

# Entity Framework – Code First Approach – Part 2

This will show walkthrough of following concepts

- 1- How to delete/update entities without loading them first.
- 2- Relationship with Tables/Entities
- 3- Lazy Loading vs. Eager Loading vs. Explicit Loading

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

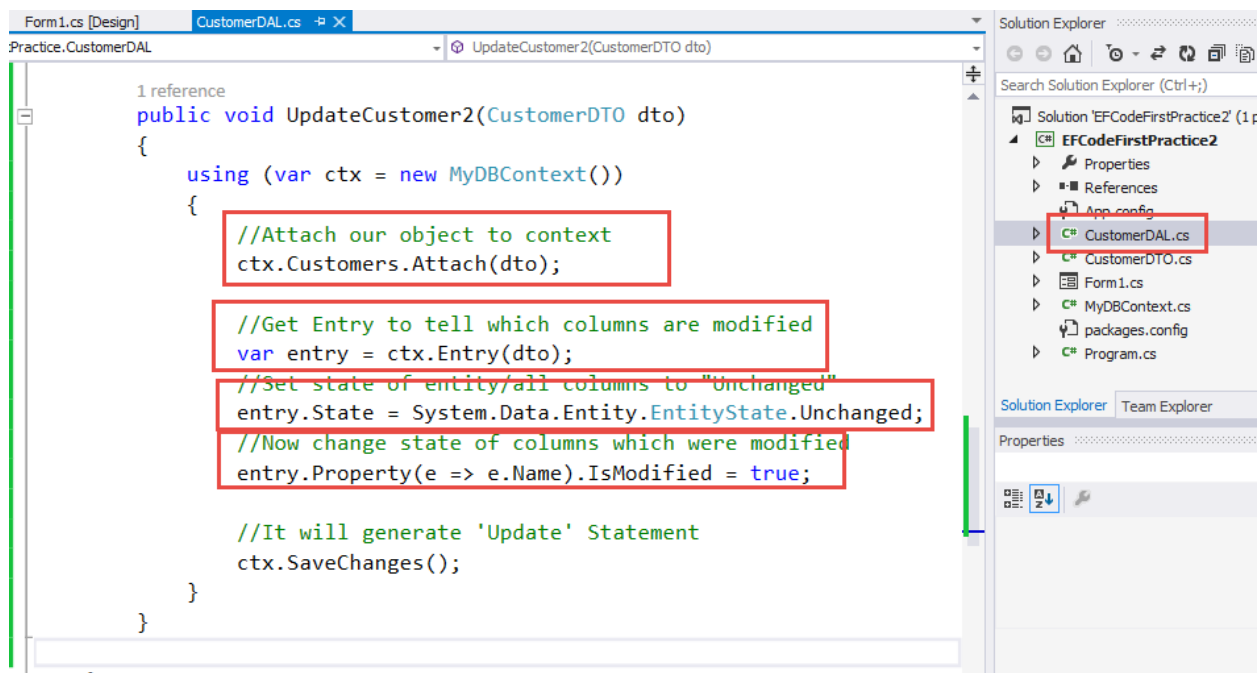
**Pre-requisite:** Entity Framework – Code First Approach – Part 1

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

## Step by Step Walkthrough

### How to delete/update entities without loading them first

- 1- As we've seen that to delete/update an entity in EF, that entity should be attached with context. In first part, we've loaded the entity from database first (as it will be attached with context in this way) and then we deleted/updated it. But we can also attach an unattached entity to context (without reading from database) and then can delete/update it.
- 2- Check this link first to understand "Entity States" <https://msdn.microsoft.com/en-us/data/jj592676.aspx>
- 3- Open "CustomerDAL.cs" file and add a new function in it. Here we are doing following task
  - a. Attaching an "Unattached" entity with our context. If you are going use "Entry" function, this attachment is optional. At this time, entity state will be "Unchanged".
  - b. Then we are going to get "Entry" object to set its state. This will automatically attach the entity if it is not attached already.
  - c. Then we can set different states of the entities. If more columns are changed (& less are unchanged), we can change the state of entity to "Modified" and then can set state of individual columns to "Unchanged". If more columns are unchanged (& less are changed), we can change the state of entity to "Unchanged" and then can set state of individual columns to "Modified".
  - d. Now when we'll call "SaveChanges()", only columns with "Modified" state will be sent in query.

