**Code First Approach**

Introduction

The Code First approach is the second of two approaches used in developing ASP.NET MVC Web Applications. The first being; Database First.

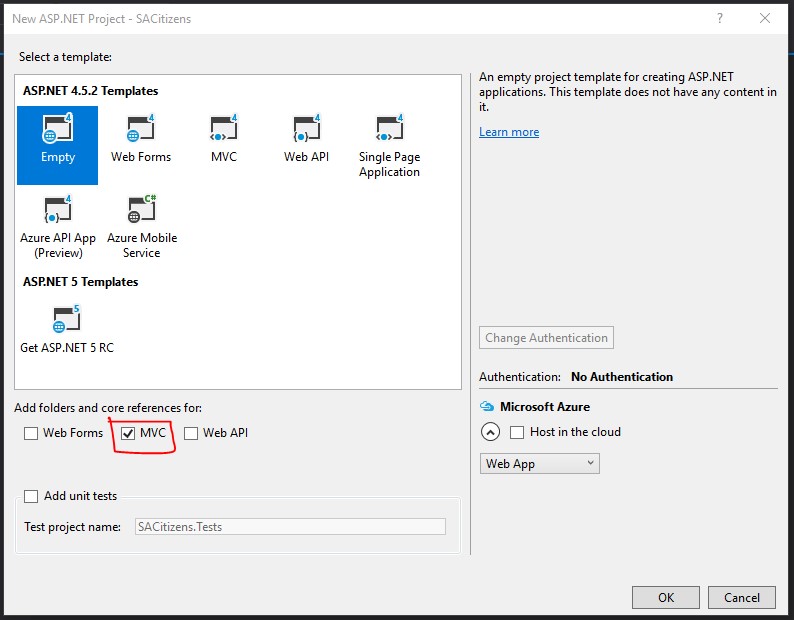
While Database First focuses on creating an external database object and then generating the model class from that, Code First allows us to write the model first and then a database object is generated from the model.

That’s the simplest way I can put it.

This guide will run through a working application from start to finish. It will show you how to create a Model class, database class and how to connect these to your application.

Create an ASP.NET Web Application project called *SACitizens*

* Make sure you choose an empty template
* Then Add Folders and Core References for MVC

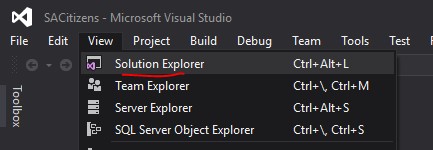


You’ve chosen an empty template so the project will be empty however all folders needed for MVC will be created for you. Folders for Models, Controllers ad Views.

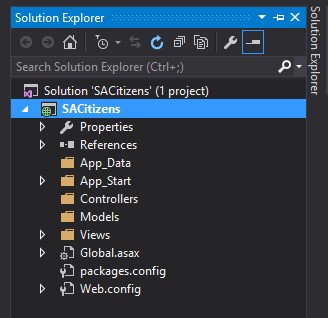
* Click OK on the bottom right 😊

When your project is created – You will see all folders you need in your Solution Explorer. This is usually found on the top right.

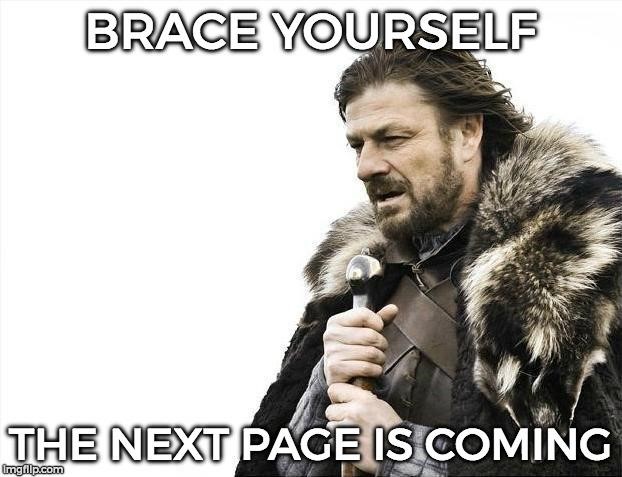
If you cannot see your Solution Explorer, you will need to click on View (Task Bar) and then click on Solution Explorer -



