A Simple Step to Entity Framework: Code First Approach

Introduction

The Entity Framework is an Object Relational Mapping (ORM) providing an automated mechanism for CRUD operations to a database.

Entity Framework provides various approaches like:

* Database first
* Model First
* Code First

This article explains the Code First approach. This is mainly used for domain driven development. Initially we are not focusing on the database development, so we are creating classes instead of tables in the database. Based on these classes we can simply create our database.

**A quick start**

**1.** Create a console application:

Graphical user interface, text

Description automatically generated

**2.** Add a new class:

Graphical user interface, text

Description automatically generated

**3.**Class name as Employee:

Graphical user interface, text

Description automatically generated

**4.**Write the field names in the Employee table (consider this Employee class is your Employee Table):

Text

Description automatically generated

**5.** Go to the project References => Manage NuGet Packages.

Graphical user interface, text, application

Description automatically generated

**6.** Install EntityFramework:

Text

Description automatically generated

**7.** Return to our Employee class and add the reference:

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

**ComponentModel.DataAnnotations**

Provides attribute classes to define metadata for ASP.NET MVC and ASP.NET data controls.

[System.ComponentModel.DataAnnotations Namespace](http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.aspx).

**Some examples**

Table

Table Name

Column

Column name

Key

PK for the table

ForeignKey

FK Relation

Required

Required field validation

MinLength

Min length of the field

MaxLength

Max Length

Text

Description automatically generated

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
6. **using** System.ComponentModel.DataAnnotations;
8. **namespace** CodeFirstSampleAp
9. {
10. **public** **class** Employee
11. {
13. **public** Employee()
14. {
16. }
18. [Key]
19. **public** **int** EmpID { **get**; **set**; }
20. **public** **string** EmpName { **get**; **set**; }
21. **public** **string** Address { **get**; **set**; }
22. }
24. }

Add a new class MyContext and inherit the DbContext with a "using" for the reference as in the following:

using System.Data.Entity;

Text

Description automatically generated

Next, we need to plan where we need to create our database. So in the app config create our connection string.

Text

Description automatically generated

<connectionStrings>

<add name="MydbConn"

connectionString="Data Source=.;Initial Catalog=CompanyDB;Integrated Security=true"

providerName="System.Data.SqlClient"/>

</connectionStrings>

Then return to our MyContext class and pass our connection details here.

Also, we can plan our database Initializer. That means all the time we don't want to create a new database. Based on our requirements we can plan this.

Text

Description automatically generated

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Data.Entity;
4. **using** System.Linq;
5. **using** System.Text;
6. **using** System.Threading.Tasks;
8. **namespace** CodeFirstSampleAp
9. {
10. **public** **class** MyContext : DbContext
11. {
13. **public** MyContext()
14. : **base**("MydbConn")
15. {

           Database.SetInitializer<MyContext>(**new** CreateDatabaseIfNotExists<MyContext>());

2. }
4. **public**  DbSet<Employee> Employees { **get**; **set**; }
6. }
7. }

More options for the database Initializer:

Text

Description automatically generated

Suppose we are working with a production db, so we can prevent or turn off this feature.

1. Database.SetInitializer<MyContext>(**null**);

Next go to main and add a value to our new table:

Text

Description automatically generated

1. using System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** CodeFirstSampleAp
8. {

**public** **class** Program

    {

**static** **void** Main(**string**[] args)

        {

MyContext context = **new** MyContext();

Employee emp = **new** Employee() { EmpID = 1, EmpName = "Shiju", Address = "Cochin" };

context.Employees.Add(emp);

context.SaveChanges();

        }

1. }
2. }

Yes! Our Code First application is completed.

Go to our db. It automatically created the new db name CompanyDB that we specified in the AppConfig file.

Graphical user interface, application

Description automatically generated with medium confidence

Check whether the table has our inserted value.

Graphical user interface, text, application

Description automatically generated

Database First Approach In Entity Framework

In this article, we are going to learn what is Entity Framework (EF)? How to implement it in our application and approaches of EF like the following:

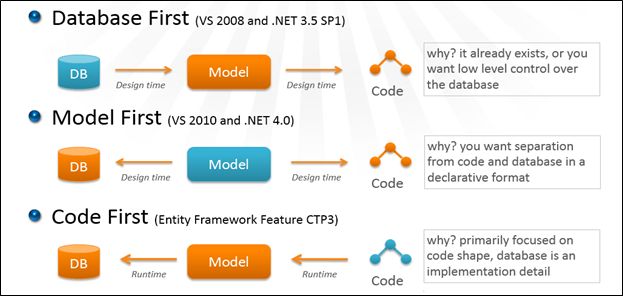
1. Database First Approach.
2. Model First Approach.
3. Code First Approach.

What is Entity Framework?

EF is a data access framework from Microsoft that helps to bridge the gap between data structures and objects in your application.

It automatically,

1. Generates strongly-typed entity objects that can be customized beyond 1-1 mapping.
2. Generates mapping/plumbing code.
3. Translates LINQ queries to database queries.
4. Materializes objects from data store calls.
5. Tracks changes in generating updates/inserts.



In the above screenshot, you can see how one by one approach comes in EF. Now we are learning the first approach i.e. Database First.

Database First Approach In Entity Framework

In this article, we are going to learn what is Entity Framework (EF)? How to implement it in our application and approaches of EF like the following:

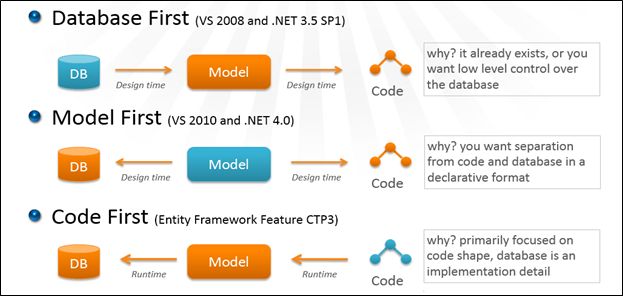
1. Database First Approach.
2. Model First Approach.
3. Code First Approach.

What is Entity Framework?

EF is a data access framework from Microsoft that helps to bridge the gap between data structures and objects in your application.

It automatically,

1. Generates strongly-typed entity objects that can be customized beyond 1-1 mapping.
2. Generates mapping/plumbing code.
3. Translates LINQ queries to database queries.
4. Materializes objects from data store calls.
5. Tracks changes in generating updates/inserts.



In the above screenshot, you can see how one by one approach comes in EF. Now we are learning the first approach i.e. Database First.

Database First Approach

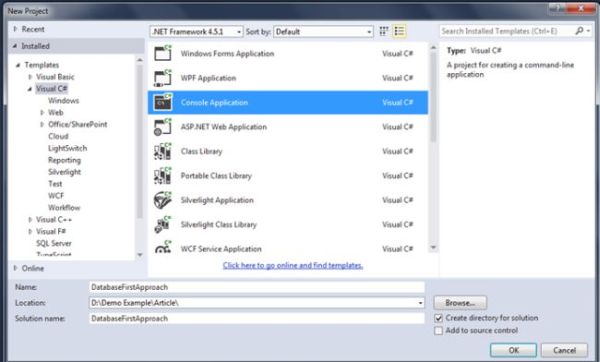
The name specifies the functionality of this approach means the database is already there and we are just adding the entity model in our application and using that model we accessing the database objects and data.

The following are the steps for using the Database first approach in your application:

1. Open Visual Studio.

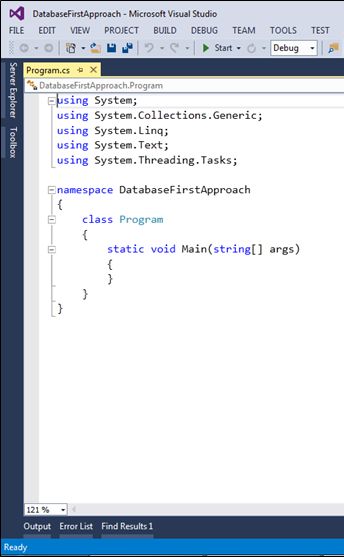
1. File, then New Project.

1. Select C# Temple and select Console Application name it ‘**DatabaseFirstApproach**’.



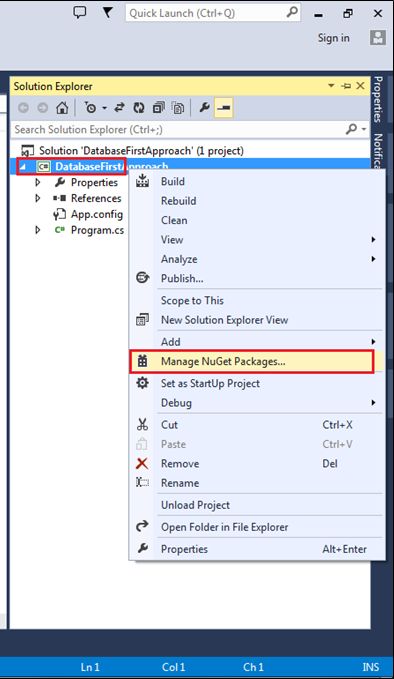
1. Click on OK.

1. You can see  the following screenshot:

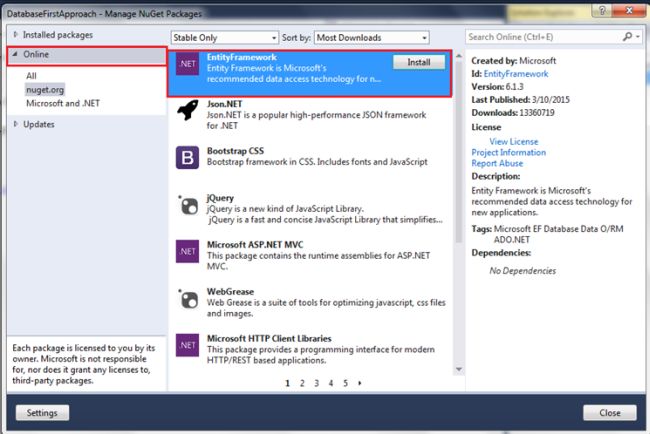


1. Now add Entity Framework Nuget package.

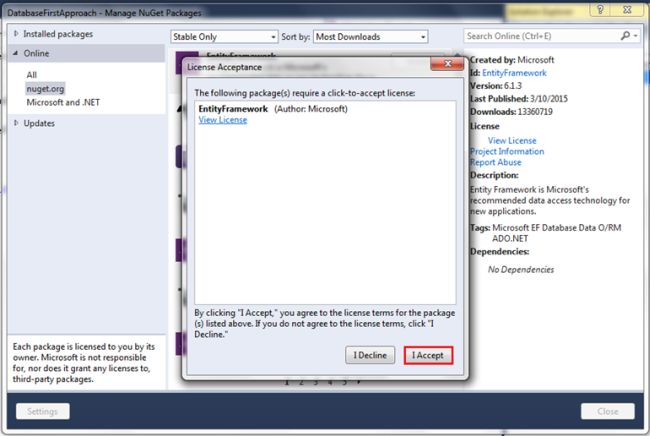
1. Right-click on the project and select Manage NuGet packages.



