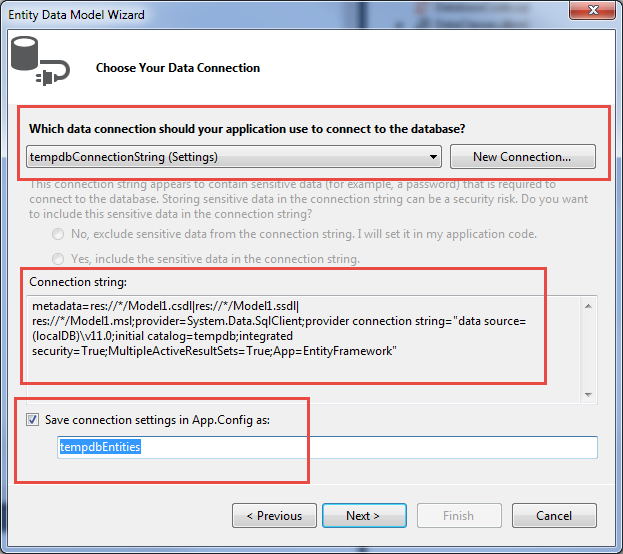
1. Create a database table.
2. Create a LINQ to Entity Model.



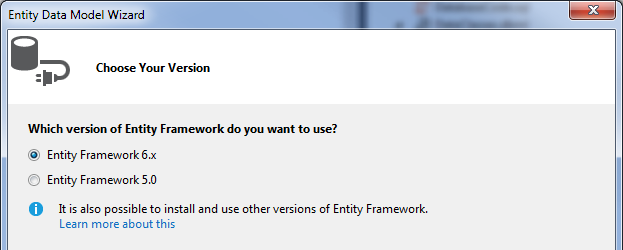
1. Choose the model’s contents



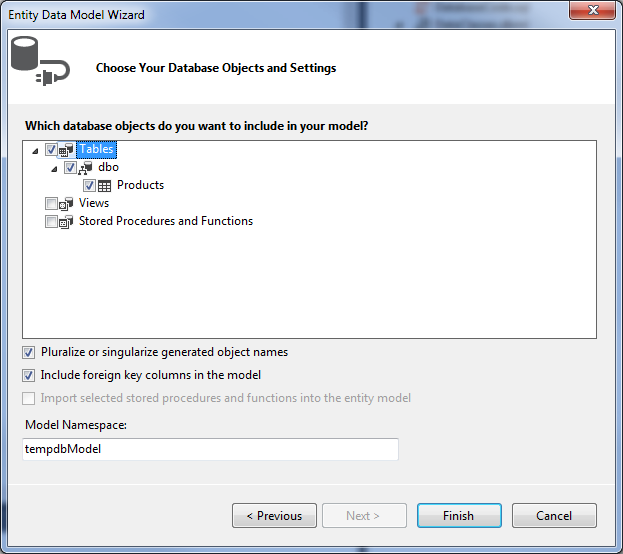
1. Make a new connection to the database with the products table (as needed).



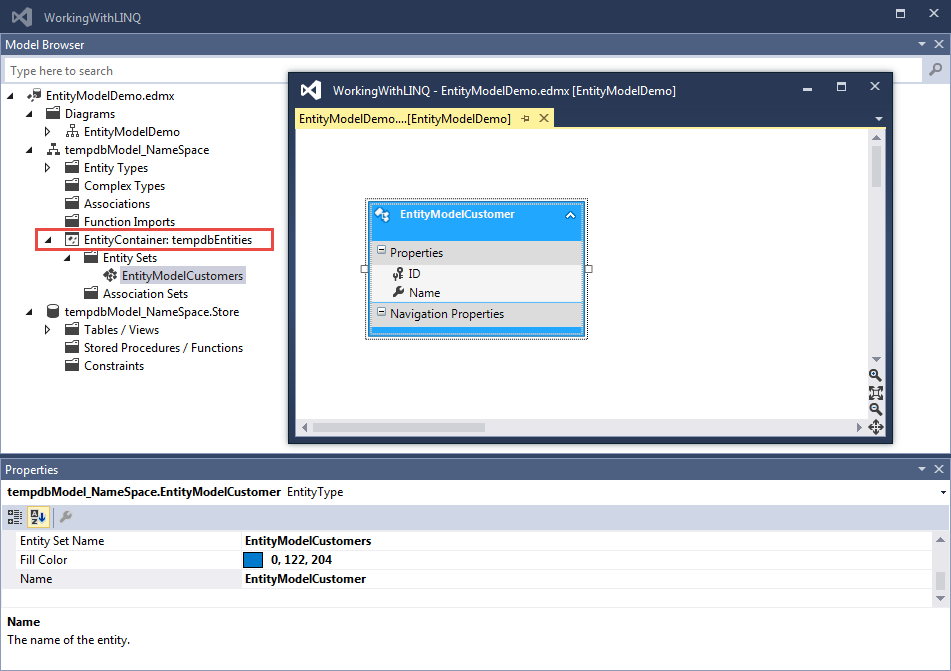
1. Choose the version of EF you wish to use.