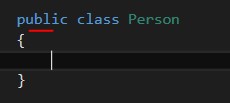
After you’ve completed this step, your Solution Explorer should appear on the far right –

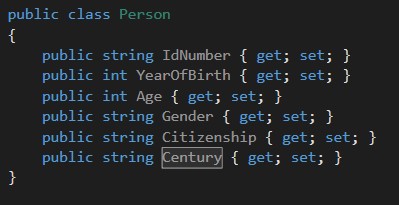


Now we can go ahead in creating the rest of the application –



Now in your Solution Explorer – Right click on the Models folder and add a class. Name this class *Person* and click Add. This will be your model.

Inside your model, make sure that the class is decorated with the public attribute so that its properties and methods can be accessed throughout the entire application.

Now we are going to be adding some properties into the model. The properties will be –

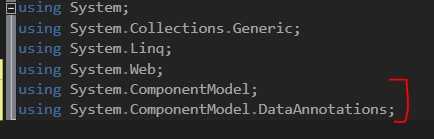
* IdNumber (string)
* Year of Birth (int)
* Age (int)
* Gender (string)
* Citizenship (string)
* Century (string)

Now we need to add some validation into the model. I will limit the validation for the purposes of this guide, however you should practice strengthening the validity of your models so that you can build more secure applications.

Before that though, we need to add some namespaces so that we can implement validation.

At the top of your model, under the last using statement, type the following –

* using System.ComponentModel;
* using System.ComponentModel.DataAnnotations;



You have used these statements before so there’s no need to go over them at this point 😉

Let’s go about adding the necessary validation –

[

Key

]

is used to denote

that the following

property will be a primary

key.

[

Displa

y

(

Name =

""

)]

is

used to change the way

the property is displayed

to the user.

Now we will need to add

4

methods to our model

so that this application

can have some

functionality.

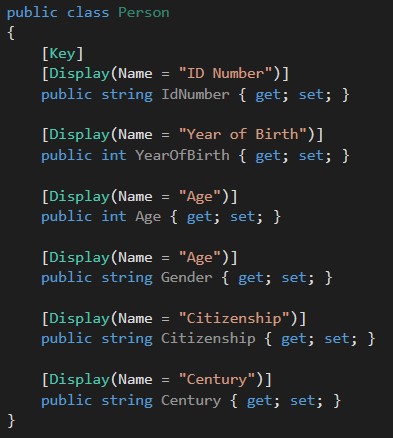
Let’s briefly discuss what

the application does. A

user will enter their

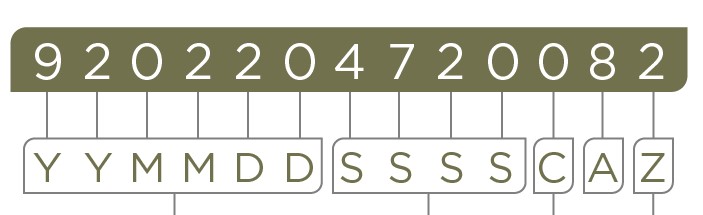
ID

number and from that



input, the application should automatically determine the year of birth, age, gender, and citizenship of that user.

How can we break down an ID Number to achieve this task? The following graphic will give you a clue.

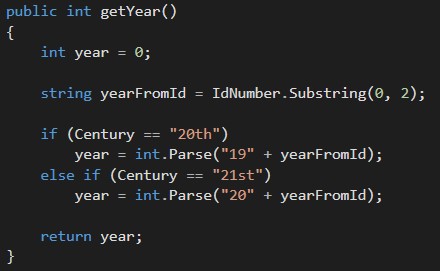


In an ID Number

* The first 6 numbers represent your date of birth
* The next 4 numbers determine your gender
* The next value denotes your citizenship

Let’s get cracking with those methods!

