Articles » Platforms, Frameworks & Libraries » LINQ » General

One-Many and One-One relationships using LINQ to SQL

By **Shivprasad koirala**, 6 Jul 2009

★★★★☆ 3.94 (15 votes)

Download source code - 1.02 MB

Table of contents

- Introduction
- Previous LINQ, Silverlight, WCF, WPF, and WWF articles
- Simplest LINQ to SQL example
- Encapsulated LINQ classes with Set and Get
- One-Many and One-One relationships
- Source code

Introduction

In this article, we will start with a basic LINQ to SQL example and then see how we can implement one-many and one-one relationships using `Entityref` and `EntitySet`. I have attached a source code which demonstrates this in a practical manner.

Catch my videos for WCF, WPF, WWF, LINQ, Silverlight, Design Patterns, UML, and a lot on http://www.questpond.com.

Previous LINQ, Silverlight, WCF, WPF, and WWF articles

Below are some of my old articles which you would want to refer to in case you are not acquainted with .NET 3.5 basic concepts.

1. This article talks about a complete three tier implementation using LINQ:

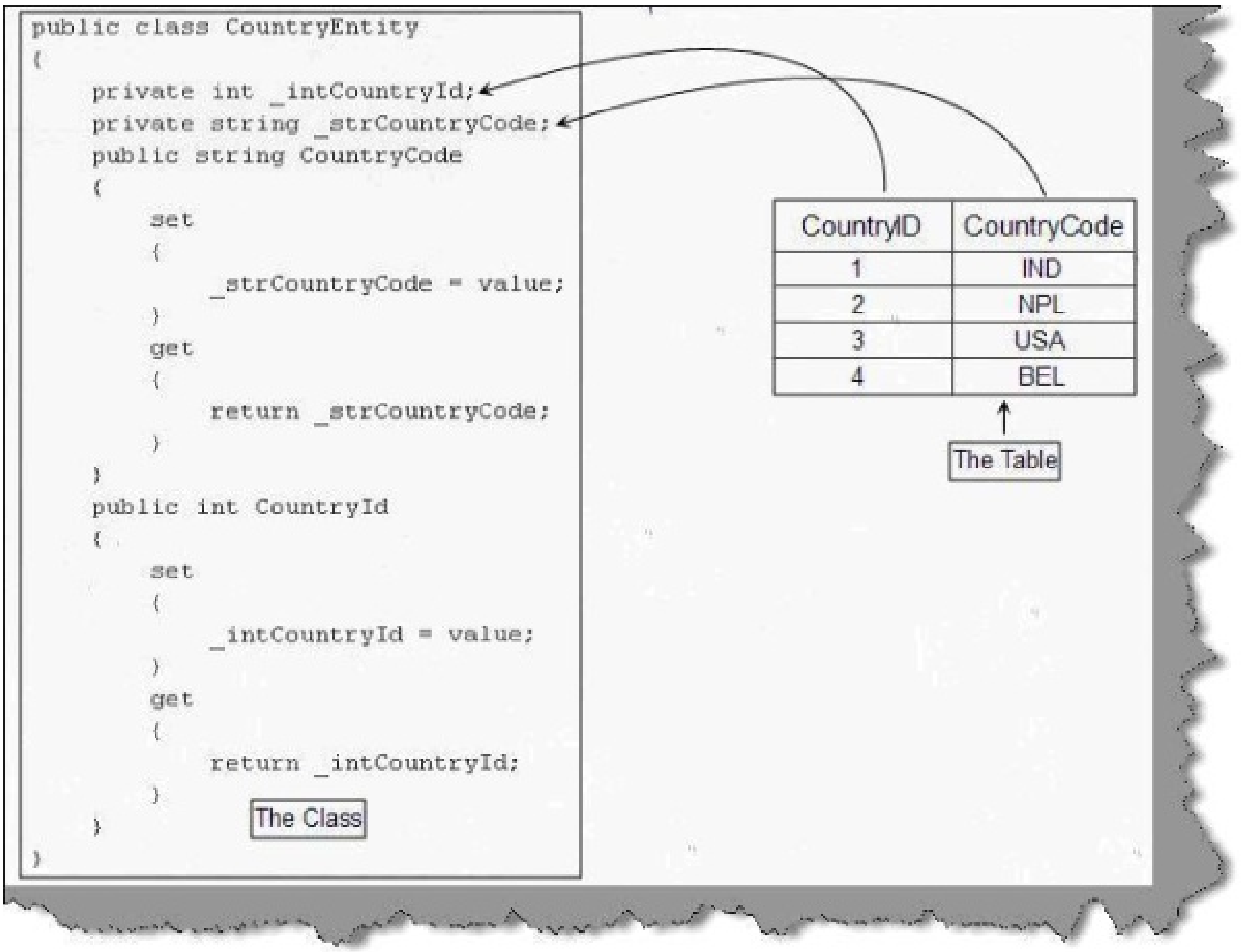
http://www.codeproject.com/KB/aspnet/SaltAndPepper.aspx
2. WCF FAQ series covering simple to advanced concepts: http://www.codeproject.com/KB/aspnet/WCF.aspx
3. WPF and Silverlight FAQ series covering layout, animation, and bindings: http://www.codeproject.com/KB/WPF/WPFSilverLight.aspx
4. Windows workflow FAQ series covering state machines and sequential workflow in detail: http://www.codeproject.com/KB/WF/WWF.aspx