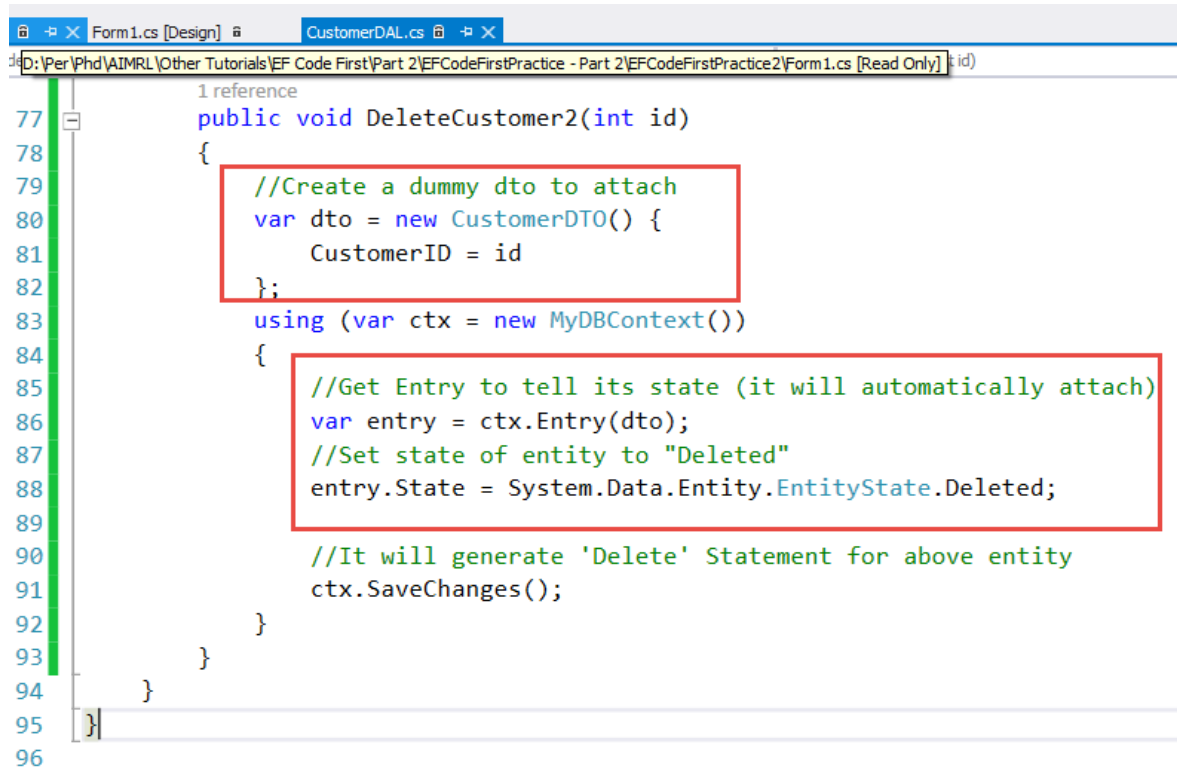


- 4- Now you can call "UpdateCustomer2" function in your form instead of "UpdateCustomer" function. Verify if update is working as expected.

```
CustomerDAL dal = new CustomerDAL();
if (id == 0) //Insert Case
{
    dal.SaveCustomer(dto);
}
else //update case
{
    dal.UpdateCustomer2(dto);
}

MessageBox.Show("Record is saved!");
}
```

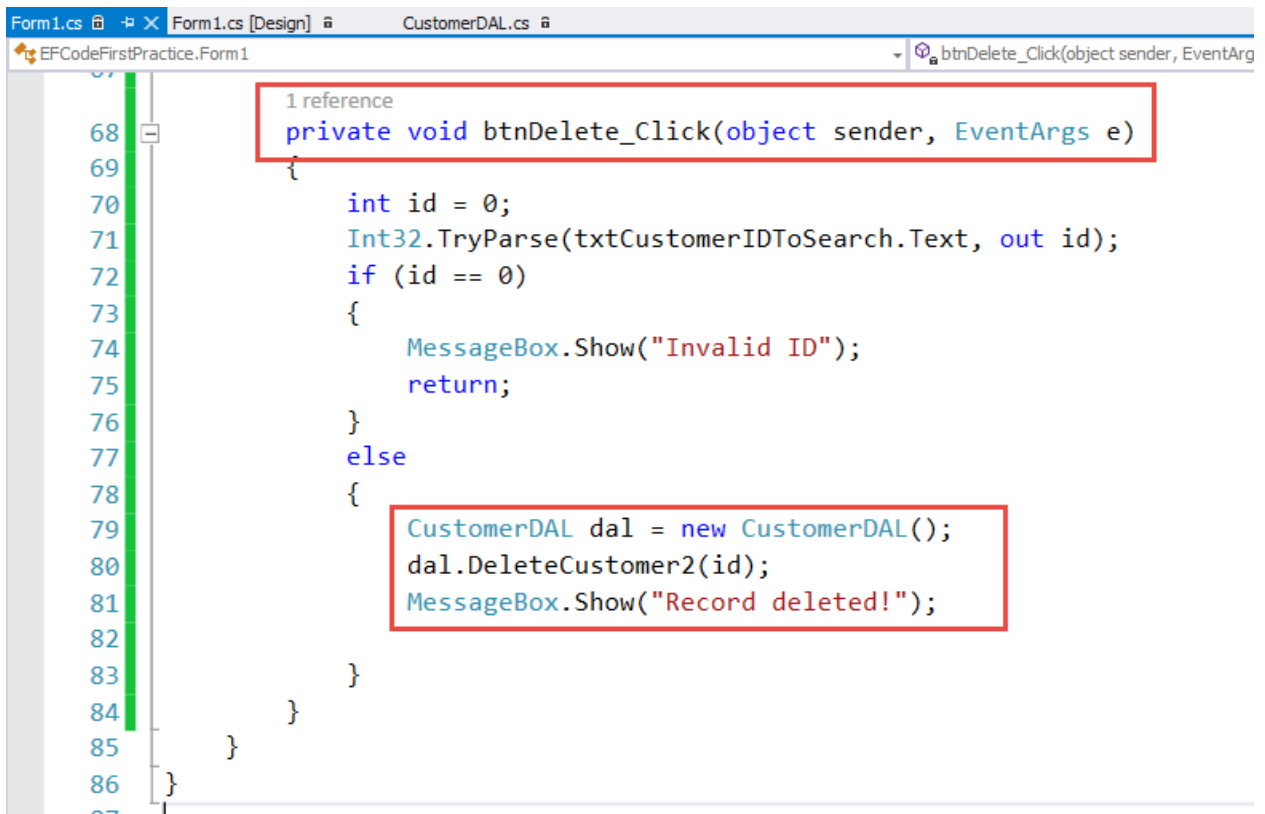
- 5- Now we can use same approach to delete a record. But in deletion we don't need to go at column level. Add following function in your "CustomerDAL" class.



```
1 reference
public void DeleteCustomer2(int id)
{
    //Create a dummy dto to attach
    var dto = new CustomerDTO() {
        CustomerID = id
    };
    using (var ctx = new MyDBContext())
    {
        //Get Entry to tell its state (it will automatically attach)
        var entry = ctx.Entry(dto);
        //Set state of entity to "Deleted"
        entry.State = System.Data.Entity.EntityState.Deleted;

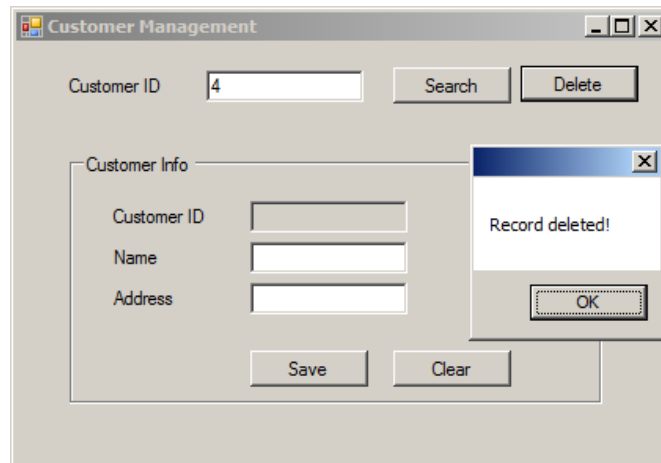
        //It will generate 'Delete' Statement for above entity
        ctx.SaveChanges();
    }
}
```