# Method #1 – getYear()

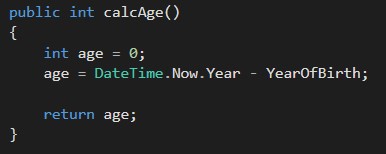


This method gets the year portion (first 2 numbers) from the ID Number and then according to what century that person was born in, we then determine the full year of birth.

* string yearFromId = IdNumber.Substring(0, 2); uses substring to extract the first 2 numbers from the ID. In Substring(0, 2); 0 means start at the beginning of the id, and 2 means extract exactly 2 values.
* The user will be asked to select the century in which they were born so that we can accurately predict their year of birth. If my yearFromId is

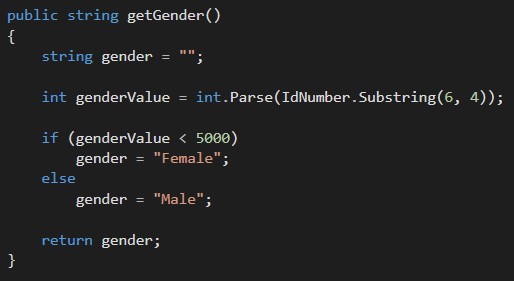
“97” and I select 20th century, then my year of birth will be “19” + “97”, which is 1997.

# Method #2 - calcAge()



Once we have the year of birth, it’s easy to calculate age. Just call the DateTime method to get the current year and subtract the year of birth.

# Method #3 – getGender()

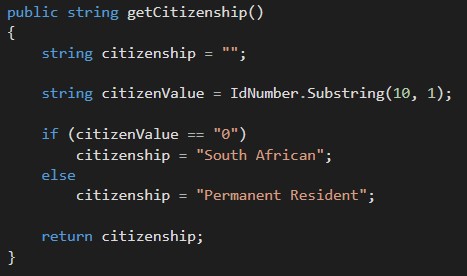


This method determines the gender based on the value given from the ID number. The 4 digits in your ID, after the date of birth tells you what gender you were born with. If these 4 digits are between 0000 and 4999, then you’re a female, but if they’re between 5000 and 9999, then you’re male.

So we use Substring(6, 4) to extract these 4 digits. 6 meaning start from the value at the 6th index, 4 meaning extract 4 values. Once we have these values, then we convert them to integer. After that, we compare and determine the gender.



# Method #4 – getCitizenship()



This method determines the citizenship based on the value at the 10th index of the ID number. If this value is 1, you’re South African but if it’s 0, then you’re a Permanent Resident.

So just like the previous method, we use substring to extract the value and then we compare it to determine citizenship.

Congratulations!!!! You’re not even halfway yet 🤣🤣

We have just created our model and this model will act as our Table in the database. But now, we need to create a database model, otherwise known as a Context class. Then we will add our Person class (table) into our Context class (database).

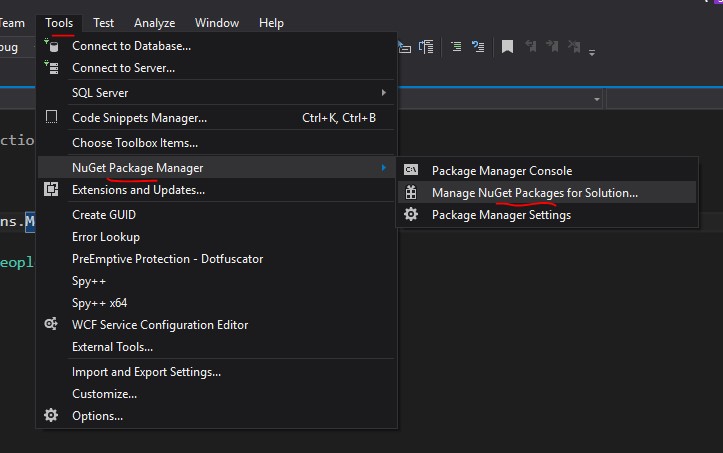
Now in your Solution Explorer – Right click on the Models folder and add a class. Name this class *PeopleContext* and click Add. This will be your context class (database).