Simplest LINQ to SQL example

Let's first start with a simple LINQ to SQL example and then we will try to understand how we can establish relationships in LINQ entities.

Step 1: Define Entity classes using LINQ

When we design a project using a tiered approach like 3-tier or N-tier, we need to create business classes and objects. For instance, below is a simple class mapped to a country table. You can see how the class properties are mapped to one fashion with the table. These types of classes are termed as entity classes.



```
public class CountryEntity
{
    private int _intCountryId;
    private string _strCountryCode;
    public string CountryCode
    {
        set
        {
            _strCountryCode = value;
        }
        get
        {
            return _strCountryCode;
        }
    }
    public int CountryId
    {
        set
        {
            _intCountryId = value;
        }
        get
        {
            return _intCountryId;
        }
    }
}
```

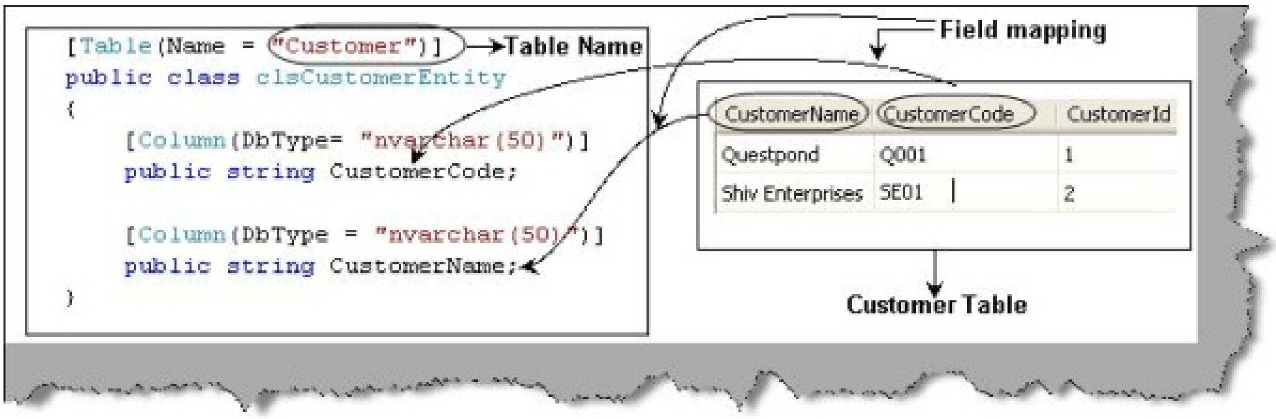
CountryID	CountryCode
1	IND
2	NPL
3	USA
4	BEL

In LINQ, we need to first define these entity classes using attribute mappings. You need to import the `System.Data.Linq.Mapping` namespace to get attributes for mapping. Below is a code snippet which shows how the `Table` attribute maps the class with the database table name Customer and how the `Column` attributes help mapping properties with table columns.

```
[Table(Name = "Customer")]
```

```
public class clsCustomerEntityWithProperties
{
    private int _CustomerId;
    private string _CustomerCode;
    private string _CustomerName;

    [Column(DbType = "nvarchar(50)")]
    public string CustomerCode
    {
        set
        {
            _CustomerCode = value;
        }
        get
        {
            return _CustomerCode;
        }
    }

    [Column(DbType = "nvarchar(50)")]
    public string CustomerName
    {
        set
        {
            _CustomerName = value;
        }
        get
        {
            return _CustomerName;
        }
    }

    [Column(DbType = "int", IsPrimaryKey = true)]
    public int CustomerId
    {
        set
        {
            _CustomerId = value;
        }
        get
        {
            return _CustomerId;
        }
    }
}
```

