Entity Framework – Code First Approach – Part 3

This will show walkthrough of following concepts

1- LINQ to Entities
2- Executing a query using EF

This tutorial is prepared with Visual Studio 2013 + SQL Server.

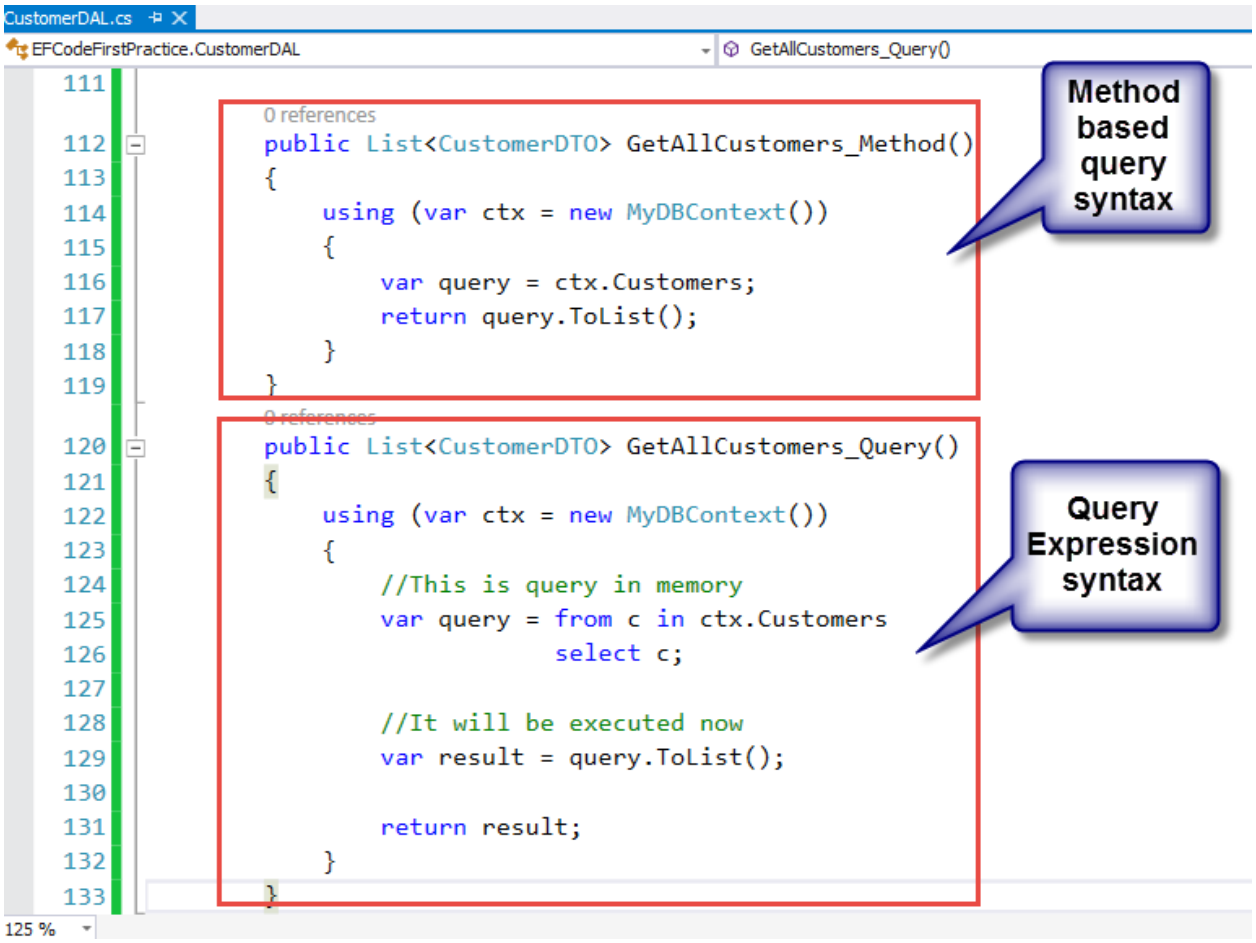
Pre-requisite: Entity Framework – Code First Approach – Part 2

Version	Last updated	Comments	Modified By
V1.0	28-04-2016		Bilal Shahzad

Step by Step Walkthrough

1- **LINQ to Entities** queries can be composed in two different syntaxes: query expression syntax and method-based query syntax.
2- Before moving forward in this tutorial, Go through this link quickly
   a. https://msdn.microsoft.com/en-us/library/bb399367(v=vs.100).aspx

3- Requirement 1: Give me all customers.



4- Requirement 2: Give me all customers where name starts with “Bil”.

CustomerDAL | GetCustomersByName_Query(String name)

```
0 references
public List<CustomerDTO> GetCustomersByName_Method(String name)
{
    using (var ctx = new MyDBContext())
    {
        var query = ctx.Customers.Where(c=> c.Name.StartsWith(name));
        return query.ToList();
    }
}
0 references
public List<CustomerDTO> GetCustomersByName_Query(String name)
{
    using (var ctx = new MyDBContext())
    {
        //This is query in memory
        var query = from c in ctx.Customers
                    where c.Name.StartsWith(name) == true
                    select c;
        //It will be executed now
        var result = query.ToList();
        return result;
    }
}
```

Method Based Syntax

Query Expression Syntax

5- Requirement 3: Give me ID & name of all customers where CustomerID is greater than 3. Following screenshot is showing how to achieve this.