BUT before we do anything with this context class, we need to install Entity Framework into our application. Entity Framework is a framework which allows us to map and query objects from databases within our application.

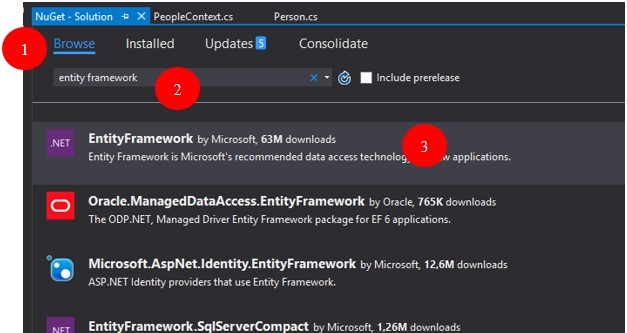
Essentially, without the Entity Framework, we cannot create applications that involve databases. Because we chose an empty template at the beginning, we need to manually install this framework. If you didn’t choose an empty template (why 🙃), then most likely you have Entity Framework already installed.

Here’s how to install Entity Framework –

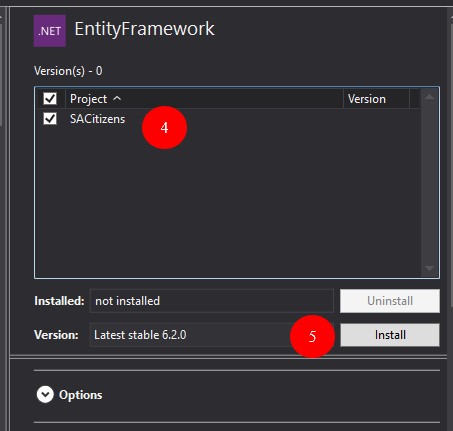
* On the task bar on top, click on Tools
* Then hover over Nuget Package Manager
* Then click Manage Nuget Packages for Solution



* When the page opens up: 1) Click Browse. 2) Search “entity framework” 3) Click one on the EntityFramework package



Then on the right side: 4) Click next to the name of your project. 5) Click Install



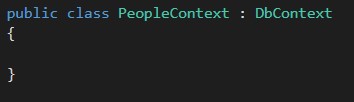
* First pop up that appears afterwards: Click OK to confirm installation.
* Second pop up that appears: Click I Accept to accept user license

Entity framework is now installed 😊

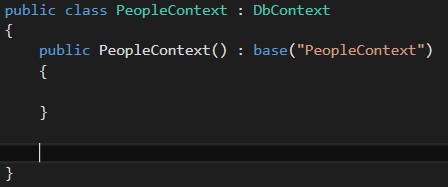
Let’s go back to our *PeopleContext* class –

* First step is to add a namespace. This namespace allows us to use Entity Framework within our context class.
* So now you can add using System.Data.Entity;to the top of your context class.

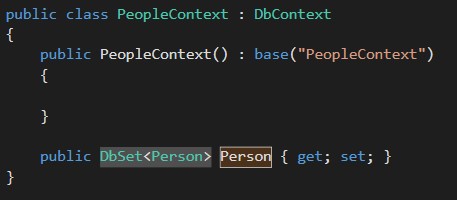
Then we need to turn our normal class into a context class by allowing the class to inherit properties from the DbContext. DbContext was not going to be accessible without installing Entity Framework.



Inside this context, we need to create an empty constructor.



And then, we need to add our table (Person class).