Below is a more sophisticated pictorial view of the entity classes mapping with the customer table structure:



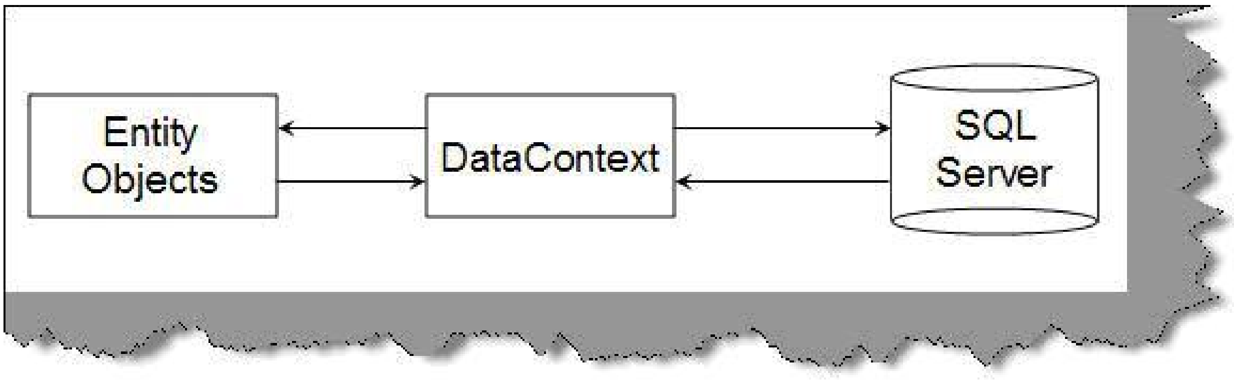
```
[Table(Name = "Customer")]
public class clsCustomerEntity
{
    [Column(DbType= "nvarchar(50)")]
    public string CustomerCode;

    [Column(DbType = "nvarchar(50)")]
    public string CustomerName;
}
```

CustomerName	CustomerCode	CustomerId
Questpond	Q001	1
Shiv Enterprises	SE01	2

Step 2: Using the DataContext to bind the table

data with the entity objects

The second step is to use the DataContext object of LINQ to fill your entity objects. DataContext acts like a mediator between database objects and your LINQ entity mapped classes.



The first thing is to create an object of DataContext and create an active connection using the SQL connection string.

```
DataContext objContext = new DataContext(strConnectionString);
```

The second thing is to get the entity collection using the Table data type. This is done using the GetTable function of the DataContext.

```
Table<clsCustomerEntity> objTable = objContext.GetTable<clsCustomerEntity>();
```

Once we get all the data in table collection, it’s time to browse through the table collection and display the records.

```
foreach (clsCustomerEntity objCustomer in objTable)
{
    Response.Write(objCustomer.CustomerName + "<br>");
}
```

You can get the above source code from the download attached with this article.