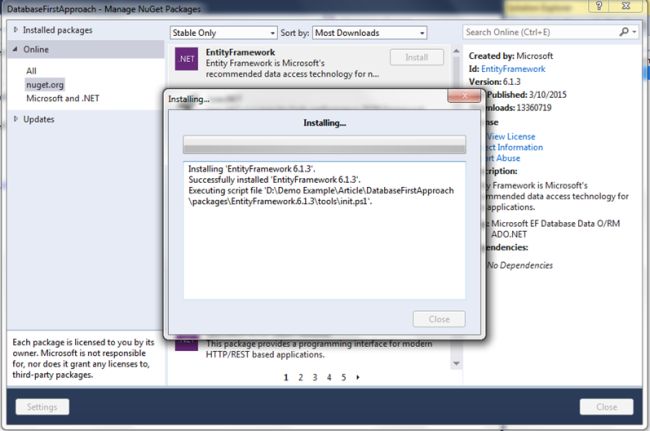
1. You can see a window like the following screen:



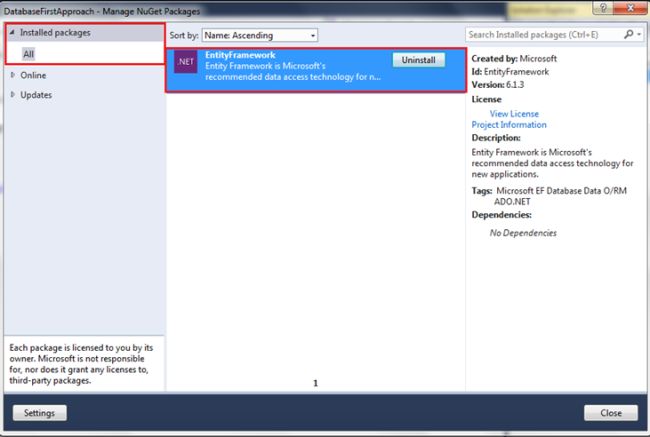
In the above screen, you can see the Installed Package on the Project or search online, click on Installed.



Click on Accept, it will show the following installation screen:



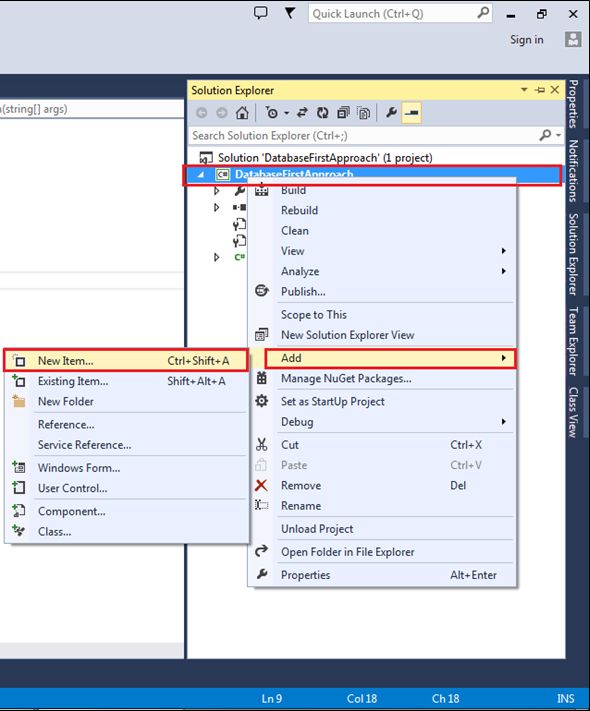
After installation is finished, you can see it in the installed packages on the left side like the following:



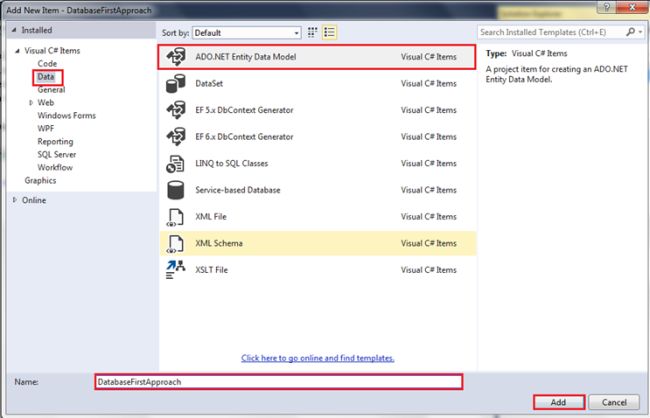
Click on close.

1. I have attached the database script with this demo just download it, create a database, and run that script.

1. Now add Entity Data Model in your application: Right Click, then Add New Item.

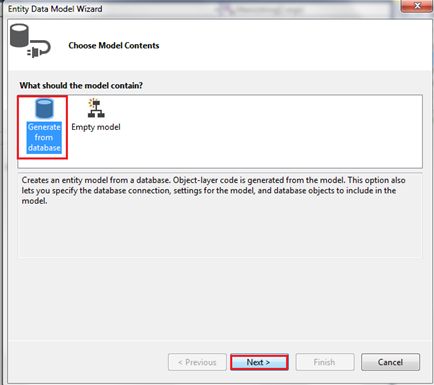


1. Here's the next window,



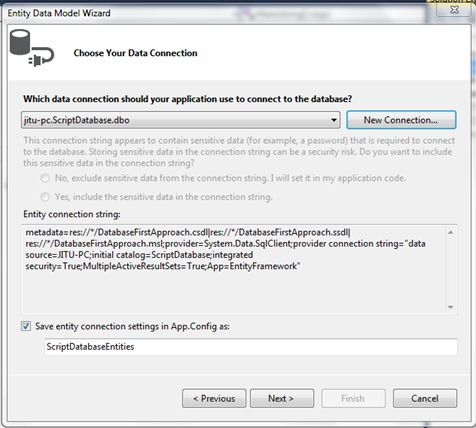
Select Data from the left side, then select ADO.NET Entity Data Model and name the model. After that click Add.

1. After Clicking Add it will populate another window:

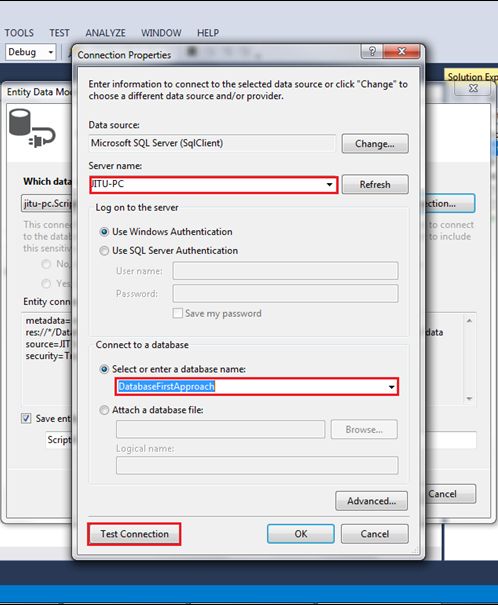


Here we are generating a model from the database, so select first and click the next button.

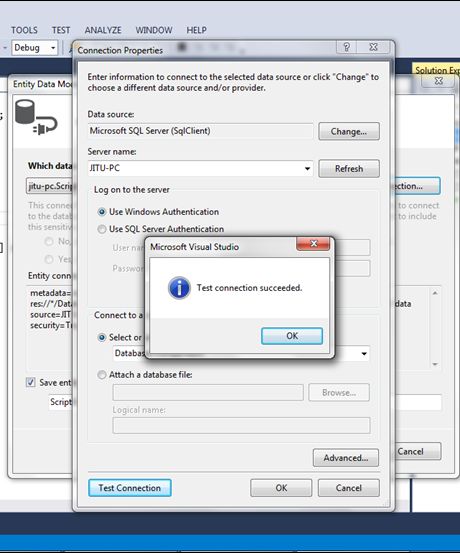
1. Next comes the Entity Data Model Wizard for database selection.



Click on New Connection,

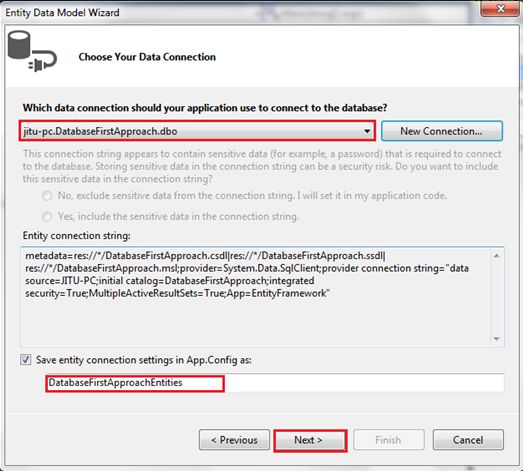


Select here server name, database name, and click on Test Connection for checking.



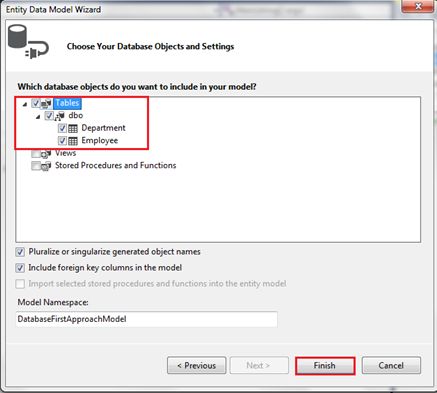
Now connection succeeded.

1. Next, click on the OK button, you can see the following screen:



The above screen shows that we selected database name and connection string name in App.config file -> Click Next.