So above is what the final Context class should look like –

* We have PeopleContext inheriting from DbContext
* Our empty constructor with a base. Base is used to parse the name of the context to our DbContext class
* Then we added our person class as a DbSet, which will act as a table in our database.
* In public DbSet<Person> Person { get; set; } the word in between the < > is the name of the class and then the word after is the name that we want to give to our table. It is usually the same name as the class.

So all in all, think of –

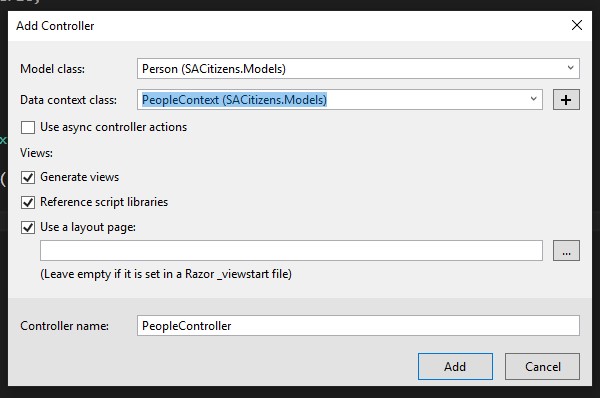
* DbContext as a database
* DbSet as a database table

The difficult part is done. Please remember to Build the application before you move to the next part 😀

Now we add a controller –

In your Solution Explorer – Right click on the Controllers folder and add a controller. Click on the third option; *MVC 5 Controller with views, using Entity Framework.* Then click Add.

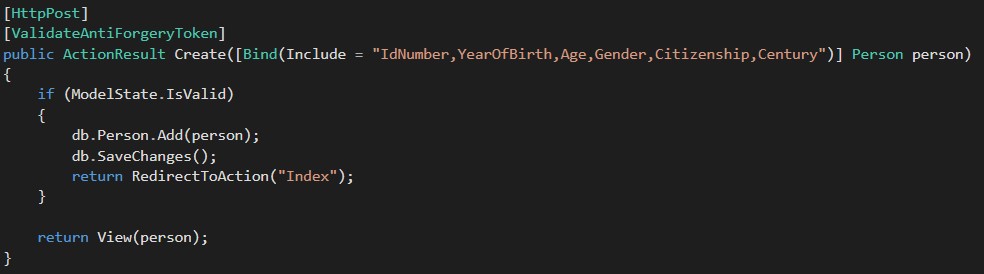
This will create a controller along with all the views that you need.



1. For model class, select the *Person* model that we created
2. For Data Context class, select the *PeopleContext* class that we create
3. Then click Add

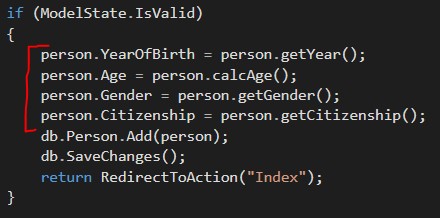
Visual Studio will now scaffold (create) everything for you.

When your controller pops up, scroll down and look for the [HttpPost]Create action method



This is the method that creates records and saves them into the database. So, what we need to do is assign our model methods into our model properties. So that when we create a new record, the methods are executed and the values are saved into the properties.

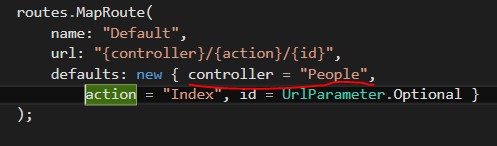
So just underneath, if (ModelState.IsValid, we will add the following 4 lines of code –



And I’m sure you’re familiar with this syntax and what it does exactly so no need for explanation here.

Now we need to edit the RouteConfig file. In your Solution Explorer – Collapse the App\_Start folder and open the RouteConfiq file.

Change the default route controller, in the last line, from “Home” to “People” (or to whatever your controller is called).

