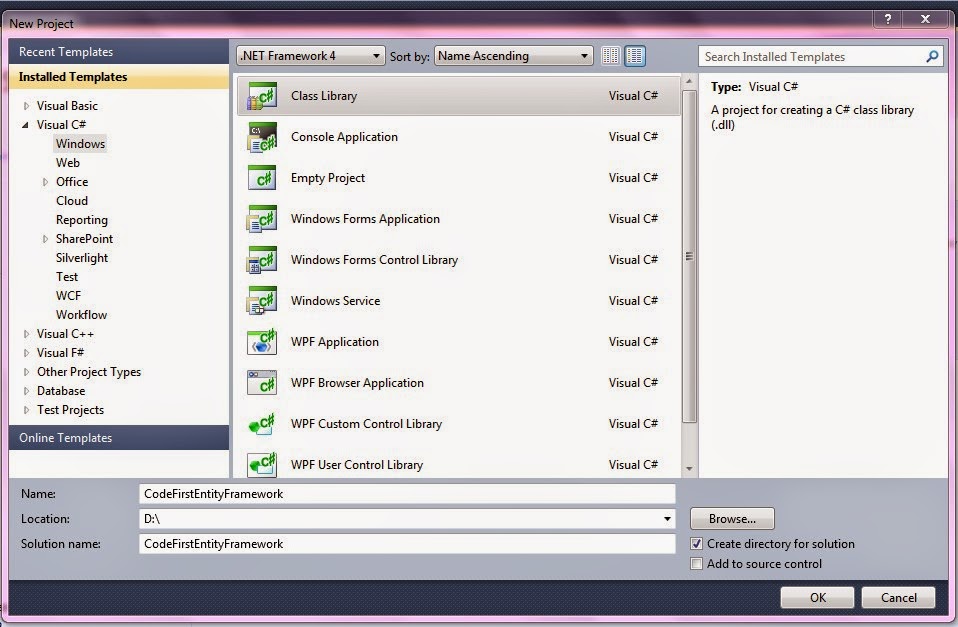
Code First Approach using Entity Framework 4.0 Sample example using ASP.NET,C#.NET

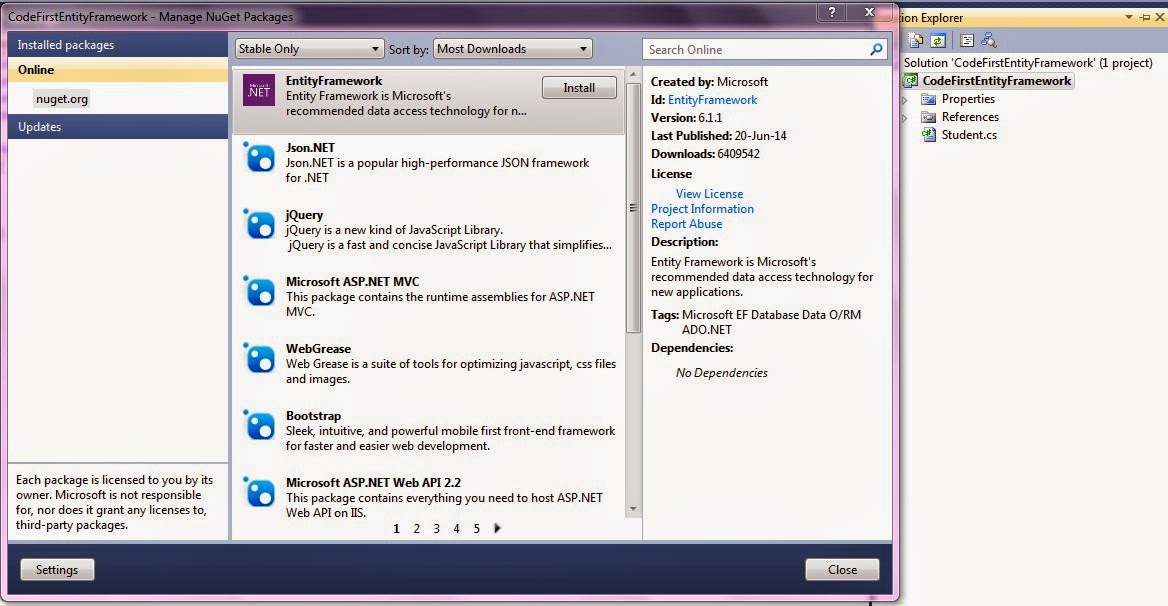
Hi friends,in this article I will explain about Entity Framework Code First example  using ASP.NET,C#.NET.  
I already explained in the previous articles about [Telerik RadGrid Grouping -Drag and Drop a column header to group by that column in ASP.NET using C#/VB.NET](http://aspdotnet-kishore.blogspot.in/2014/07/telerik-radgrid-grouping-drag-and-drop.html), [How to show validation control's Error messages in Alert box in ASP.NET](http://aspdotnet-kishore.blogspot.in/2014/02/how-to-show-validation-controls-error.html) and [How to Upload Multiple Files Using FileUpload Control in ASP.NET using C#/VB.NET](http://aspdotnet-kishore.blogspot.in/2014/06/upload-multiple-files-using-fileupload.html)