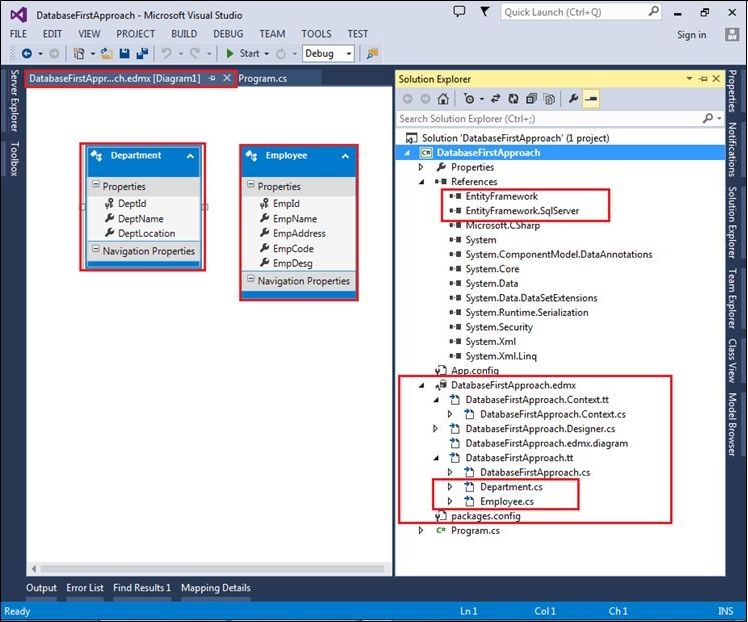
1. After clicking it will show you the choose your database objects or table settings like the following:



1. In the above screen, I selected all the database objects, I have only two database tables,

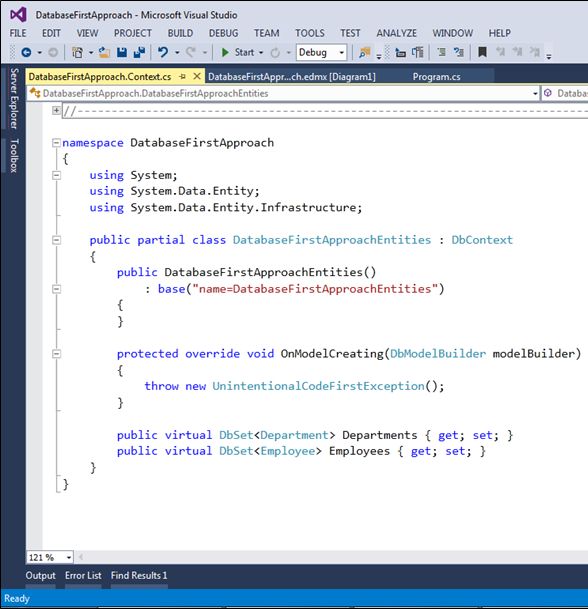
* 1. Department
  2. Employee.

Next, click on the Finish button after clicking you can see the following screen:



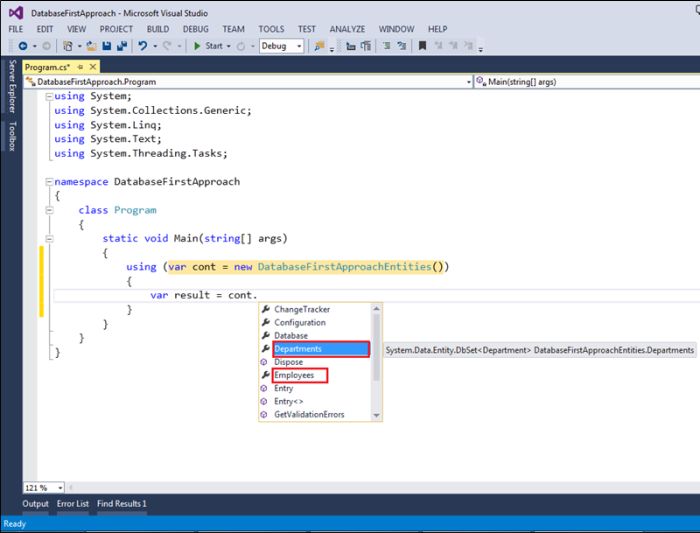
In the above screen, you can see the Entity Data Model file with two entities we previously added Department and Employee. On the right side in the solution explorer, you can see in the boxes, first is Entity Framework and SQL Server references and next is the DB Context file generated and two database objects Department.cs and Employee.cs.

1. The following screen shows the auto-generated entities file with DbContext inherited:



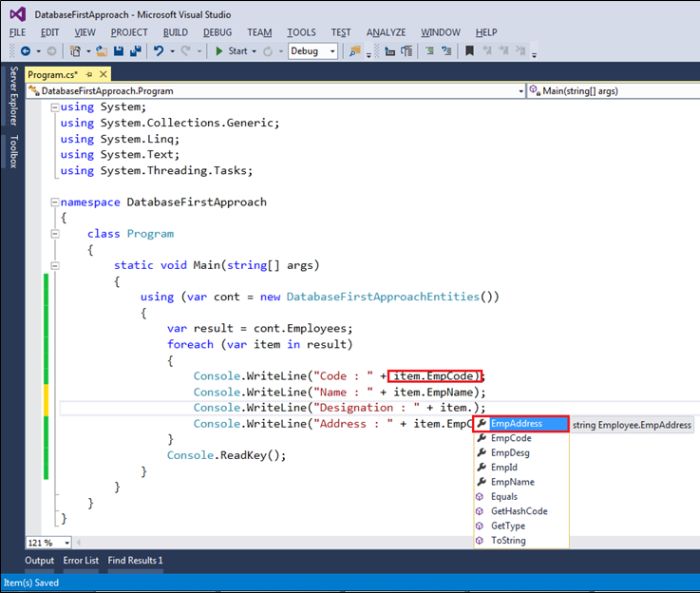
1. Now, the next step is accessing data from this DBContext file using the objects Department and Employee.

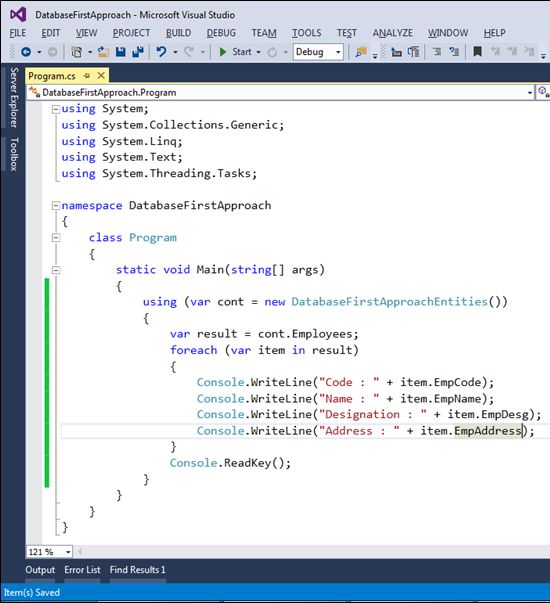
1. Open Program.cs file and write the following data access logic in that:



When you create an object of context class and access objects using (.), then it will show the list of database objects.

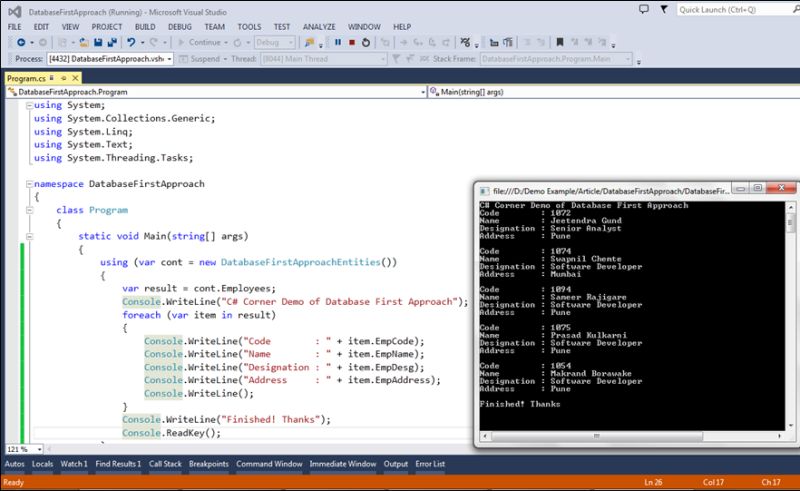
1. You can access properties of that database objects like the following:





The above screen shows the access data from objects and showing in the command prompt. Now, save the changes and click on Run.

1. You can see the following output:



Great, your first Entity Framework approach Database First Approach created successfully!

Model First Approach

The name specifies the functionality of this approach means creating model first using the Entity Framework Designer tool. We are just adding the entity model in our application, after generating the database from this model and using that model we are accessing the database objects and data.

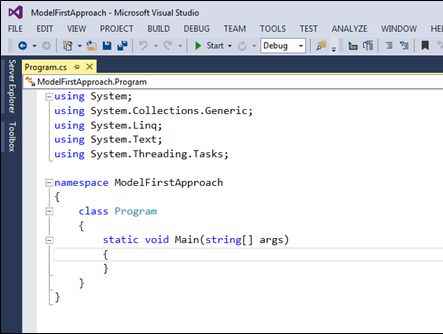
The following are the steps for using the Model first approach in your application:

1. Open Visual Studio 2013.

1. File, then New Project.

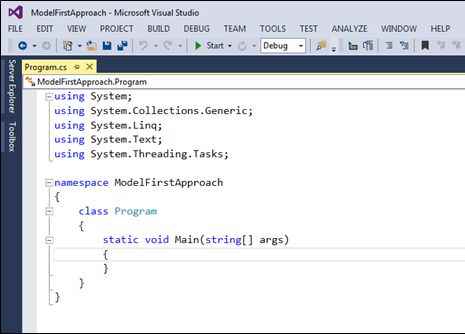
1. Select C# Temple and select Console Application name it ‘**ModelFirstApproach**’. Click OK.

1. You can see the following screen:



1. Next, add Entity Data Model to design database.

1. Right Click on Project, Add, then click New Item.

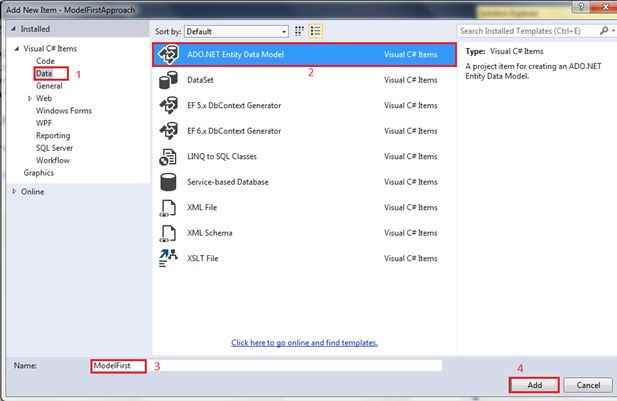


1. It will open a new window. Select data from the left side, then select ADO.NET Entity Data Model and name it ‘**ModelFirst**’ and finally click Add.

