Lab week 10 - Entity Framework - Code First (part 1) annotations

In this lab we are going to look at Code First Entity Framework. This material is directly related to the Lecture material and you should use this is a guide to complete the work.

The following classes are simplified in that they have properties, and the properties all public and are exposed with set and set. In a real situation we would wish to have protection built into the setters so that the object cannot be placed into a state which is inconsistent with their contract. ORMs are ideally used with enterprise level applications, and are often used with another layer in the application to encompass the business logic.

Part 1 (part 2) we will configure this to work with the inbuilt database all insert one record into the DB. There is Part 2 which focues on Database first and LINQ. Please to all of these in the correct sequence.

At the end of the exercise there is some further reading for those who wish to read more regarding best practice in using DbContext. The default use of DbContext does seem to be quite limiting therefore I would suggest that you do read these articles as they provide some insight on the strategies currently being used in industry to make DbContext more useful.

Your Task

**Step one** - create a new project, it does not matter if this is a GUI, Empty or Console based as we are not planning on adding any business logic or user interface. The whole purpose of today is to look at Entity Framework code first.

**Step two** - Once you have successfully created your project, you need to use NuGet to install the latest version of Entity Framework.

**Step three** - add the following classes to the project.

public class Student

{

public string Name { get; set; }

public DateTime DOB { get; set; }

public string Email { get; set; }

public Address Address { get; set; }

public List<Module> Modules { get; set; }

public List <Assessment> Assessments { get; set; }

}

public class Module

{

public String ModuleCode { get; set; }

public String ModuleName { get; set; }

public int Level { get; set; }

public List <Student> Students { get; set; }

public List <Assessment> Assessment { get; set; }

}

public class Assessment

{

public int Mark { get; set; }

public DateTime Submitted { get; set; }

public virtual Student Student { get; set; }

public virtual Module Module { get; set; }

}

public class Address

{

public String NumbnerOrName { get; set; }

public String Street { get; set; }

public String Town { get; set; }

public String PostCode { get; set; }

}

**Step four**

The code above is based on very simple entity classes, as we are trying to make this a simple as possible they lack the normal levels of protection we would normally wish to put into our classes. There is another reason why we may not wish to encompass too much business logic into these classes as it is normal practice on enterprise application to add in another layer into our application called the services layer which encompasses the business logic. If you wish to know more about this then see http://martinfowler.com/eaaCatalog/serviceLayer.html .

Although they are simple entity classes and are close to being usable with Entity framework they do lack some additional characteristics needed for Entity Framework.

**Your task is to try and identify what is missing, please do not go to the next page until you have at least tried to add to these classes.** Look at the lecture and try and align these classes with what is needed for entity framework code first.

Next page contains spoilers do not read it until you have attempted the work above!

Step four - what you should have identified.

a) Entity framework requires that each of the classes has a unique key value. You need to add one to each of the classes.

b) Although Entity Framework will work without using Annotations, it is better to use them than let the default settings configure our system. see https://msdn.microsoft.com/en-us/library/jj591583(v=vs.113).aspx for a full list. The ones we are interest in are the [key], and [MaxLength]. You will need to add the correct using to make the annotations work.

c) Any property which is a foreign key type mapping needs to be amended so that it supports lazy and tracking, this is done by making it virtual. If the property contains data collection, then you need to change it be being ICollections rather than a collection such as List<T>.

d) Perhaps the simplest change needed but you need to make sure each has a constructor and that you initialise the collection.

**Step Five**

Create a DBContext for the system call it whatever you like however if you cannot think of anything call it MyDBEntities.

There is a one-to-many relationship in the classes, between module and student. Since we do not wish to worry about creating our own intersection table we are going to allow Entity Framework to do this step for us, we cannot use data annotations for this, however the fluent API is able to create the intersection table. See the slides to identify the correct code to so this.

**Finished code**

public class Address

{

[Key]

public int AddressID { get; set; }

public String NumbnerOrName { get; set; }

public String Street { get; set; }

public String Town { get; set; }

public String PostCode { get; set; }

}

public class Assessment

{

Assessment()

{

}

[Key]

public int AssementID { get; set; }

[Required]

public int Mark { get; set; }

[Required]

public DateTime Submitted { get; set; }

[Required]

public virtual Student Student { get; set; }

[Required]

public virtual Module Module { get; set; }

}

public class Module

{

public Module()

{

Students = new List<Student>();

Assessment = new List<Assessment>();

}

[Key]

public int ModuleID { get; set; }

[Required,MaxLength(20),MinLength(2)]

public String ModuleCode { get; set; }

[Required,MaxLength(40)]

public String ModuleName { get; set; }

[Required]

public int Level { get; set; }

public virtual ICollection<Student> Students { get; set; }

public virtual ICollection<Assessment> Assessment { get; set; }

}

public class Student

{

public Student()

{

Modules = new List<Module>();

Assessments = new List<Assessment>();

}

[Key]

public int SudentID { get; set; }

[MinLength(10), MaxLength(30)]

public string Name { get; set; }

[Required]

public DateTime DOB { get; set; }

public string Email { get; set; }

public virtual Address address { get; set; }

public virtual ICollection<Module> Modules { get; set; }

public virtual ICollection<Assessment> Assessments { get; set; }

}

public partial class MyDBEntities : DbContext

{

public MyDBEntities() : base("conString")

{

}

public DbSet<Student> Students {get; set;}

public DbSet<Module> Modules {get; set;}

public DbSet<Address> Addresses {get; set;}

public DbSet<Assessment> Assessments { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

}

}

**Reading**

If you remember form the lecture the examples are to use the following code

**using (var context = new MyDBContext()) { code selecting and updating entities }**

It uses ‘using’ as it automatically disposes the context at the end of the block, this is also recommended as it helps to isolate transactions, also be aware that DbContext is not thread safe and therefore and instance should never be shared in the application. i.e. you would never use a singleton to gain access to an instance of DbContext. The problem with the code above is that it lives for a very short period of time and therefore there is little it can do such as caching or tracking changes. There are other approaches available such as using a unit of work and repository class which wraps the DbContext, note that disposing of the DbContext at the end of the session needs to be considered.

A good article covering this can be found at

<https://www.asp.net/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application>

Additionally, a very good article regarding best practice in using DbContext can be found here.

<http://mehdi.me/tag/entity-framework/>

Other good articles which I do suggest you read at some point.

<http://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/>

<http://codereview.stackexchange.com/questions/47879/unit-of-work-and-repository-with-entity-framework-6>

<http://stackoverflow.com/questions/16064902/dependency-injection-in-unit-of-work-pattern-using-repositories>