- 6- Add a button "Delete" on your form and write following code in its click event handler.



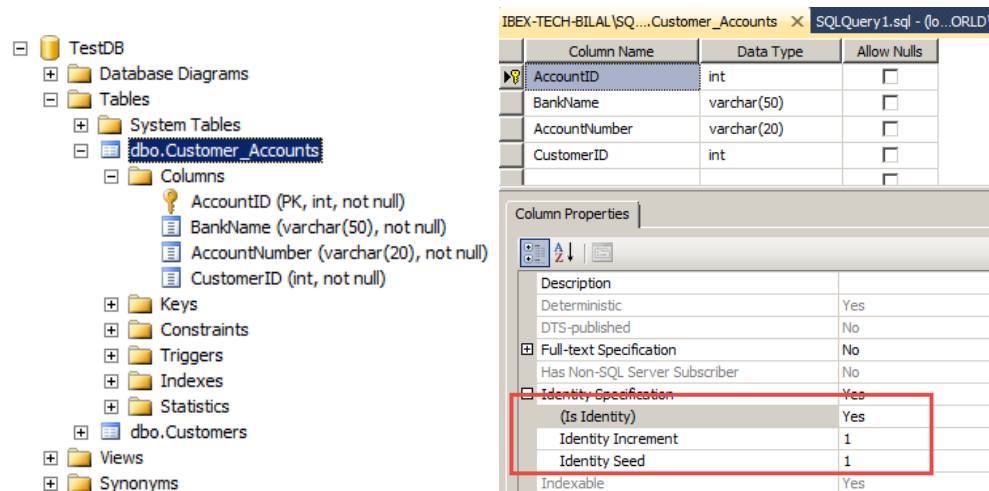
```
1 reference
private void btnDelete_Click(object sender, EventArgs e)
{
    int id = 0;
    Int32.TryParse(txtCustomerIDToSearch.Text, out id);
    if (id == 0)
    {
        MessageBox.Show("Invalid ID");
        return;
    }
    else
    {
        CustomerDAL dal = new CustomerDAL();
        dal.DeleteCustomer2(id);
        MessageBox.Show("Record deleted!");
    }
}
```

7- Run program, provide an ID and click “Delete”. Verify if this is working as expected.