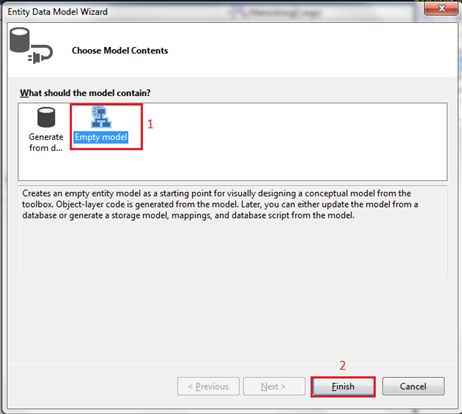
Graphical user interface, application

Description automatically generated

1. After clicking on *the Add* button it will prompt the next window for Entity Data Model Wizard, in that select Empty Model because we are generating a database from the model. After that click on Finish.

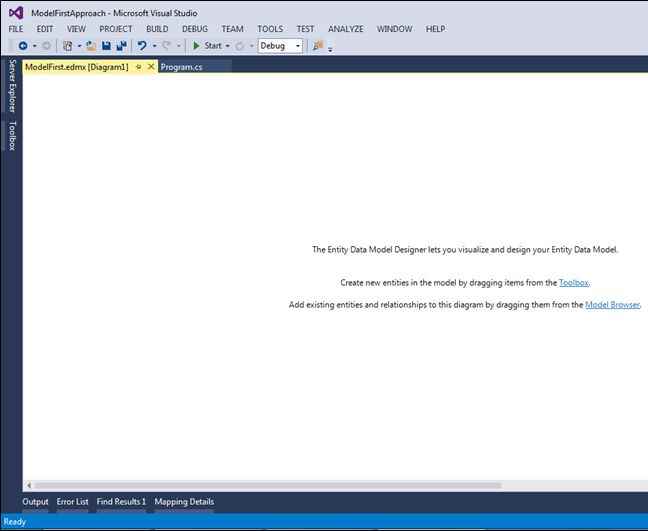


1. After clicking on the Finish button, it will open the**Entity Data Model Designer**like the following image.

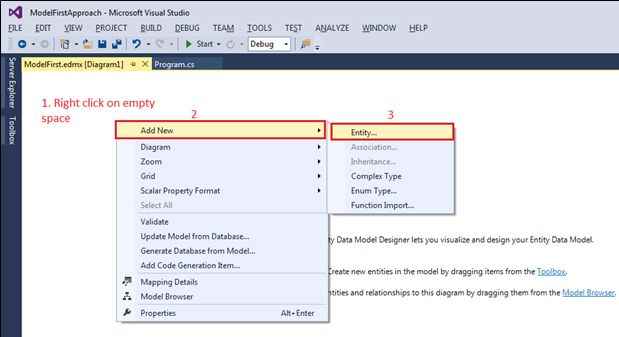


1. Right-click on empty space for adding a new Entity or object or table for a database like the following way.

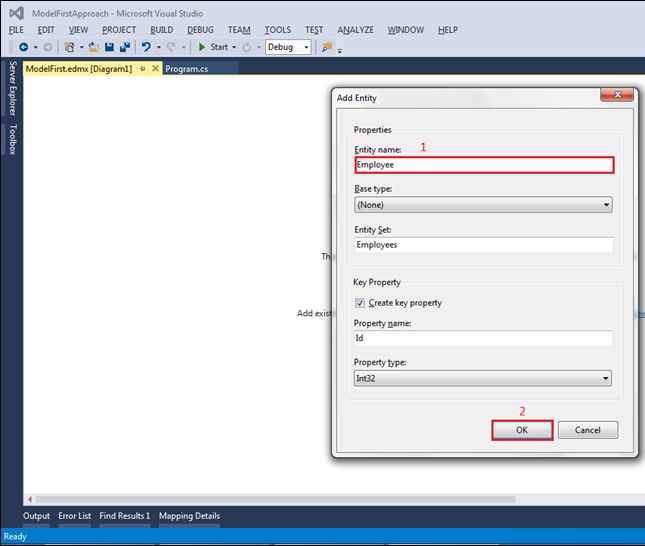
Right-click, Add New, then click Entity.



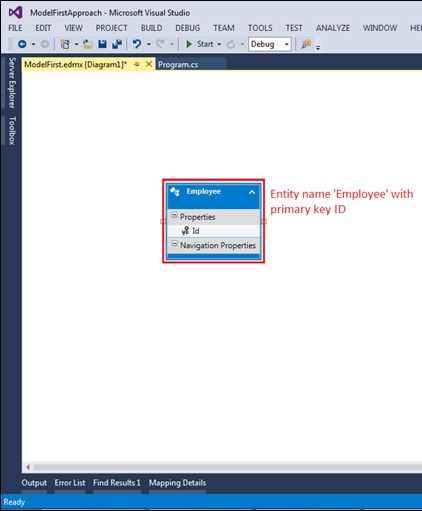
1. After the above step, it will open the add entity window providing the entity name or table name ‘**Employee**’ and clicking OK.



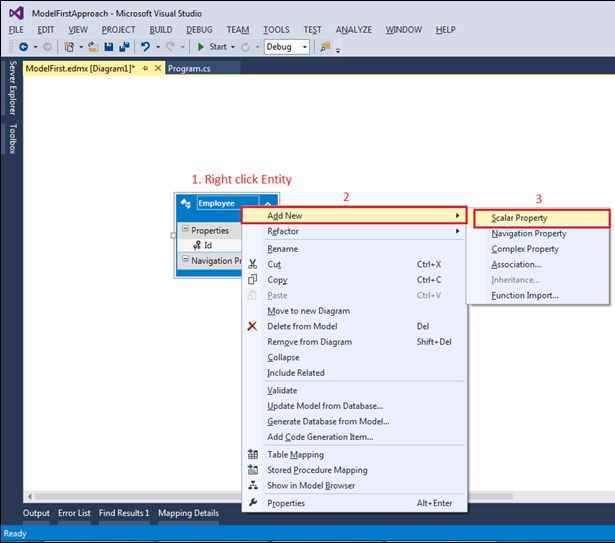
In the following image Create Key property means creating the primary key by default **ID** with the data type **Int32**.



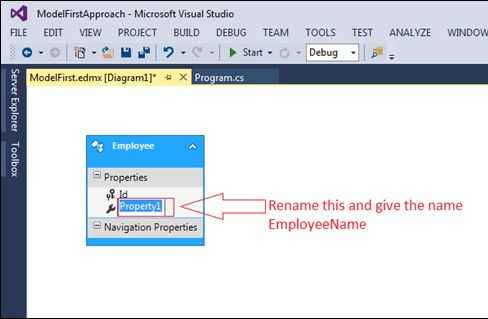
1. After clicking on the OK button you can see in Entity Data Model Designer one entity is created with the name ‘**Employee**’.

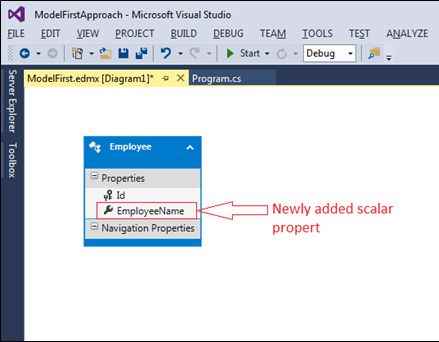


1. Now, next, add more columns or scalar property. For that Right-click on Entity, Add New, then select Scalar Property like the following image.

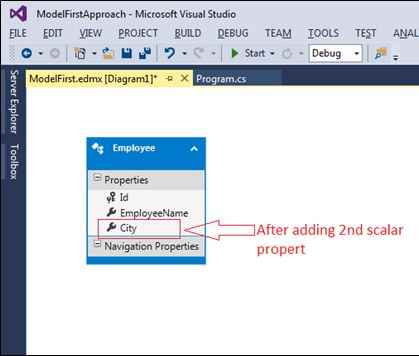


1. Next, it will show in entity model with added scalar property ‘**Propert1**’ name, rename it to ‘**EmployeeName**’.





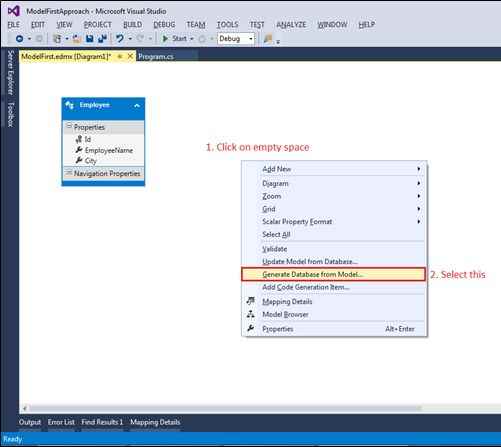
Repeat Step 13 to add one more scalar property ‘**City**’.



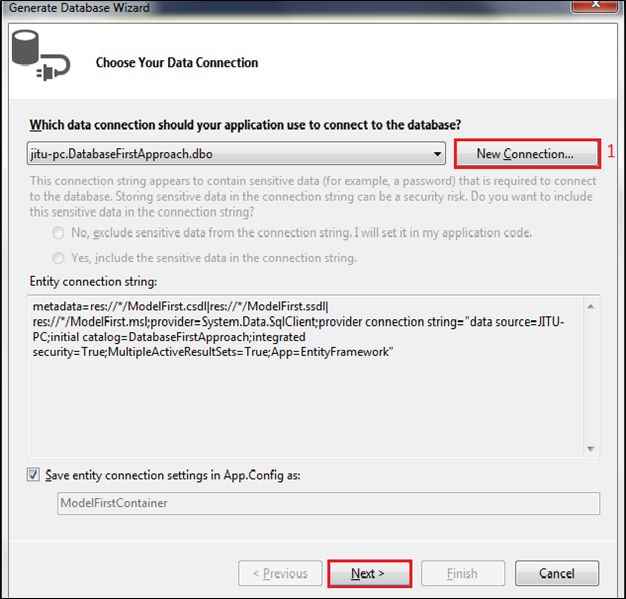
Now, this is our final entity with three columns with the primary key seen in the preceding image.

1. Next, generating a database from this Entity Model.

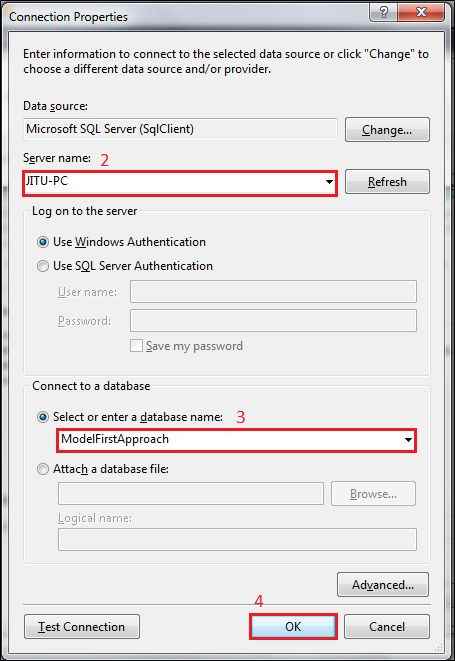
1. Right-click on empty space and select *Generate Database from Model*.