**Code First:** In the Code First approach, you avoid working with visual model designer (EDMX) completely. You write your POCO classes first and then create database from these POCO classes. Developers who follow the path of Domain-Driven Design (DDD) principles prefer to begin by coding their classes first and then generating the database required to persist their data.

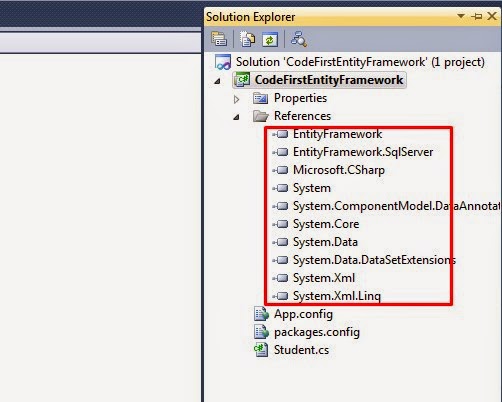
Create the Class Library project in Visual Studio 2010 as below.

[](http://4.bp.blogspot.com/-BNhxX1c0KfM/U_664AhkUKI/AAAAAAAABec/nLHKLsfM8Wo/s1600/Code+First+entity+framework.jpg)

Install Entity Framework as shown in the below figure.



After that below references will be added

[](http://2.bp.blogspot.com/-mXWGaY6_Mhs/U_67DyTNiFI/AAAAAAAABes/glmw8a9aiD4/s1600/Code+First+entity+framework3.jpg)

Create Student entity classes as below (You can use Entity Framework 4.1/4.3/5.0 for this example.):

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.ComponentModel.DataAnnotations;  using System.ComponentModel.DataAnnotations.Schema;  namespace CodeFirstEntityFramework  {      [Table("Student")]      public class Student      {          [Key]          public int StudentID { get; set; }          [Column("Gender")]          [MaxLength(50)]          public string Gender { get; set; }      }  } |

Add your connectionstring in App.config like below

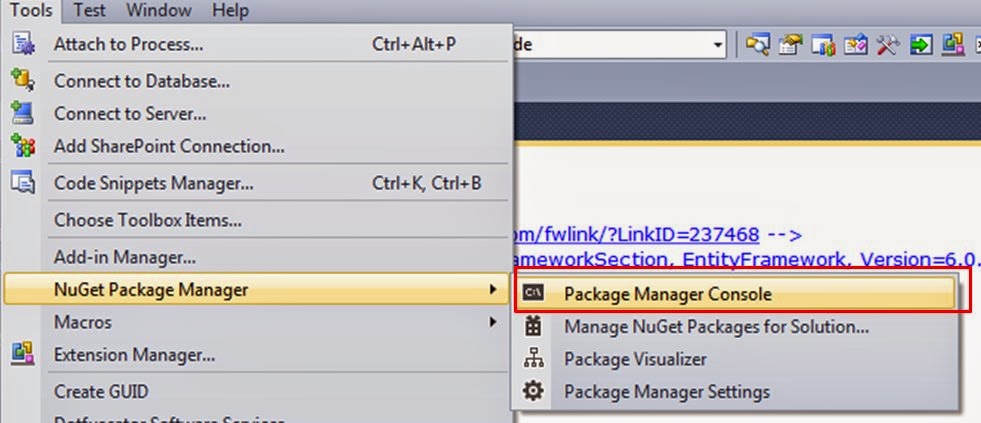
|  |
| --- |
| <connectionStrings>                                  <add name="Student\_DBConnectionString"          connectionString="Data Source=Test;Initial Catalog=Student\_EntityFramework;User Id=sa;password=12345;Max Pool Size=75000"          providerName="System.Data.SqlClient"/>                  </connectionStrings> |

Add Context.cs class and write the following code

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Data.Entity;  namespace CodeFirstEntityFramework  {      public class Context : DbContext      {          public Context()              : base("Student\_DBConnectionString")          {          }          public DbSet<Student> Student\_DbSet { get; set; }      }  } |

Open Package Manager Console

Tools---> Nuget Package Manager-- -> Package Manager Console

[](http://3.bp.blogspot.com/--301qsPA0zk/U_67YqpwppI/AAAAAAAABe0/I_FY2wpv0ms/s1600/Code+First+entity+framework4.jpg)