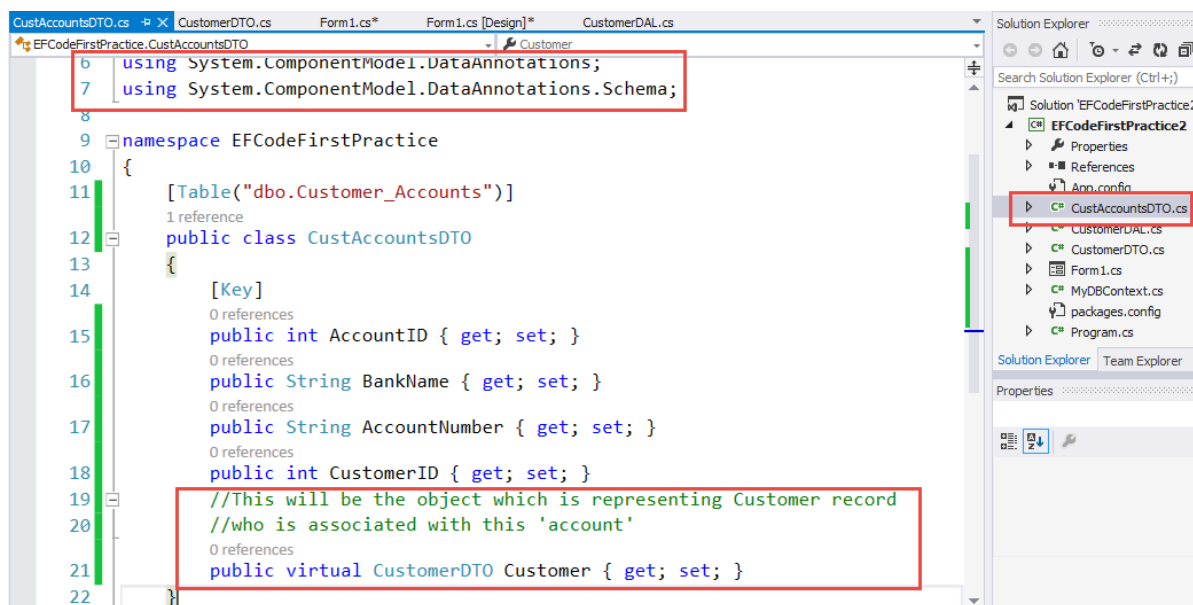


## Relationship with Tables/Entities

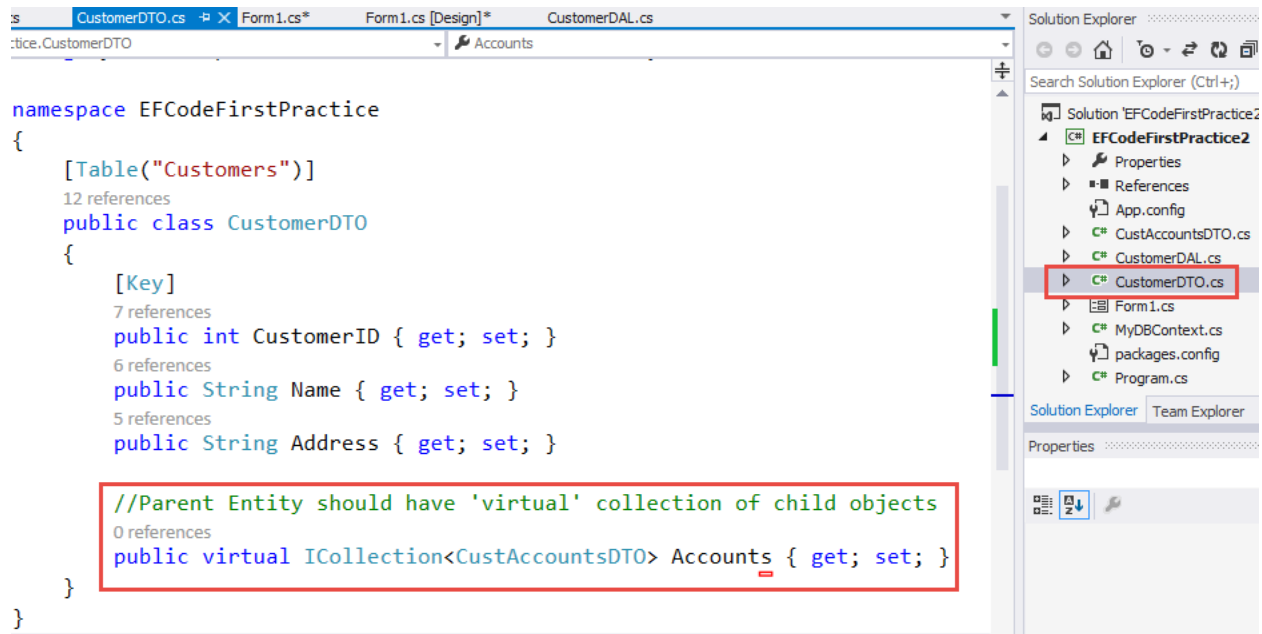
- 1- In Database, a relationship between tables is normal thing. To load related data using queries, we use JOINS. But when we talk about “Entities”, we create “Navigational Properties to represent relationship between entities. And when we load data of parent entity, we can use different approaches to load data of child entities.
  - a. Lazy Loading
  - b. Eager Loading
  - c. Explicit Loading
- 2- Check this link <https://msdn.microsoft.com/en-us/data/jj574232.aspx> first and then go through next steps.
- 3- Once you have absorbed the above link, now let’s create another table as shown below. This table is using “CustoemrID” as foreign key (although no physical relationship in database).