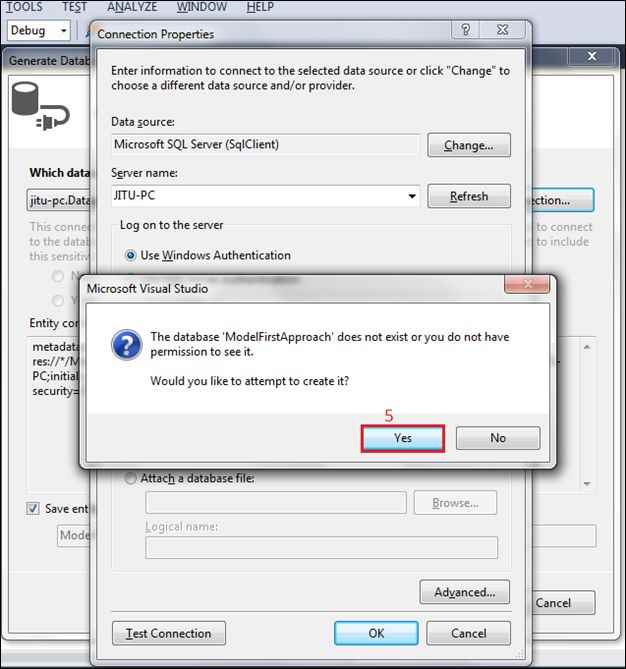
1. After clicking on **Generate Database from Model**, it will prompt the window of **Generate Database Wizard** for establishing the connection to create a database or use an already created database.



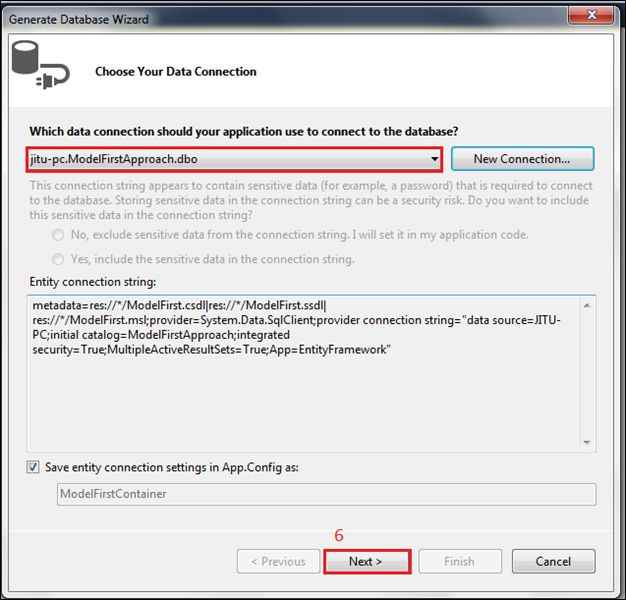
After clicking New Connection, it will open *the Connection Properties* window for selecting database instance and server name. Here I selected ‘**JITU-PC**’ and gave the database name or selecting the existing database. Here I provided the database name because I am going to create a new database with the name ‘**ModelFirstApproach**’.



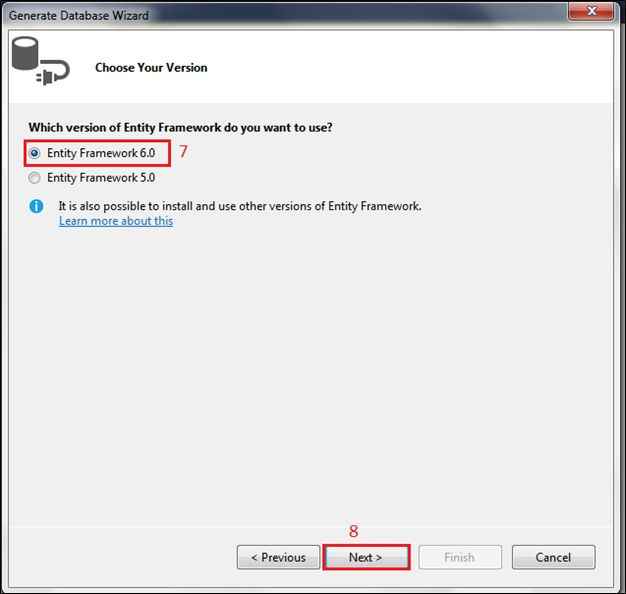
After clicking the OK button it will prompt you with a message box with the message as in the following image, Click on *Yes*. If you select *No,* then the database is not created.



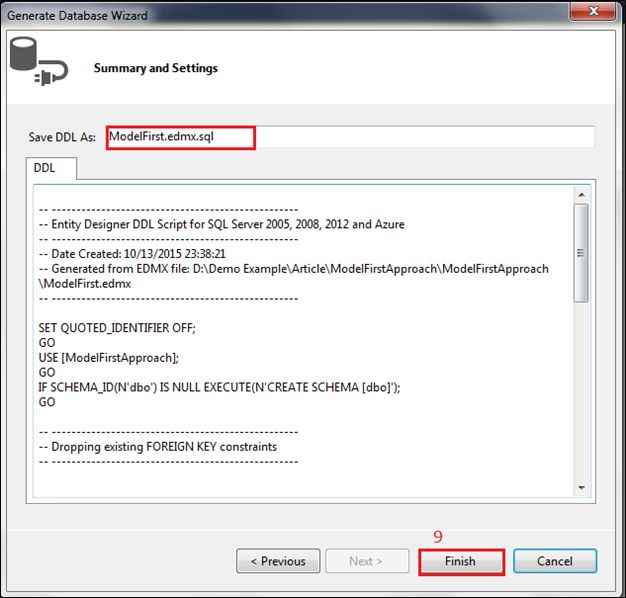
It will show you the previous window with the updated database name as in the following image:



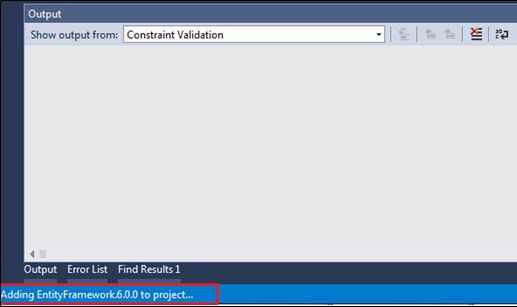
Click on *the Next* button. It will open the next step for selecting *an Entity Framework* Version to use for creating the database. Here I selected version 6.0.



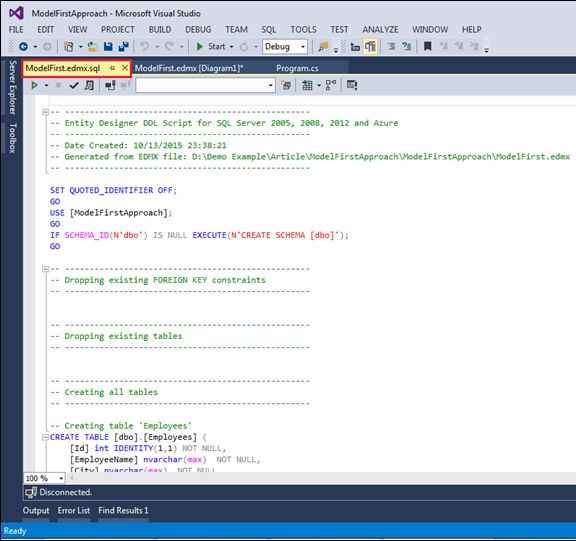
Click on *the Next* button, it will show you the last step.

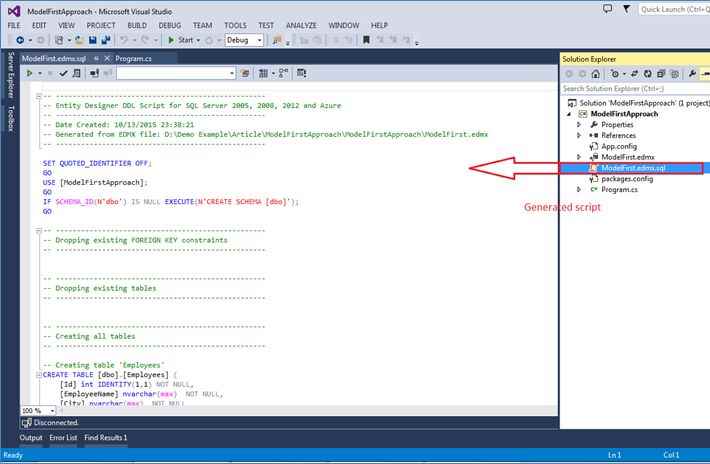


In this model first approach, the entity framework generates a .sql file or script for creating the database. After that click on *the Finish* button.

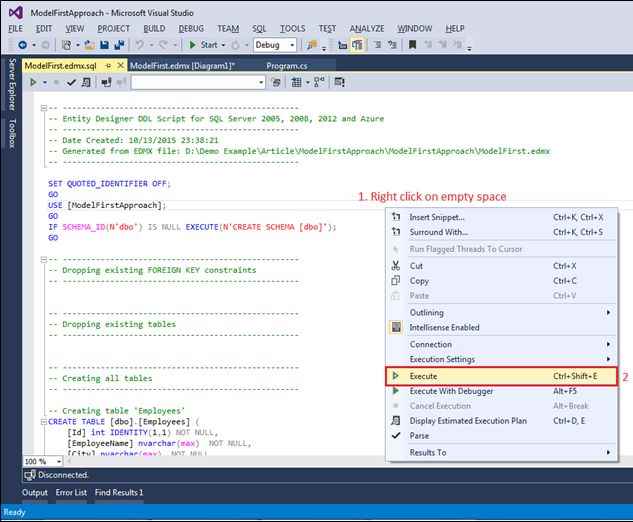


See the bottom of the image, after clicking on finish it started adding the Entity Framework. Finally, you can see the generated script of the database on the visual studio page like the following.