Now we need to run some commands in the**Package Manager console**

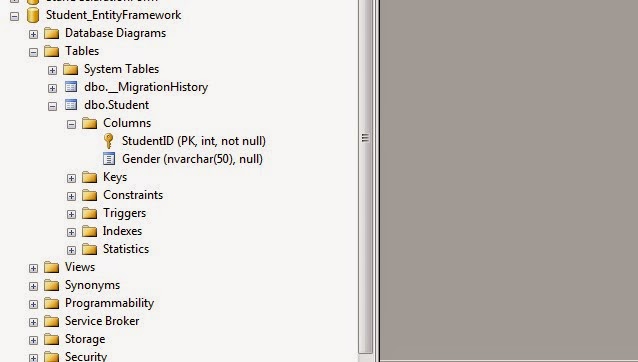
To go to Package Manager Console Go to **Tools >> Library package Manager** and click on Package Manager Console which shows a console.

Here we have to type **“Enable-Migrations”**which creates the Migrations folder and Metadata of the database we are going to create.

Once we enable the Migrations run the following command **“Add-Migration Initial”**

Once the above step is done we have to run the following command which completes the database creation **“Update-Database”**

Then your database will be created with Student Table in the database.See in the below figure.

[](http://4.bp.blogspot.com/-toO5b8qX5cw/U_67cuJQFoI/AAAAAAAABe8/qxR39WdVPg4/s1600/Code+First+entity+framework5.jpg)

I think the differences are:

**Code first**

* Very popular because hardcore programmers don't like any kind of designers and defining mapping in EDMX xml is too complex.
* Full control over the code (no autogenerated code which is hard to modify).
* General expectation is that you do not bother with DB. DB is just a storage with no logic. EF will handle creation and you don't want to know how it do the job.
* Manual changes to database will be most probably lost because your code defines the database.

**Database first**

* Very popular if you have DB designed by DBAs, developed separately or if you have existing DB.
* You will let EF create entities for you and after modification of mapping you will generate POCO (Plain old CLR objects) entities.
* If you want additional features in POCO entities you must either T4 modify template or use partial classes.
* Manual changes to the database are possible because the database defines your domain model. You can always update model from database (this feature works quite good).
* I often use this together VS Database projects (only Premium and Ultimate version).

**Model first**

* IMHO popular if you are designer fan (= you don't like writing code or SQL).
* You will "draw" your model and let workflow to generate your database script and T4 template to generate yout POCO entities. You will lose part of control on both your entities and database but for small easy projects you will be very productive.
* If you want additional features in POCO entities you must either T4 modify template or use partial classes.
* Manual changes to database will be most probably lost because your model defines the database. This works better if you have Database generation power pack installed. It will allow you updating database schema (instead of recreating) or updating database projects in VS.

If you are building an MVC3 application, I believe Code first has the following advantages:

* **Easy attribute decoration** - You can decorate fields with validation, require, etc.. attributes, it's quite awkward with EF modelling
* **No weird modelling errors** - EF modelling often has weird errors, such as when you try to rename an association property, it needs to match the underlying meta-data - very inflexible.
* **Not awkward to merge** - When using code version control tools such as mercurial, merging .edmx files is a pain. You're a programmer used to C#, and there you are merging a .edmx. Not so with code-first.
* Contrast back to Code first and you have complete control without all the hidden complexities and unknowns to deal with.
* I recommend you use the Package Manager command line tool, don't even use the graphical tools to add a new controller to scaffold views.
* **DB-Migrations** - Then you can also Enable-Migrations. This is so powerful. You make changes to your model in code, and then the framework can keep track of schema changes, so you can seamlessly deploy upgrades, with schema versions automatically upgraded (and downgraded if required). (Not sure, but this probably does work with model-first too)

Earlier this month we discussed the advantages of implementing the [Repository pattern](http://www.incredible-web.com/blog/repository-pattern/) within your architecture. In this post we will take things a step further down our Multi-Tier Architectiure and discuss the Data Store within our Data Access Layer. We will take a look at the two most common approaches used in creating an [Entity Framework](http://www.incredible-web.com/method/entity-framework/) conceptual model on top of your relational Database, Code First and Model first.

To start off it is worth mentioning that the [Entity Framework 4.0](http://www.incredible-web.com/method/entity-framework/) actually provides three distinctive development approaches when it comes to data access within your web site:

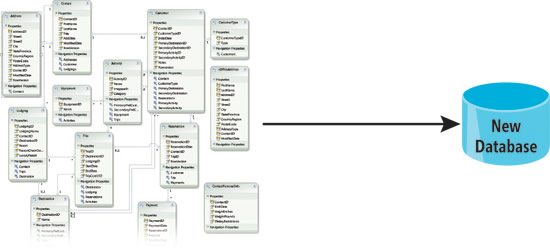
* Model First
* Code First
* Database First

In this article we'll only tackle the two most recent approaches, Code First and  Model First, the main reason being the fact that the Database First approach  has been used for a very long time and most web developers can easily point out the advantages and disadvantages of this approach.