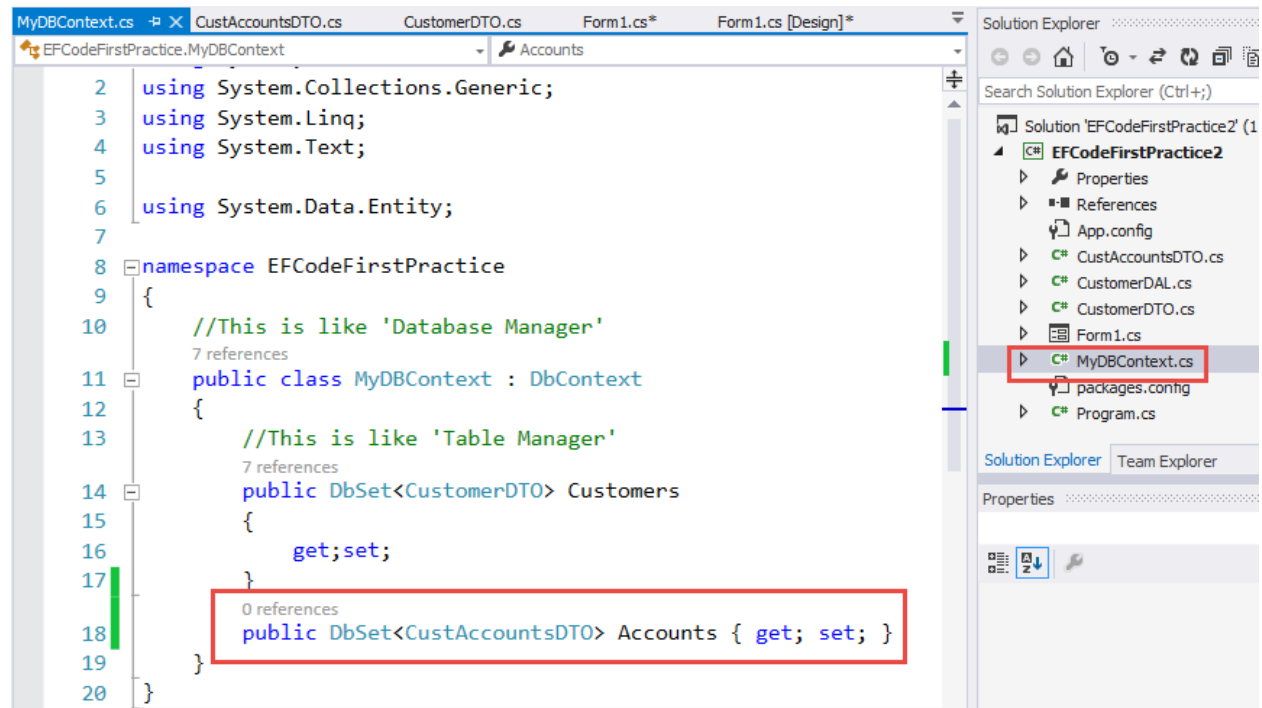
- 4- Add a new class file and add following code. Note that we have added an extra “Navigation” property to refer related “Parent” object.



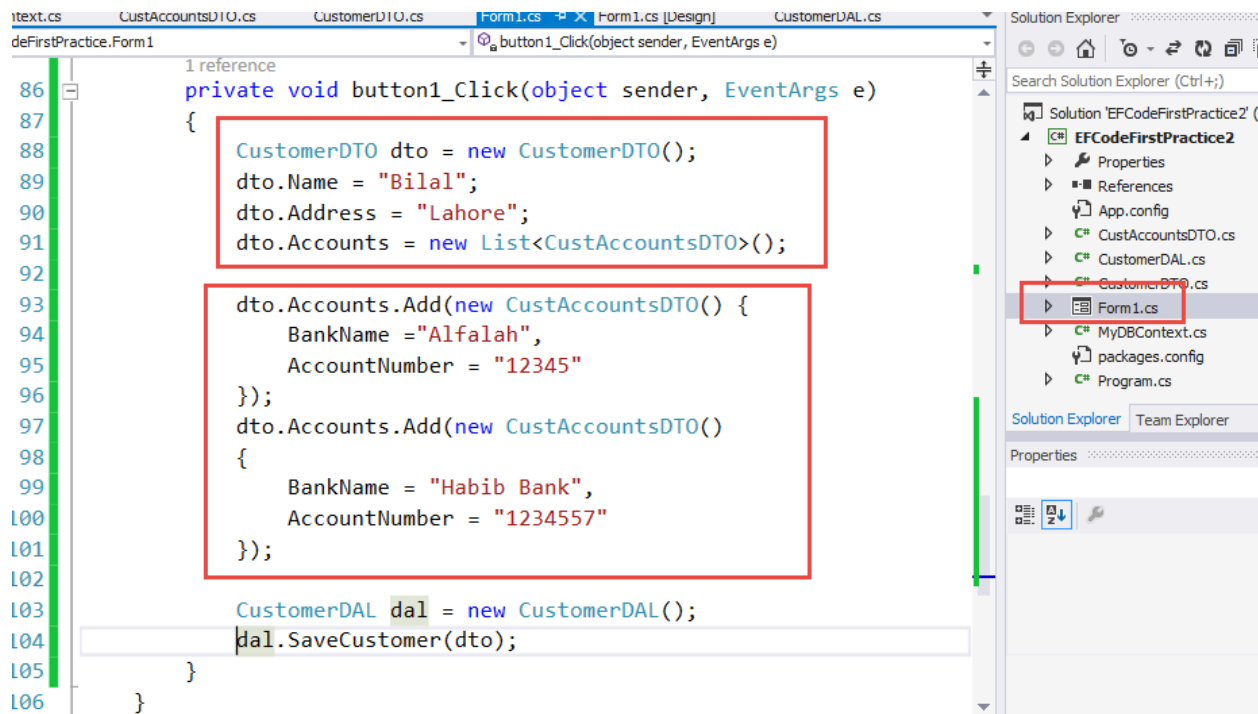
- 5- Open “CustomerDTO.cs” file and add a “Navigation” property to refer related “Child” objects.



- 6- Now add “DbSet” for our new entity in “MyDBContext” class as shown below.



- 7- Now for testing purposes, add a button in your form and then write following code in click event handler of that button. Here we are creating a dummy customer object and adding two accounts to its “accounts” collection. We hadn’t made any change in “SaveCustomer” function.