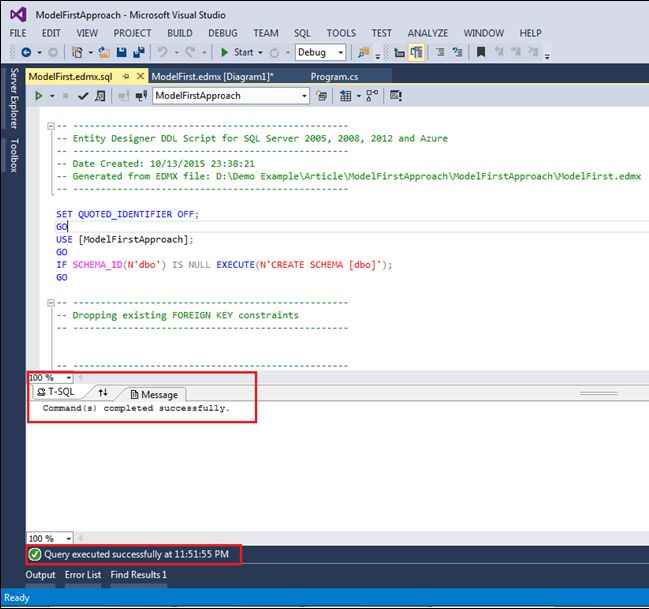
1. Now the final step, Right-click on the opened script file and select *Execute*.



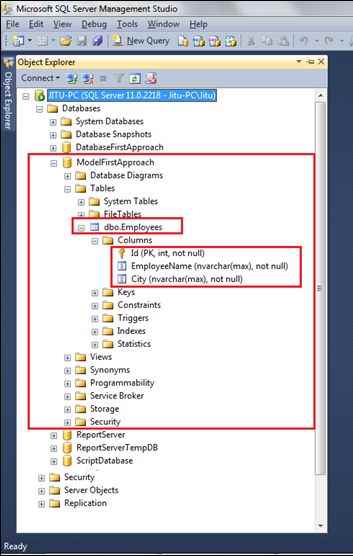
1. After selecting Execute, it will prompt you to one window for Connecting to the Server. Enter *the Server name* or select ‘**JITU-PC**’ and click on *Connect*.



1. After clicking on *the Connect* button it will execute the generated scripts.



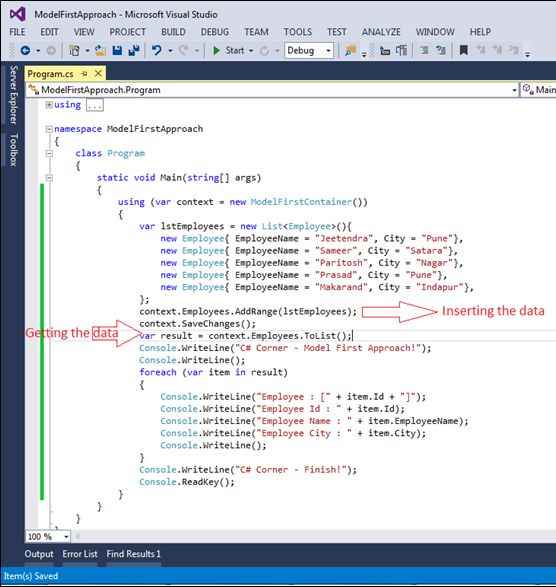
1. Now just open the SQL Server Management Studio to check the database is created or not. See the following image, after opening the SQL Server the database is created.



1. Congratulations! You have created the database using the Entity Framework Model First Approach.

1. Now, test the application inserting the record and getting the same record on-screen.

1. Add the following code on**Program.cs**file.



1. Save the changes and **Press F5**or **Run** the application, it will show you the following output:

Data base first approach using MVC

# Database First Approach in Entity Framework

## Introduction

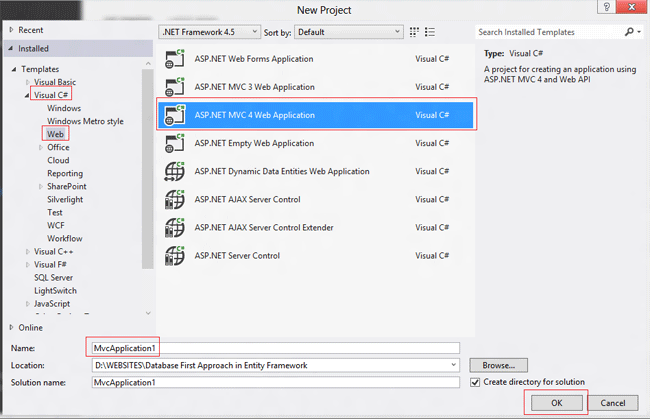
Read my last article "[Code First Approach in Entity Framework](https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/code-first-approach-in-entity-framework/)" for an EF introduction even for the basic level.

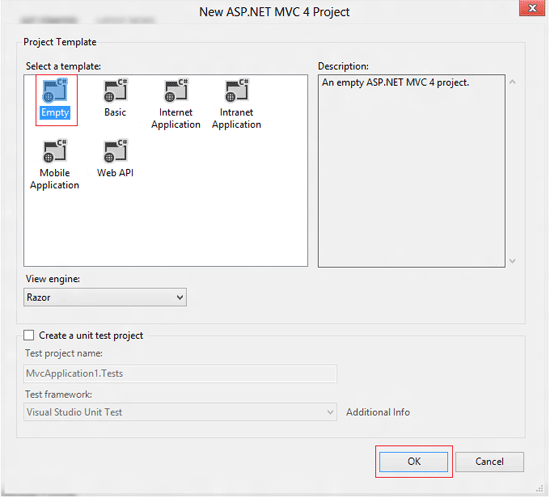
## What is Database First Approach?

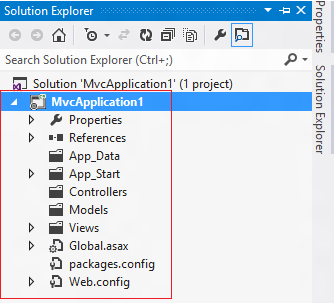
The Database First Approach provides an alternative to the Code First and Model First approaches to the Entity Data Model and it creates model codes (classes, properties, DbContext, etc.) from the database in the project and those classes become the link between the database and controller. If you remember my last article "[Code First Approach in Entity Framework](https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/code-first-approach-in-entity-framework/)" (I'll recommend you to read my "Code First" article, since there are many basics covered), where first we have created model codes (classes, properties, DbContext) and then these model classes create a database for us at run time and that's why we called it "Code First".

## Demo MVC Application

Create a new ASP.NET MVC Project by New > Project > ASP.NET MVC 4 Web Application > Empty Template, follow the images:



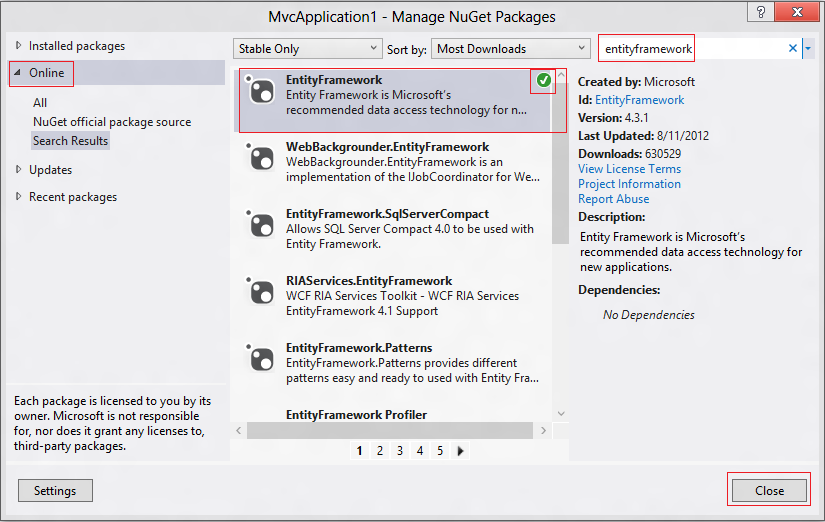




Now, follow the steps.

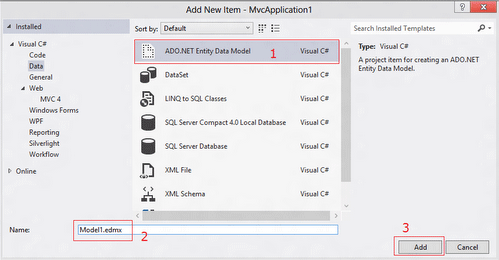
**Step 1:** Adding NuGet Package (if not available in references)

Right-click on the References folder and select "Manage NuGet Packages". Alternatively, you can install it from Tools > Library Package Manager > Package Manager Console and type "Install-Package EntityFramework". Okay, let's do this from the NuGet Package window, type "Entity Framework" in the search box, and click to install it. After installation, you will notice a new library file in References Folder "EntityFramework".



**Step 2:**Adding Model1.edmx

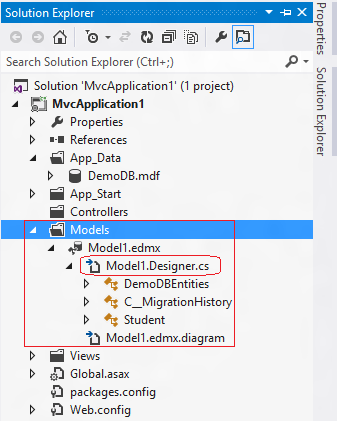
Now, we have all the required NuGet libraries. Let's go ahead and add an "Entity Data Model" file by right-clicking on the Model folder and select Add > New Item and in the window select "ADO.NET Entity Data Model", you can see additional steps in the image below. Remember to select the "Generate from database" option in the "Entity Data Model Wizard" and select any SQL Server Database or you may select Database file (.mdf). I'll be using Database file in this demo since my database file is located at another location (not in the project) so, I'll be prompted to copy the database file into the project's "App\_Data" folder. If you will select SQL Server or any other database vendor, you will not be asked to do so. Please refer to the image:



A screenshot of a computer

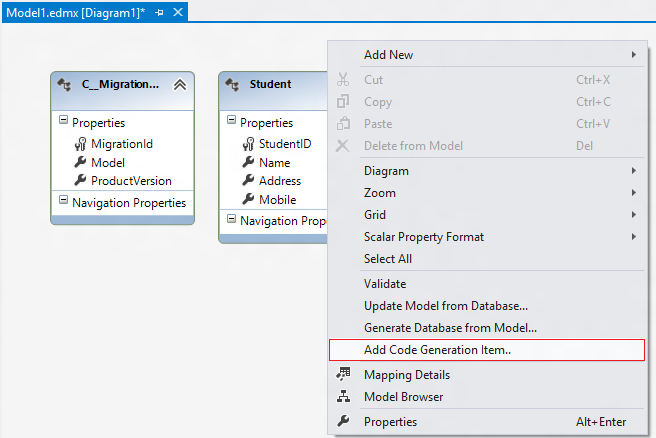
Description automatically generated with medium confidence

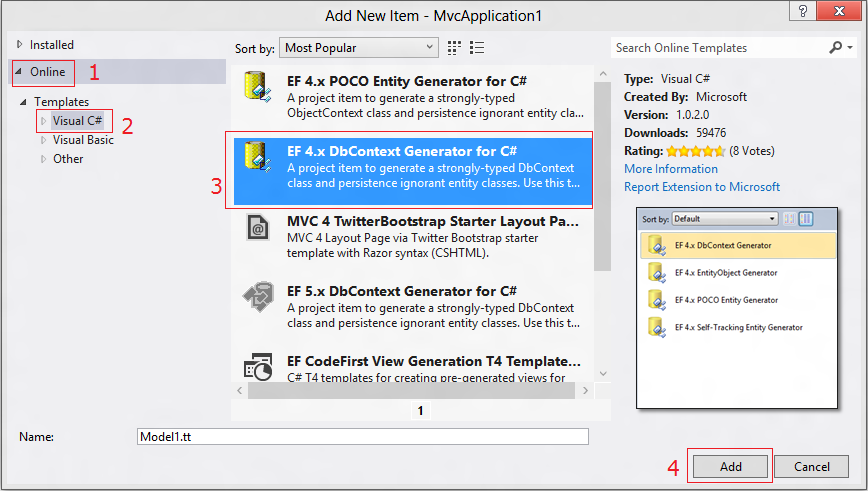
Now, you have an Entity Data Model file in your Model folder with all its necessary supportive files.