1. Choose the object or objects you wish to Linq to.



1. Verify the names of the EF Context Objects.



1. Modify the code to connect to the database through the context object.

#### LINQ to Many Vendor's SQL Server objects

private void buttonLINQToDataEntities\_Click(object sender, EventArgs e)

{

Console.WriteLine("\n \*\*\* LINQ To a Data Entity \*\*\*");

using (tempdbEntities objContext = new **tempdbEntities(**))

{

Console.Write("\n Traditional approach: ");

// Traditional approach has you specify how to process data from the array

foreach (var element in objContext.EntityModelCustomers.AsEnumerable())

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// LINQ approach

var objCustomers = from c in objContext.EntityModelCustomers.AsEnumerable()

select c;

Console.Write("\n LINQ approach: ");

foreach (var element in objCustomers)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

/\*\*\* Additional Examples of that would take much more code to write \*\*\*/

// filter in a LINQ query

var objOnlySue = from c in objContext.EntityModelCustomers.AsEnumerable()

where c.Name == "Sue"

select new { ID = c.ID, Name = c.Name };

Console.Write("\n LINQ approach Filtered Data: ");

foreach (var element in objOnlySue)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// order the customer by name with LINQ

var objNamesSorted = from c in objContext.EntityModelCustomers.AsEnumerable()

orderby c.Name descending

select new { ID = c.ID, Name = c.Name };

Console.Write("\n LINQ approach Sorted Data: ");

foreach (var element in objNamesSorted)

{

Console.Write(element.ID.ToString() + "," + element.Name.ToString() + " ");

}

// LINQ approach (using an ANNOYMOUS type)

var objIDsNamesAndMore = from c in objContext.EntityModelCustomers.AsEnumerable()

select new { ID = c.ID, Name = c.Name, StaticData = "a", DynamicData = DateTime.Now };

Console.WriteLine("\n ANNOYMOUS type data:");

foreach (var element in objIDsNamesAndMore) { Console.Write(element.ToString() + "\n"); }

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

# **Demo**: Using the Entity Framework

Your instructor will now demo how to use the Entity Framework to connect to a SQL Server table.

# Lab 06-3

In this lab, you will add and display data from the Emails table.

To add an Entity Framework Database Model, follow these steps:

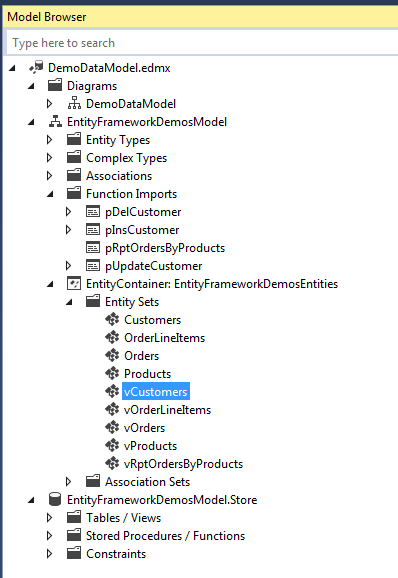
1. Create a database table.
2. Create a LINQ to Entity Model.
3. Choose the model’s contents
4. Make a new connection to the database with the products table (as needed).
5. Choose the version of EF you wish to use.
6. Choose the object or objects you wish to LINQ to.
7. Verify the names of the EF Context Objects.
8. Modify the code to connect to the database through the context object.

Estimated Time: 20 Minutes

## Using Views and Stored Procedures

Whenever programming to a database you should not use the tables directly. Instead you should always program though abstraction layer objects like Views and Stored Procedures. The process is not much different than working directly with a table and it provides better security and lower cost of ownership for your application.

In the Entity Framework, Views should up as Entity Sets and Stored Procedures show as Function Imports.



Once the EF context model is created you just have to write code to interact with its object model. Here is an example.

#### Entity Framework using SQL Views and Store Procedures

using (EntityFrameworkDemosProcessor.EntityFrameworkDemosEntities objContext = new EntityFrameworkDemosEntities())

{ /\* NOTE YOU MUST ADD a reference to the System.Data.Entity.dll before this will work in a Console application \*/

//Select From a Table

Console.WriteLine("Select From a Table");

var objProducts = from p in objContext.Products

select p;

foreach (var row in objProducts)

{

Console.WriteLine("The ID: {0}, Name: {1}, Price {2};", row.ProductID, row.ProductName, row.ProductPrice);

}

//Select From a View

Console.WriteLine("Select From a View");

var objVProducts = from p in objContext.vProducts

select p;

foreach (var row in objVProducts)

{

Console.WriteLine("The ID: {0}, Name: {1}, Price {2};", row.ProductID, row.ProductName, row.ProductPrice);

}

//Select From a complex Reporting View

Console.WriteLine("Select From a complex Reporting View");

var objvRptProducts = objContext.vRptOrdersByProducts;

foreach (var row in objvRptProducts)

{

Console.WriteLine("The Customer: {0}, Product: {1}, Date:{2};", row.CustomerName, row.ProductName, row.OrderDate);

}

//Inserting Directly to table (Not Recommended)

Console.WriteLine("Inserting Directly to table (Not Recommended)");

try

{

objContext.Products.AddObject(new Product() { ProductID = 300, ProductName = "ProdC", ProductPrice = 9.99M });

objContext.SaveChanges(); //This code send the change to the actual database

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

Console.WriteLine("Now check to see if the new row was added!"); Console.ReadLine();

//Insert with a Stored Procedure (Recommended!)

Console.WriteLine("Insert with a Stored Procedure");

objContext.pInsCustomer(3,"Tim Tomas");

Console.WriteLine("Now check to see if the new row was added!"); Console.ReadLine();

//Update with a Stored Procedure

Console.WriteLine("Update with a Stored Procedure");

objContext.pUpdateCustomer(3,"Tim Thomas");

Console.WriteLine("Now check to see if the row was modified!"); Console.ReadLine();

//Delete with a Stored Procedure

Console.WriteLine("Update with a Stored Procedure");

objContext.pDelCustomer(3);

Console.WriteLine("Now check to see if the row was deleted!"); Console.ReadLine();

# **Demo**: Using the Entity Framework

Your instructor will now demo how to use the Entity Framework to connect to a SQL Server abstraction objects (views and stored procedures).