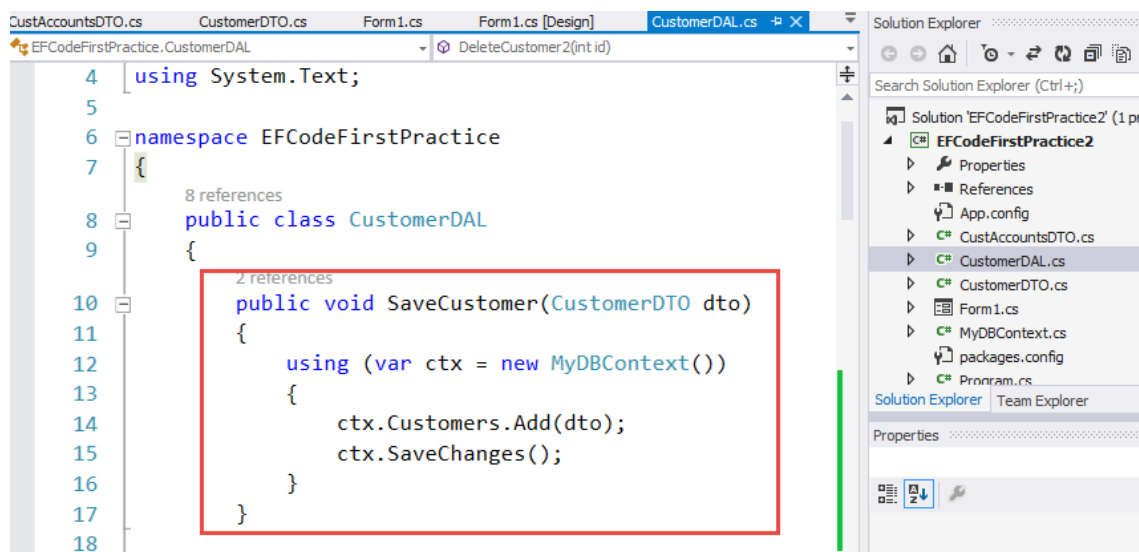


```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    CustomerDTO dto = new CustomerDTO();
    dto.Name = "Bilal";
    dto.Address = "Lahore";
    dto.Accounts = new List<CustAccountsDTO>();

    dto.Accounts.Add(new CustAccountsDTO() {
        BankName = "Alfalah",
        AccountNumber = "12345"
    });
    dto.Accounts.Add(new CustAccountsDTO()
    {
        BankName = "Habib Bank",
        AccountNumber = "1234557"
    });

    CustomerDAL dal = new CustomerDAL();
    dal.SaveCustomer(dto);
}
```

- 8- This is “SaveCustomer()” function where we are providing customer object to save. This was written in Part 1. Here context does many things for us. It will insert “customer” object first, will get “auto” generated customer id and will use that id while inserting all “child” objects (in this case accounts) into database. And it does all that in one transaction.



```
using System.Text;

namespace EFCODEFirstPractice
{
    8 references
    public class CustomerDAL
    {
        2 references
        public void SaveCustomer(CustomerDTO dto)
        {
            using (var ctx = new MyDBContext())
            {
                ctx.Customers.Add(dto);
                ctx.SaveChanges();
            }
        }
    }
}
```

9- Here we can run queries in SQL server to verify the changes we've made in above step.

```
SQLQuery1.sql - (lo...ORLD\BShahzad (52))* X
```

```
use TestDB
```

```
Select * from dbo.Customers
```

```
select * from dbo.Customer Accounts
```

133 %

CustomerID	Name	Address
5	Bilal	Lahore

AccountID	BankName	AccountNumber	CustomerID
1	Alfalah	12345	5
2	Habib Bank	1234557	5