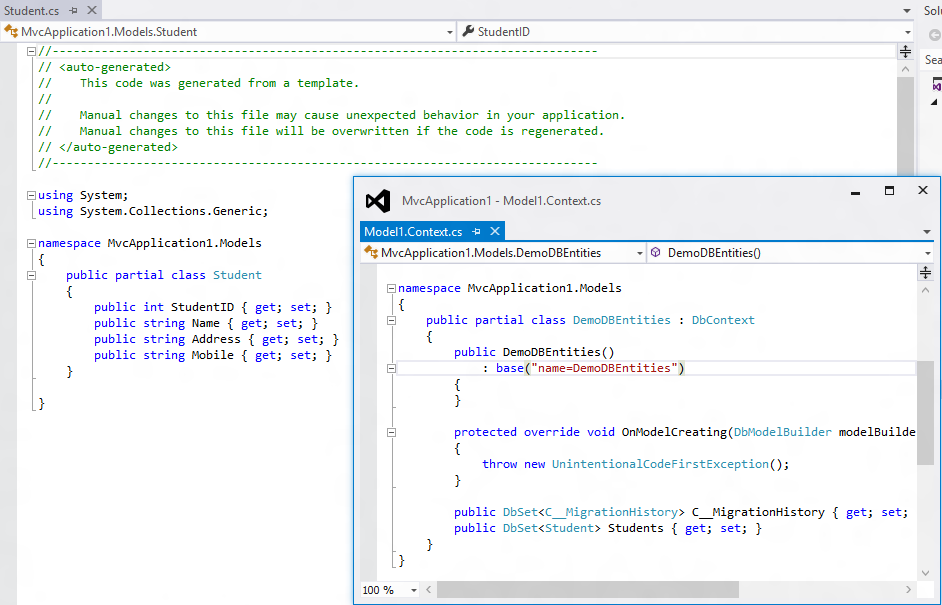
If you open the "Model1.Designer.cs" file, you will see all the properties, DbContext and DbSet that we need to establish a link between the model and the database, but you know these lines are a bit complicated or you can call it "not clean". Don't worry, we can create a cleaner version of this file. How?

You can do this by right-clicking on the "Model1.edmx" file and selecting "Add Code Generation Item" and from the window install "EF DbContext Generator for C#". Follow the image:



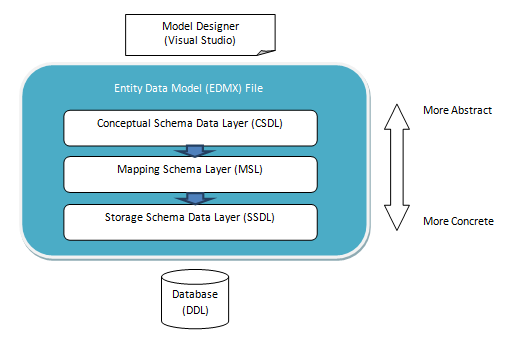
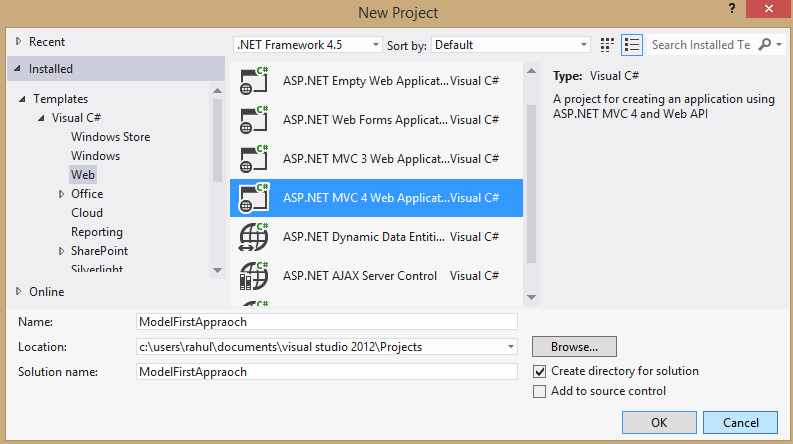
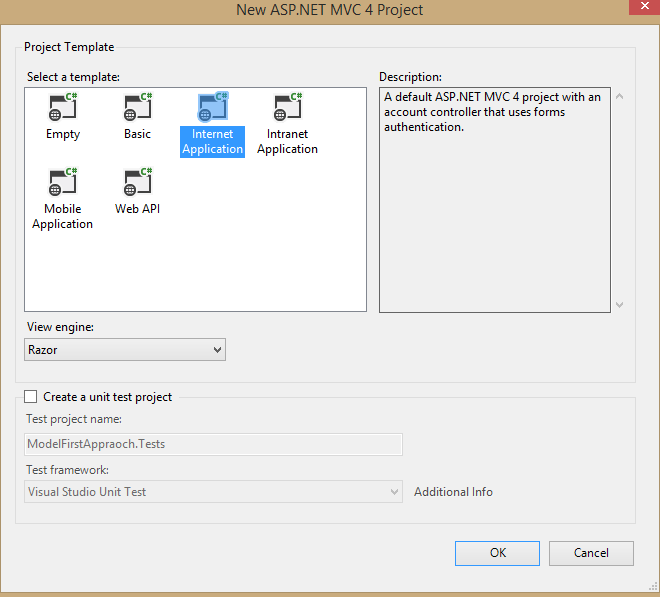
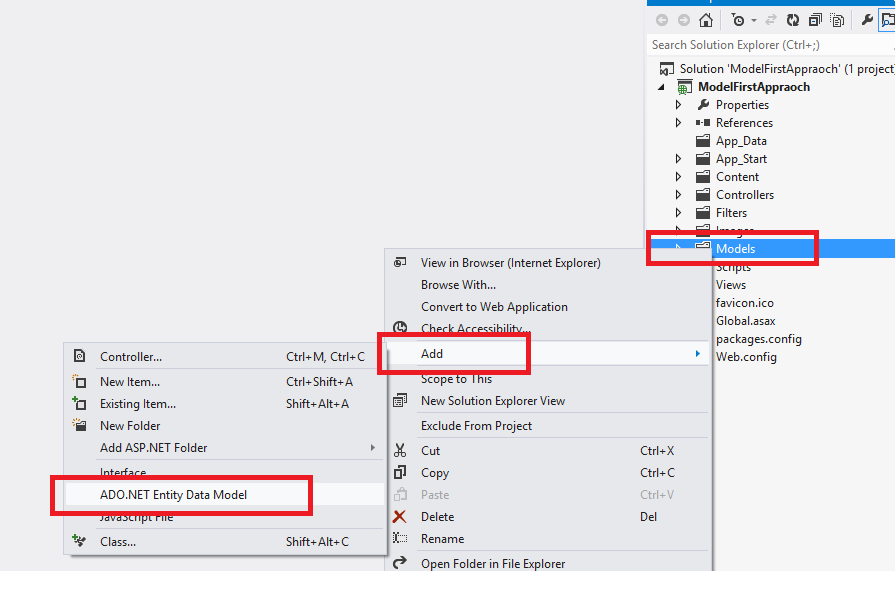