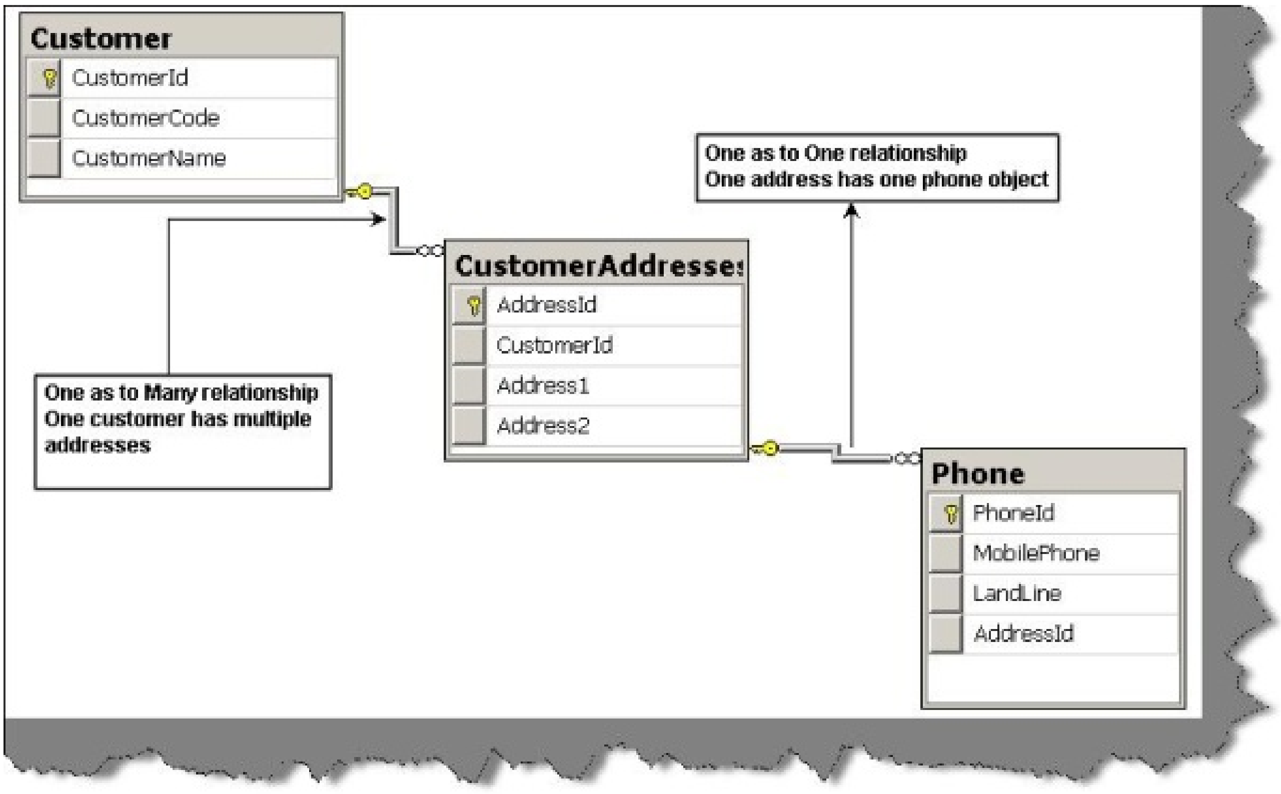
Encapsulated LINQ classes with Set and Get

In the previous example, we had exposed the entity class properties as public properties, which violates the basic rule of encapsulation. You can define setter and getter functions which encapsulate the private properties.

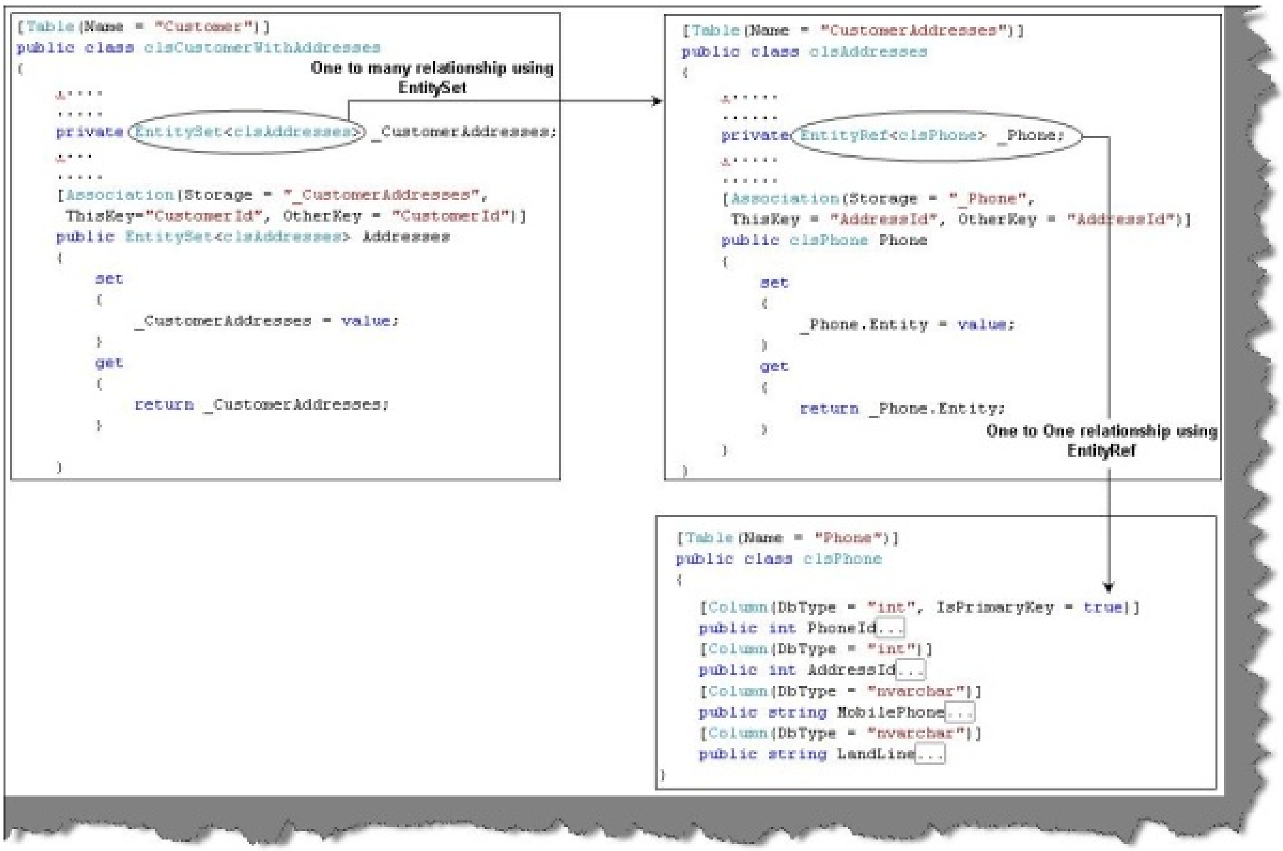
```
[Table(Name = "Customer")]
public class clsCustomerEntityWithProperties
{
    private int _CustomerId;
    private string _CustomerCode;
    private string _CustomerName;

    [Column(DbType = "nvarchar(50)")]
    public string CustomerCode
    {
        set
        {
            _CustomerCode = value;
        }
        get
        {
            return _CustomerCode;
        }
    }

    [Column(DbType = "nvarchar(50)")]
    public string CustomerName
    {
        set
        {
            _CustomerName = value;
        }
        get
        {
            return _CustomerName;
        }
    }

    [Column(DbType = "int", IsPrimaryKey = true)]
    public int CustomerId
    {
        set
        {
            _CustomerId = value;
        }
        get
        {
            return _CustomerId;
        }
    }
}
```