10- Now we've already seen following code to read all customers from database. In this code, we are not mentioning anything about "child" (related) objects at all. By default, child objects are not loaded until they are accessed in code. When we'll access "Accounts" property in any customer object, hit will be make to server to load relevant accounts against that customer. This is called "Lazy or Differed" loading. Also here we can see that "ctx" will be disposed at end of "using" block.

```
CustomerDTO.cs  Form1.cs  Form1.cs [Design]  CustomerDAL.cs X
```

```
.CustomerDAL
```

```
8 references
```

```
public class CustomerDAL
```

```
{
```

```
    2 references
```

```
    public void SaveCustomer(CustomerDTO dto)...
```

```
    0 references
```

```
    public List<CustomerDTO> GetAllCustomers()
```

```
    {
```

```
        using (var ctx = new MyDBContext())
```

```
        {
```

```
            var list = ctx.Customers.ToList();
```

```
            return list;
```

```
        }
```

```
    }
```

```
    1 reference
```

```
    public CustomerDTO GetCustomerByID(int id)...
```

```
    0 references
```

```
    public void DeleteCustomer(int id)...
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'EFCodeFirstPractice2' (1 p

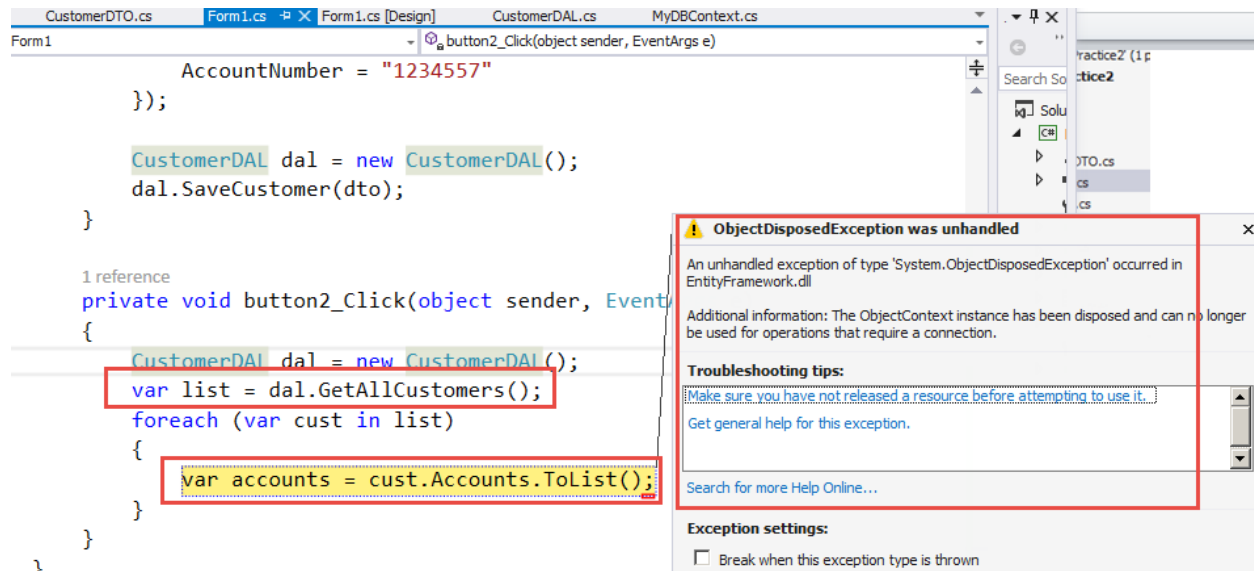
- EFCodeFirstPractice2
- Properties
- References
- App.config
- C# CustAccountsDTO.cs
- C# CustomerDAL.cs
- C# CustomerDTO.cs
- Form1.cs
- C# MyDBContext.cs
- packages.config

Solution Explorer Team Explorer

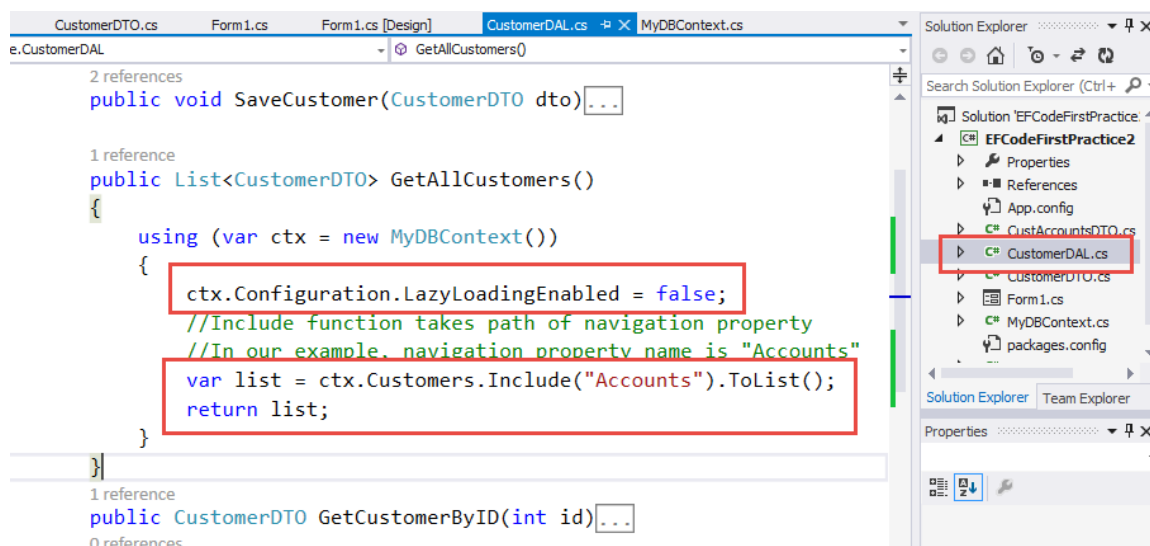
Properties



- 11- Now add another button in your form for testing purpose and add following code in click event handler of that button. Here we are reading all customers and then iterating the list. But we've seen in above code that as soon "GetAllCustomers" function is completed, "ctx" was disposed so the connection was closed. Now when we'll try to access the child objects (which are still not loaded), it tries to find the "context" object. Therefore, when you will run the project and clicks the button, you will see "error" while accessing the child objects.



- 12- Now to fix above issue, there can be multiple ways
- a) Don't expose the context object (if you want to load related objects through lazy loading approach)
  - b) Access the child objects (while getting the parent objects) within the context so child will be loaded in that function.