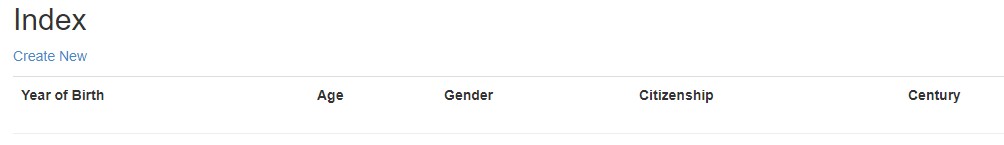


The Route Config file is a very delicate file in MVC. If you ran this program without changing this file, the application was going to look for the “Home” controller and since we don’t have one; you were going to come across an error.

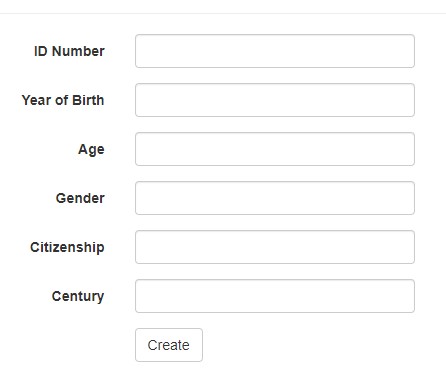
Now when you run this program, it will look for the “People” controller.

So, lets run the program and see what we have.

When the program opens – you’ll see a list of the properties from our model



Click on Create New - and then you’ll see a form



This is what EntityFramework has created for us. It has automatically “coded” text boxes for all of our properties. But we don’t want all of this. What is it that we want exactly?

1. The ID Number text box can stay because we need to input the id number anway
2. Year of birth, age, gender and citizenship need to go away. Because these values will be automatically generated once we enter our ID Number.
3. Century needs to change from a text box into 2 radio buttons so that the user can select which century they were born in.

We need to make these changes in the *Create* view. So, let’s close this application and –

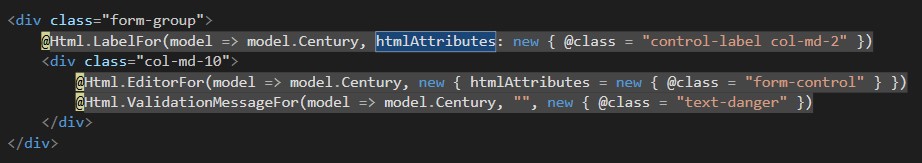
* In Solution Explorer, expand *Views,* then expand *People*, then open *Create.cshtml*

The code in this file should be familiar to you.

So now you will have to look for the form groups that house the code for year of birth, age, gender and citizenship and delete or comment them out.

Next, you will need to look for the form group that houses the code for Century and change it into 2 radio buttons.

Century Old Code:



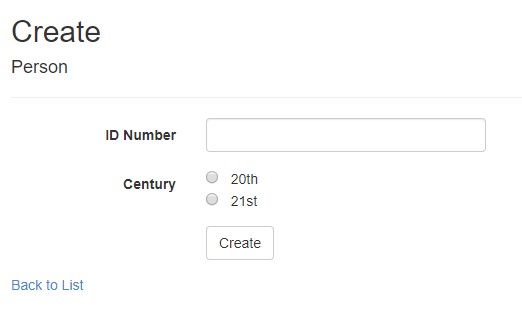
Century New Code (changes are highlighted):



New code to add:

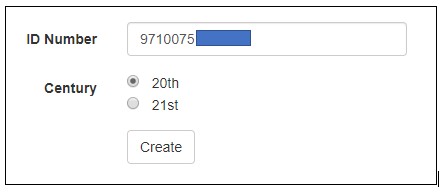
* @Html.RadioButtonFor(model => model.Century, "20th", new { htmlAttributes = new { @class = "form-control" } }) &nbsp; 20th <br />
* @Html.RadioButtonFor(model => model.Century, "21st", new { htmlAttributes = new { @class = "form-control" } }) &nbsp; 21st

Now we can run our application and test is out 👏👏 This is how your *Create* view should look like now –



1. Enter your ID Number
2. Choose the corresponding century
3. Click Create

And you should see all the details worked out for you, on the *Index* view 😊



**PLEASE NOTE:** Since ID Number is a primary key, you cannot use the same ID Number for more than one record. You will get an error.