One-Many and One-One relationships

LINQ helps you define relationships using `EntitySet` and `EntityRef`. To understand how we can define relationships using LINQ, let's consider the below example where we have a customer who can have many addresses and every address will have phone details. In other words, customer and address has a one-many relationship while address and phone has a one-one relationship.



To define a one-many relationship between the customer and address classes, we need to use the `EntitySet` attribute. To define a one-one relationship between the address and phone class, we need to use the `EntityRef` attribute.



```
[Table(Name = "Customer")]
public class clsCustomerWithAddresses
{
    .....
    .....
    private EntitySet<clsAddresses> _CustomerAddresses;
    ....
    .....
    [Association(Storage = "_CustomerAddresses",
     ThisKey="CustomerId", OtherKey = "CustomerId")]
    public EntitySet<clsAddresses> Addresses
    {
        set
        {
            _CustomerAddresses = value;
        }
        get
        {
            return _CustomerAddresses;
        }

    }
```

```
[Table(Name = "CustomerAddresses")]
public class clsAddresses
{
    ......
    ......
    private EntityRef<clsPhone> _Phone;
    ......
    ......
    [Association(Storage = "_Phone",
     ThisKey = "AddressId", OtherKey = "AddressId")]
    public clsPhone Phone
    {
        set
        {
            _Phone.Entity = value;
        }
        get
        {
            return _Phone.Entity;
        }
    }
}
```

```
[Table(Name = "Phone")]
public class clsPhone
{
    [Column(DbType = "int", IsPrimaryKey = true)]
    public int PhoneId...
    [Column(DbType = "int")]
    public int AddressId...
    [Column(DbType = "nvarchar")]
    public string MobilePhone...
    [Column(DbType = "nvarchar")]
    public string LandLine...
}
```

Note: You need to define a primary key attribute for every entity class or else the mapping relationship will not work.

Shown below is the class entity snippet for the customer class which shows how it has used `EntitySet` to define one-many relationship with the address class. Association is defined using the `Association` attribute. The `Association` attribute has three important properties: `storage`, `thiskey`, and `otherkey`.