  - c) Load the child objects using "Include" function (Eager Loading)
  - d) Disable "Lazy Loading" and load the child Objects using "Explicit" loading approach.
- 13- Here we can see how to disable "Lazy Loading" and using "Include" function to load child elements with parent elements. This will load customer object (& all child elements linked with that customer).

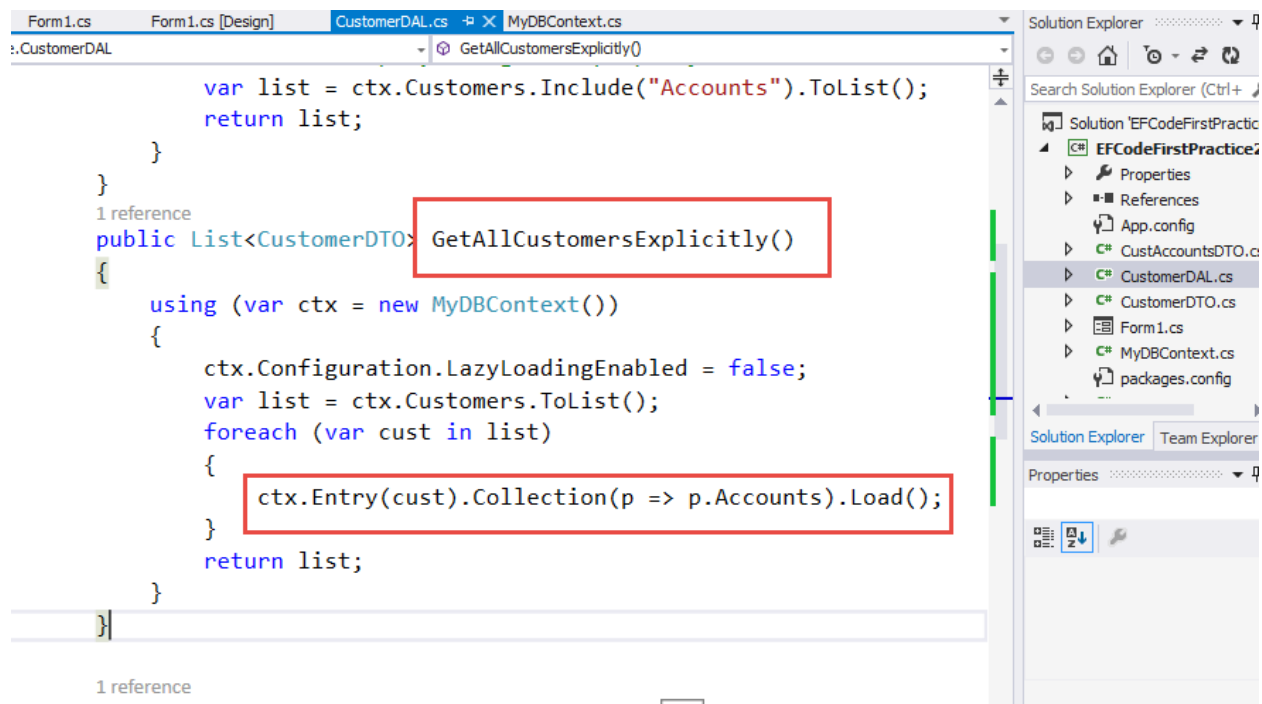


14- Now run the code and verify the execution.

15- The disadvantage of “Include” function is that it brings the data in de-normalized fashion. Here is the result which will be sent from SQL Server to EF after query execution. Data will come in this format and then EF will generate customer object and will load accounts data. Now if there are many customers and each customer is having many accounts, we can see “repetition” of data.

	CustomerID	Name	Address	C1	AccountID	BankName	AccountNumber	CustomerID1
1	5	Bilal	Lahore	1	1	Alfalah	12345	5
2	5	Bilal	Lahore	1	2	Habib Bank	1234557	5

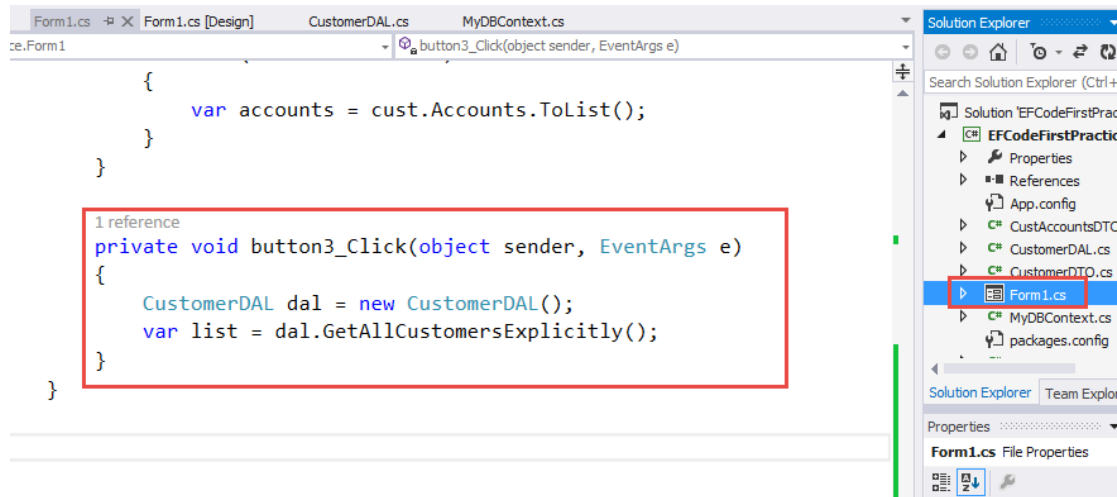
16- Explicit Loading: Even with lazy loading disabled it is still possible to lazily load related entities, but it must be done with an explicit call. To do so you use the Load method on the related entity’s entry. Add a new function in “CustomerDAL” and add the code as shown in following screenshot. Here we are loading all customers and then during iteration, we are “explicitly” generating request to load relevant child objects (i.e. Accounts).



```
Form1.cs  Form1.cs [Design]  CustomerDAL.cs  MyDBContext.cs
:CustomerDAL  GetCustomersExplicitly()

var list = ctx.Customers.Include("Accounts").ToList();
return list;
}
}
1 reference
public List<CustomerDTO> GetAllCustomersExplicitly()
{
    using (var ctx = new MyDBContext())
    {
        ctx.Configuration.LazyLoadingEnabled = false;
        var list = ctx.Customers.ToList();
        foreach (var cust in list)
        {
            ctx.Entry(cust).Collection(p => p.Accounts).Load();
        }
        return list;
    }
}
}
1 reference
```

17- Now add another button in your form for testing purpose and add following code in click event handler of that button.



18- Now run the code and verify the execution of this new button.

## Entity Framework (EF) Documentation

<https://msdn.microsoft.com/en-us/data/ee712907>