**Code First with two models and foreign key(navigational properties)**

Create an MVC application that will create a Department and Wages models. Then code the two models below. The red highlighted portion in the Wage model is the navigational property which is actually the foreign key called departmentID.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace CodeFirstDepartmentWagesEg.Models

{

public class Department

{

[Key]

[DisplayName("Department ID")]

public int departmentID { get; set; }

[DisplayName("Department Name")]

public string departmentName { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.ComponentModel.DataAnnotations.Schema;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel;

namespace CodeFirstDepartmentWagesEg.Models

{

public class Wage

{

[Key]

[Required]

[DisplayName("Employee ID")]

public int empid { get; set; }

[Required]

[DisplayName("Name")]

public string name { get; set; }

[DisplayName("Department ID")]

public int departmentID { get; set; }

public virtual Department Department {get;set;}

[Required]

[DisplayName("Hours Worked")]

public int hoursworked { get; set; }

[Required]

[DisplayName("Rate of Pay")]

public float rate { get; set; }

[DisplayName("Basic Wages")]

public double basicWages { get; set; }

[DisplayName("Deduction")]

public double deduction { get; set; }

[DisplayName("Net Wages")]

public double NetWages { get; set; }

}

}

Create a context class as seen below and then install the entity framework by going to the tab tools and select Manage NuGet Packages. Then search for the latest entity framework and install the entity framework references. After this add the following name space highlighted below. Then scaffold the controller and view and run your application.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

namespace CodeFirstDepartmentWagesEg.Models

{

public class WagesDepartmentContext:DbContext

{

public WagesDepartmentContext()

:base("WagesDepartmentContext")

{

}

public DbSet<Department> Departments{ get; set; }

public DbSet<Wage> Wages { get; set; }

}

}

Now that your application works, add 2 methods on the Wage model. First method to calculate the basic wages. Another method to calculate the deductions as 10% of the basic wages. Last method to calculate the net wages and display all outputs on the view. Make sure to change the basicwages, deduction and net wages on the view to a DisplayFor. Call the methods in the controller.

**Code First Migrations**

1. Create MVC Application with the two models listed below as well as the context class.

public class Course

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

[DisplayName("Course ID")]

public int CourseID { get; set; }

[DisplayName("Course Name")]

public string CourseName { get; set; }

[DisplayName("Course Fees")]

public double CourseFees { get; set; }

}

public class Student

{

[Key]

[DisplayName("Student Number")]

public int StudentId { get; set; }

[DisplayName("Name")]

public string name { get; set; }

[DisplayName("surname")]

public string surname { get; set; }

[DisplayName("Course ID")]

public int CourseID { get; set; }

public virtual Course Course { get; set; }

}

1. Add the entity framework

public class StudentContext:DbContext

{

public StudentContext() : base("StudentDatabase")

{

}

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

public DbSet<Course> Courses { get; set; }

}

1. Build the app and now we realise that we should have added the field to the Student table.
2. Go to Student model and add the following code after the foreign key:

[DisplayName("Total Fees")]

public decimal TotalFees { get; set; }

1. Once you have done that, now we have changed the model so the database needs to change.
2. Go to Tools > NuGet Package Manager > Package manager console
3. Then a console box will open up and allow you to start code first migrations.
4. Type

**enable-migrations**

and press enter, then it will start migrations, then type

**update-database**

it will update the changes which were the field TotalFees added to the model and it will add to the table.

1. Go and scaffold the controller and views for the Course and Student.
2. Run app and you will see that the Total Fees has been added to the application