`storage` defines the name of the private variable where the address object is stored. Currently it is `_CustomerAddresses`. `ThisKey` and `OtherKey` define which property will define the linkage; for this instance, it is `CustomerId`. In other words, both the `Customer` class and the `Address` class will have a `CustomerId` property in common. `ThisKey` defines the name of the property for the `Customer` class while `OtherKey` defines the property of the addresses class.

```
[Table(Name = "Customer")]
public class clsCustomerWithAddresses
{
    private EntitySet<clsAddresses> _CustomerAddresses;

    [Association(Storage = "_CustomerAddresses",
      ThisKey="CustomerId", OtherKey = "CustomerId")]
    public EntitySet<clsAddresses> Addresses
    {
        set
        {
            _CustomerAddresses = value;
        }
        get
        {
            return _CustomerAddresses;
        }
    }
}
```

Below is the complete code snippet with other properties of the customer class:

```
[Table(Name = "Customer")]
public class clsCustomerWithAddresses
{
    private int _CustomerId;
    private string _CustomerCode;
    private string _CustomerName;
    private EntitySet<clsAddresses> _CustomerAddresses;

    [Column(DbType="int",IsPrimaryKey=true)]
    public int CustomerId
    {
        set
        {
            _CustomerId = value;
        }
        get
        {
            return _CustomerId;
        }
    }

    [Column(DbType = "nvarchar(50)")]
    public string CustomerCode
    {
        set
        {
            _CustomerCode = value;
        }
        get
        {
            return _CustomerCode;
        }
    }

    [Column(DbType = "nvarchar(50)")]
    public string CustomerName
    {
        set
        {
            _CustomerName = value;
        }
        get
        {
            return _CustomerName;
        }
    }

    [Association(Storage = "_CustomerAddresses",
      ThisKey="CustomerId", OtherKey = "CustomerId")]
    public EntitySet<clsAddresses> Addresses
    {
        set
        {
            _CustomerAddresses = value;
        }
        get
        {
            return _CustomerAddresses;
        }
    }
}
```

To define the relationship between the address class and the phone class, we need to use the `EntityRef` syntax. So below is the code snippet which defines the relationship using `EntityRef`. All the other properties are same except that we need to define the variable using `EntityRef`.

```
public class clsAddresses
{
    private int _AddressId;
    private EntityRef<clsPhone> _Phone;

    [Column(DbType = "int", IsPrimaryKey = true)]
    public int AddressId
    {
        set
        {
            _AddressId = value;
        }
        get
        {
            return _AddressId;
        }
    }
    [Association(Storage = "_Phone",
    ThisKey = "AddressId", OtherKey = "AddressId")]
    public clsPhone Phone
    {
        set
        {
            _Phone.Entity = value;
        }
        get
        {
            return _Phone.Entity;
        }
    }
}
```

Below is a complete address class with other properties:

```
public class clsAddresses
{
    private int _Customerid;
    private int _AddressId;
    private string _Address1;
    private EntityRef<clsPhone> _Phone;
    [Column(DbType="int")]
    public int CustomerId
    {
        set
        {
            _Customerid = value;
        }
        get
        {
            return _Customerid;
        }
    }
    [Column(DbType = "int", IsPrimaryKey = true)]
    public int AddressId
    {
        set
        {
            _AddressId = value;
        }
        get
        {
            return _AddressId;
        }
    }
    [Column(DbType = "nvarchar(50)")]
    public string Address1
    {
        set
        {
            _Address1 = value;
        }
        get
        {
            return _Address1;
        }
    }
    [Association(Storage = "_Phone",
    ThisKey = "AddressId", OtherKey = "AddressId")]
    public clsPhone Phone
    {
        set
        {
            _Phone.Entity = value;
        }
        get
        {
            return _Phone.Entity;
        }
    }
}
```

The phone class which is aggregated with the address class:

```
[Table(Name = "Phone")]
public class clsPhone
{
    private int _PhoneId;
    private int _AddressId;
    private string _MobilePhone;
    private string _LandLine;

    [Column(DbType = "int", IsPrimaryKey = true)]
    public int PhoneId
    {
    set
        {
            _PhoneId = value;
        }
        get
        {
            return _PhoneId;
        }
    }
    [Column(DbType = "int")]
    public int AddressId
    {
        set
        {
            _PhoneId = value;
        }
        get
        {
            return _PhoneId;
        }
    }
    [Column(DbType = "nvarchar")]
    public string MobilePhone
    {
        set
        {
            _MobilePhone = value;
        }
        get
        {
            return _MobilePhone;
        }
    }
    [Column(DbType = "nvarchar")]
    public string LandLine
    {
        set
        {
            _LandLine = value;
        }
        get
        {
            return _LandLine;
        }
    }
}
```