Model First

The Model First Approach was provided by Microsoft because it was one of the most often requested features by .Net Developers and System Designers. It is a more natural way to work compared to the Database First approach especially when designing the flow of data in the initial stages of a new project. Developers requested the flexibilty of creating the conceptual model first instead of going through the burden of outlining the project logic and then trying to design a database which would accomodate for all the data and information storage within the project. Using the Model First approach a developer can start working with the model of the databse and creating entities which make logical sense irrelevant of how they will actually be stored in the database in terms of tables.

From the model created, Visual Studio (2010 onwards) can then generate SQL statements referred to as *Data Defenition Language (DDL)* which the developer can use to execute on MS SQL in order to create the Database schema based on the designed model.



It is vital to point out that the Model First approach does not by deafult perform incremental schema updates on your database, so if the model is changed or updated for some reason, a new script must be generated which will overwrite the existing database schema. This will obviously result in a loss of data, however there is nothing to worry about as there are a number of datbase tools available which will help you update your database schema without compromising your data such as the [*Entity Designer Database Generation Power Pack from Microsoft*](http://visualstudiogallery.msdn.microsoft.com/df3541c3-d833-4b65-b942-989e7ec74c87)*.*

Code First

The Code first approach is the most recent approach provided by Microsoft as part of the [Entity Framework 4.0](http://www.incredible-web.com/method/entity-framework/). Its a more developer oriented approach where a developer can literally code the entities required within the system as if they were regular classes and objects and then create the database schema from these classes by defining special relationships and mappings within these entities. The main advantages in utilizing the Code First  approach is that the developer has complete control on the relations between the entities (somthing which is not utterly provided by the Model first approach because of automatically generated code) and in the mean time still use an in-memory model that the EF runtime can work with. Using the code first approach the developer can also provide restrictions and annotations on specific properties in the domain models (classes) which will serve as validations both on the database and also within the code itself.

The example below shows a typical domain model with data annotaions on properties and some navigation properties to other domain models:

[Table("SalesOrderDetail", SchemaName="SalesLT")]

public partial class Detail

{

// Scalar properties

[Column(Name = "SalesOrderID")]

public int OrderId { get; set; }

[Column(Name = "SalesOrderDetailID")]

public int DetailId { get; set; }

public short OrderQty { get; set; }

public int ProductId { get; set; }

public decimal UnitPrice { get; set; }

public decimal UnitPriceDiscount { get; set; }

public decimal LineTotal { get; set; }

public System.DateTime ModifiedDate { get; set; }

// Navigation properties

public virtual Product Product { get; set; }

public virtual Order Order { get; set; }

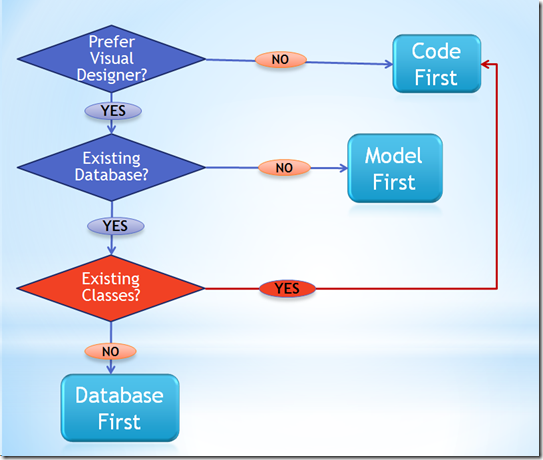
}

The 'DbContext' class is then used to specify how the classes will be created and translated into database tables and including the relationships between the data entities.

How to choose which approach to take?

It all depends on the specific web development project requirements and the set of skills you've got at your disposal within your team really. System Designers tend to prefer the Model First approach because it allows you to take into perspective the whole system and the relations amongst all the models and entities within the system without having to deal with how the data will eventually be stored or the technicalities of data annotations within the code. On the otherhand developers will justly argue that Code First gives you more control on the relations and specifications of your domain model without compromising the advantages of Conceptual Mapping within your system.

A very good decision tree I came across when I was first trying to figure out which approach suits us better at Incredible Web for certain projects is the diagram below:



Before commiting to a particular workflow a developer/system designer must go through the mentioned decisions within this decision tree. Hopefully it will make it easier for you to choose the right approach, I know it worked quite well for us.