AL.cs* | FirstPractice.CustomerDAL | GetCustomersByNameLen_Query()

```
        0 references
        public List<CustomerDTO> GetCustomersByNameLen_Method()
        {
            using (var ctx = new MyDBContext())
            {
                var query = ctx.Customers
                    .Where(c => c.Name.Length > 3)
                    .Select(c => new { c.CustomerID, c.Name });

                return query.ToList();
            }
        }
        0 references
        public List<CustomerDTO> GetCustomersByNameLen_Query()
        {
            using (var ctx = new MyDBContext())
            {
                var query = from c in ctx.Customers
                            where c.Name.Length > 3
                            select new { c.CustomerID, c.Name };
                var result = query.ToList();
                return result;
            }
        }
    }
```

Here result of query is not mapping to any entity but is of "anonymous type". Therefore it is showing error here. Same is happening in other function.

6- Problem in above case is “selection or projection” of specific columns but not a whole entity. So in this case, result is of “anonymous” type and you can’t return “Anonymous” types from a function. What We can do is
   a. Create a new type (DTO) which will have only these properties
   b. Or return List<Object> which will not be very helpful for caller of these functions.

```
practice.CustomerDAL                                   GetCustomersByNameLen_Query()
        0 references
        public List<Object> GetCustomersByNameLen_Method()
        {
            using (var ctx = new MyDBContext())
            {
                var query = ctx.Customers
                    .Where(c => c.Name.Length > 3).Select(c => new { c.CustomerID, c.Name });
                List<Object> list = new List<Object>();
                foreach (var o in query)
                {
                    list.Add(o);
                }
                return list;
            }
        }
        0 references
        public List<Object> GetCustomersByNameLen_Query()
        {
            using (var ctx = new MyDBContext())
            {
                var query = from c in ctx.Customers
                            where c.Name.Length > 3
                            select new { c.CustomerID, c.Name };
                return query.ToList();
            }
```

7- In above sample screenshot, we can see that if we return our list of anonymous objects, it doesn’t allow that but if we first store our anonymous objects into a list of objects, it works.
8- Requirement 4: Give me Customer Name & Account Number only for all customers. (Here we’ll have to use some sort of join). But first create a new class file which will hold this information. Here we can see that there is no “table” mapping is provided as it is not against any table. We’ll also **not** define any “DbSet” for this DTO in our context class.

```
ustAccountSimpleDTO.cs   CustomerDAL.cs
EFCodeFirstPractice.CustAccountSimpleDTO   AccountNumber
3   using System.Linq;
4   using System.Text;
5
6   namespace EFCodeFirstPractice
7   {
        0 references
8       public class CustAccountSimpleDTO
9       {
            0 references
10          public String CustomerName { get; set; }
            0 references
11          public String AccountNumber { get; set; }
12      }
13  }
14
```

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'EFCodeFirstPractice3' (1 proj
EFCodeFirstPractice3
Properties
References
App.config
CustAccountsDTO.cs
CustAccountSimpleDTO.cs
CustomerDAL.cs
CustomerDTO.cs
Form1.cs
MyDBContext.cs
packages.config
Solution Explorer Team Explorer
Properties
CustAccountSimpleDTO.cs File Properti

9- Here we are applying join on two “entities” by using conventional approach. We’ve also selected data from two entities and generated objects will be of type “CustAccountSipmleDTO”.

```
1 reference
public List<CustAccountSimpleDTO> GetCustAccount_Query()
{
    using (var ctx = new MyDBContext())
    {
        var query = from p in ctx.Customers
                    from a in ctx.Accounts
                    where p.CustomerID == a.CustomerID
                    select new CustAccountSimpleDTO {
                        CustomerName  = p.Name,
                        AccountNumber = a.AccountNumber
                    };
        var result = query.ToList();
        return result;
    }
}
```

Here we hadn't used "join" clause

10- Following is another way to achieve above functionality

```
1 reference
public List<CustAccountSimpleDTO> GetCustAccount_Join_Query()
{
    using (var ctx = new MyDBContext())
    {
        var query = from p in ctx.Customers
                    join a in ctx.Accounts on p.CustomerID equals a.CustomerID
                    select new CustAccountSimpleDTO
                    {
                        CustomerName = p.Name,
                        AccountNumber = a.AccountNumber
                    };
        var result = query.ToList();
        return result;
    }
}
```

11- Now check following link again and go through different examples of “LINQ to Entities”

a. https://msdn.microsoft.com/en-us/library/bb399367(v=vs.100).aspx

12- Now let suppose, we've a stored procedure which returns result (Name & Account Number) like above example. Here is our SQL code to create required stored procedure.

SQLQuery3.sql - (lo...ORLD\BShahzad (52))* | SQLQuery2.sql - not connected* | SQLQuery1.sql - not connected*

```sql
CREATE Procedure dbo.GetCustomerAccounts
AS
BEGIN

    SELECT c.Name,a.AccountNumber
    FROM dbo.Customers c INNER JOIN dbo.Customer_Accounts a
    ON c.CustomerID = a.CustomerID

END
```

13- Here is code to execute above SP (or any query) using EF. We know that return value of above SP matches with our "CustAccountSipmleDTO" type so we can use this type for getting result from SP.

```
1 reference
public List<CustAccountSimpleDTO> GetCustAccount_SP()
{
    using (var ctx = new MyDBContext())
    {
        //This is our RAW query (which can be any query)
        var sqlQuery = "execute dbo.GetCustomerAccounts";
        var result = ctx.Database.SqlQuery<CustAccountSimpleDTO>(sqlQuery).ToList();
        return result;
    }
}
```