Now finally we need to consume this relationship in our ASPX client behind code:

The first step is to create the data context object with the connection initialized:

```
DataContext objContext = new DataContext(strConnectionString);
```

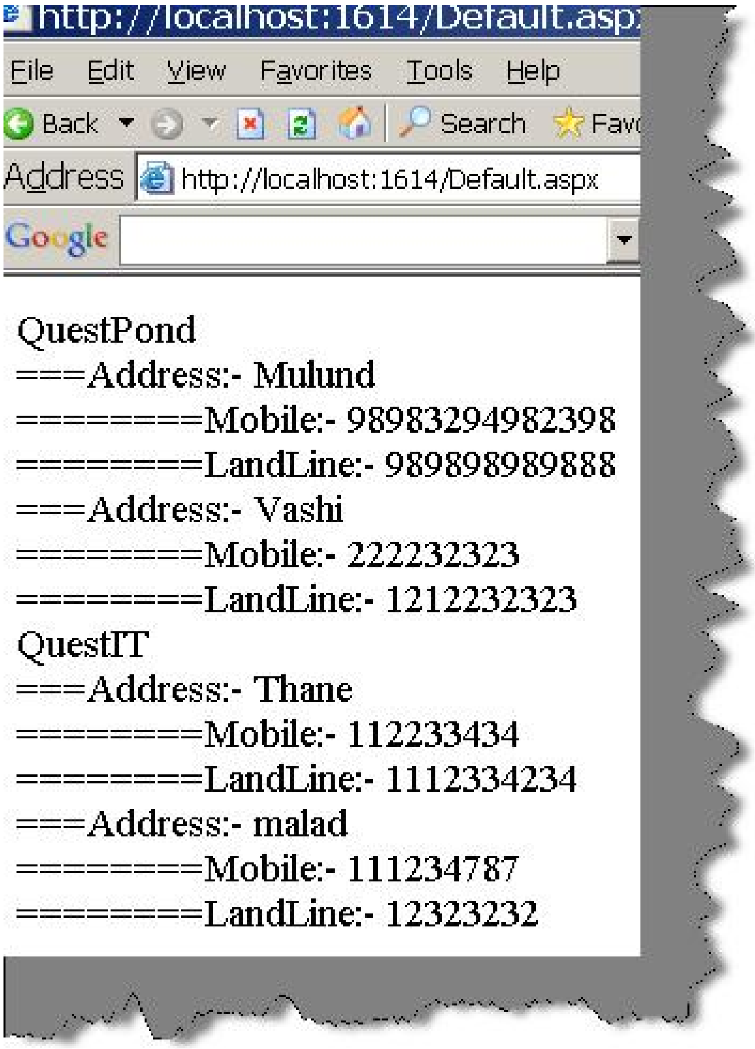
The second step is firing the query. Please note that we are just firing the query for the customer class. The LINQ engine ensures that all child table data is extracted and placed as per the relationships defined in the entity classes.

```
var MyQuery = from objCustomer in objContext.GetTable<clsCustomerWithAddresses>()
select objCustomer;
```

Finally, we loop through the customer, loop through the corresponding addresses object, and display phone details as per the phone object.

```
foreach (clsCustomerWithAddresses objCustomer in MyQuery)
{
    Response.Write(objCustomer.CustomerName + "<br>");
    foreach (clsAddresses objAddress in objCustomer.Addresses)
    {
        Response.Write("===Address:- " + objAddress.Address1 + "<br>");
        Response.Write("========Mobile:- " + objAddress.Phone.MobilePhone + "<br>");
        Response.Write("========LandLine:- " +
                       objAddress.Phone.LandLine + "<br>");
    }
}
```

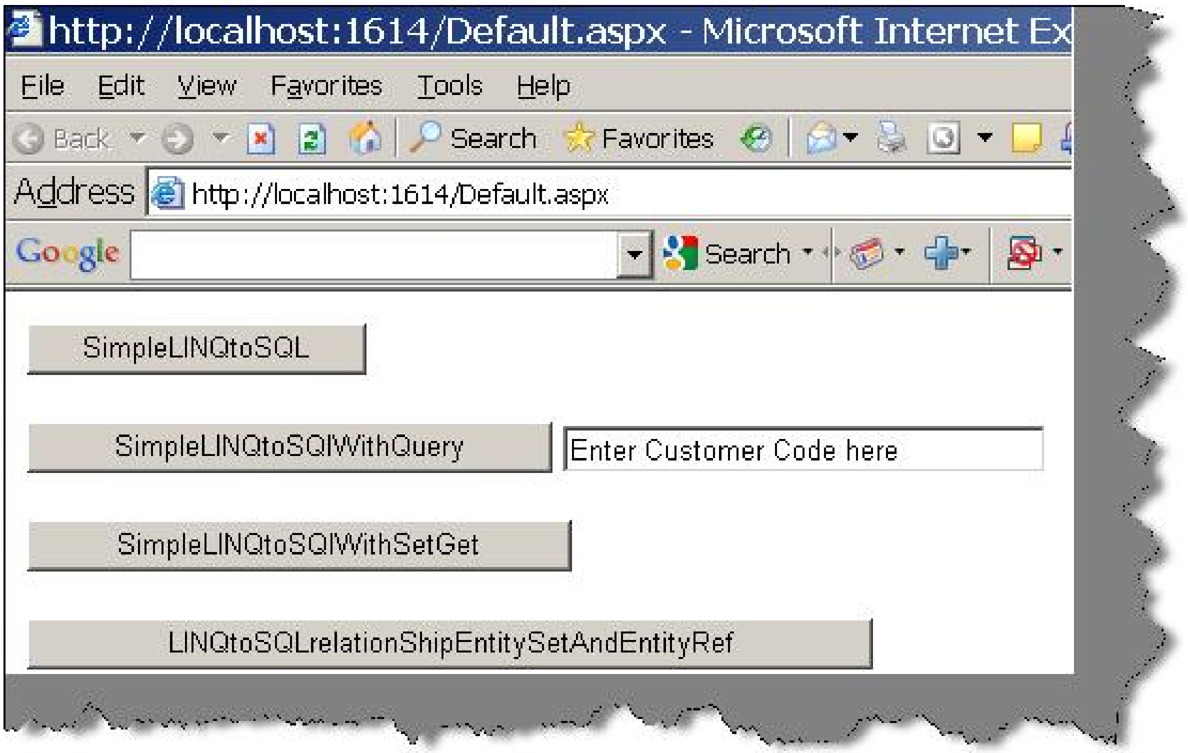
The output looks as shown below. Every customer has multiple addresses and every address has a phone object.


http://localhost:1614/Default.asp
File
Edit
View
Favorites
Tools
Help
Back
Search
Favo
Address
http://localhost:1614/Default.aspx
Google

```
QuestPond
===Address:- Mulund
========Mobile:- 98983294982398
========LandLine:- 989898989888
===Address:- Vashi
========Mobile:- 222232323
========LandLine:- 1212232323
QuestIT
===Address:- Thane
========Mobile:- 112233434
========LandLine:- 1112334234
===Address:- malad
========Mobile:- 111234787
========LandLine:- 12323232
```

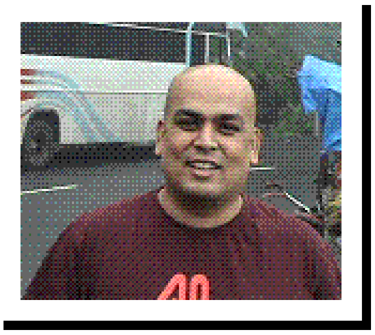
Source code

We have also attached a source which has the customer, address, and phone tables. The sample code demonstrates a simple LINQ example, LINQ example with properties, and relationship LINQ example with `entityref` and `entityset`.


http://localhost:1614/Default.aspx - Microsoft Internet Ex
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:1614/Default.aspx
Google Search
SimpleLINQtoSQL
SimpleLINQtoSQlWithQuery
Enter Customer Code here
SimpleLINQtoSQlWithSetGet
LINQtoSQLrelationShipEntitySetAndEntityRef

License

About the Author



Shivprasad koirala
Architect http://www.questpond.com
India

I am a Microsoft MVP for ASP/ASP.NET and currently a CEO of a small E-learning company in India. We are very much active in making training videos , writing books and corporate trainings. Do visit my site for .NET, C# , design pattern , WCF , Silverlight , LINQ , ASP.NET , ADO.NET , Sharepoint , UML , SQL Server training and Interview questions and answers

Comments and Discussions

10 messages have been posted for this article Visit **http://www.codeproject.com/Articles/37784/One-Many-and-One-One-relationship-using-LINQ-to-SQ** to post and view comments on this article, or click **here** to get a print view with messages.