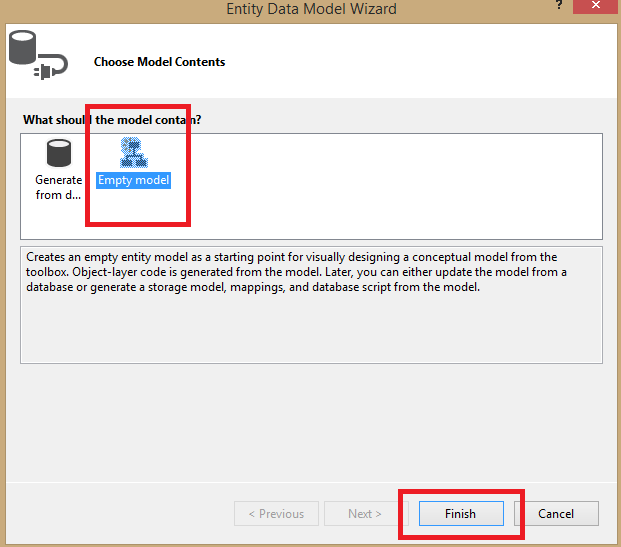
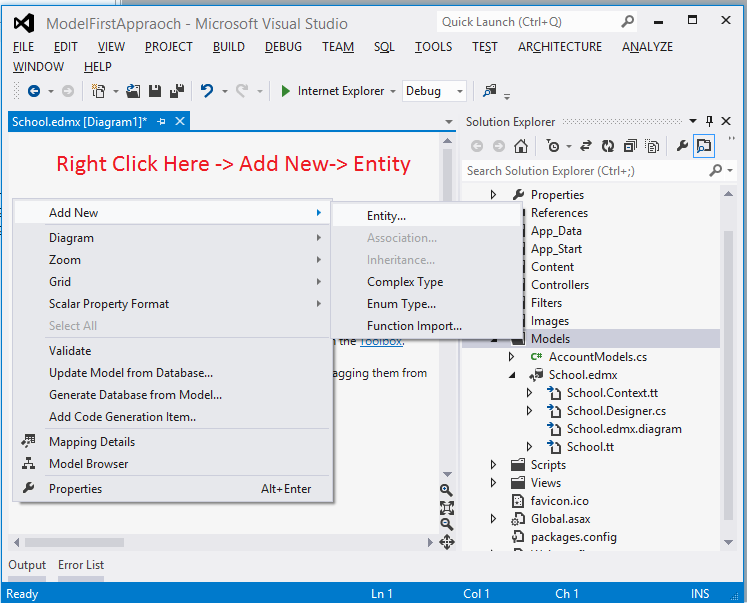
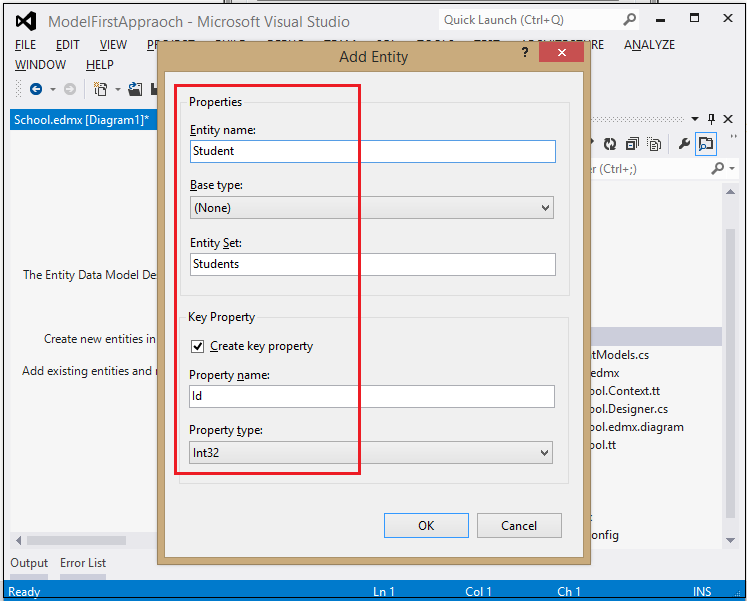
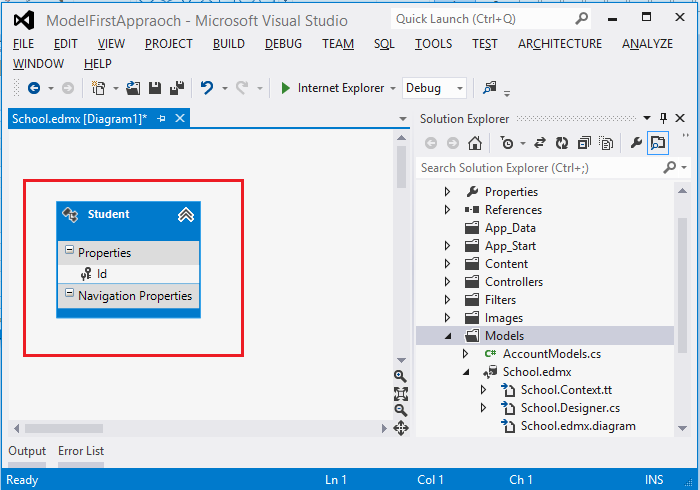
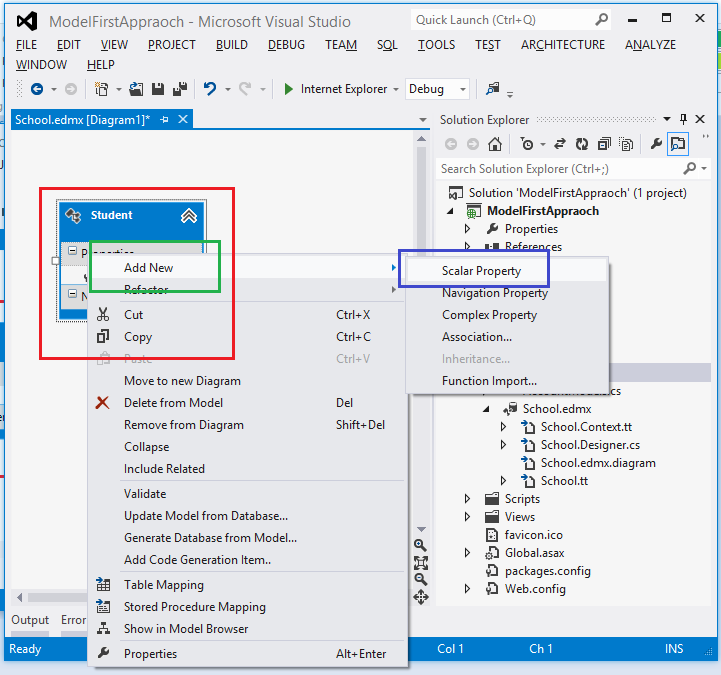
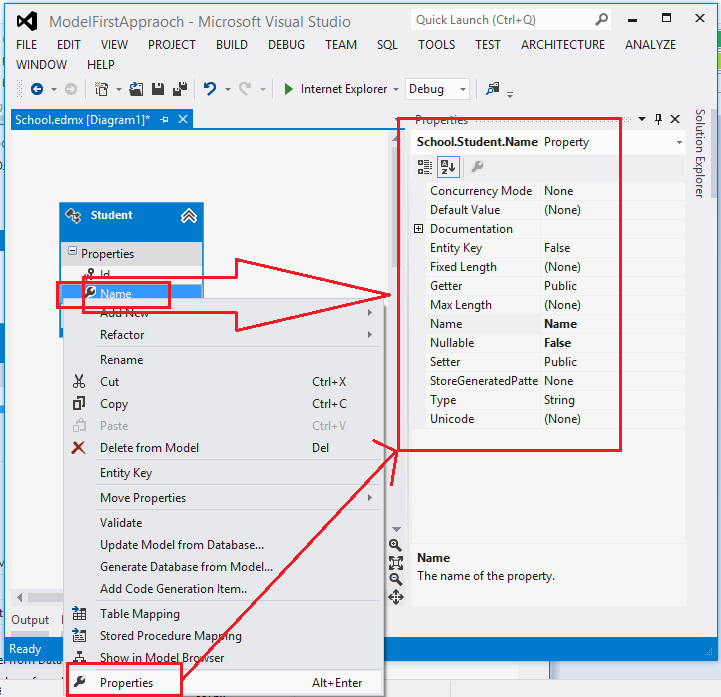
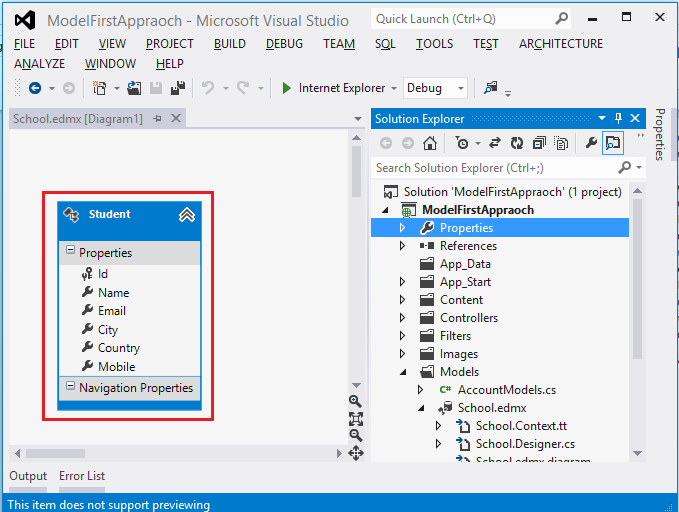
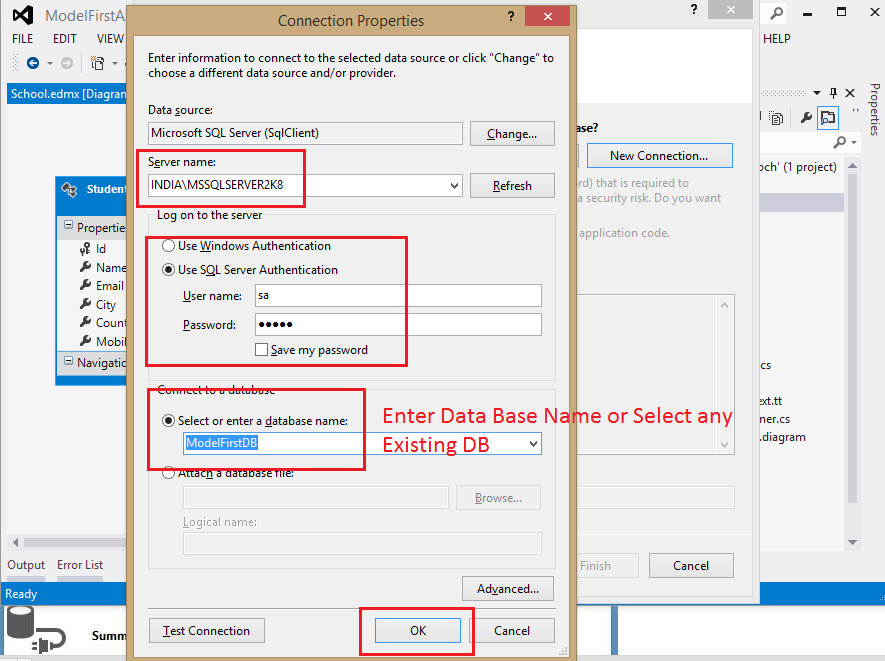
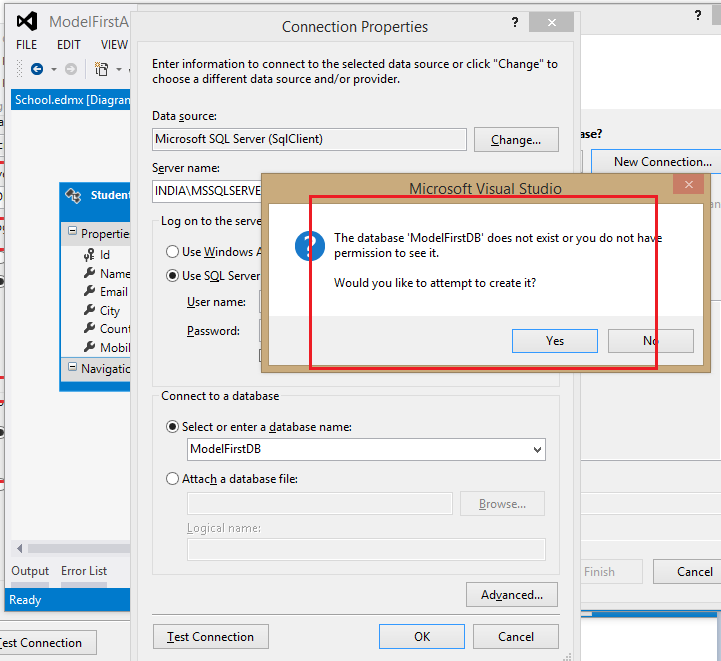
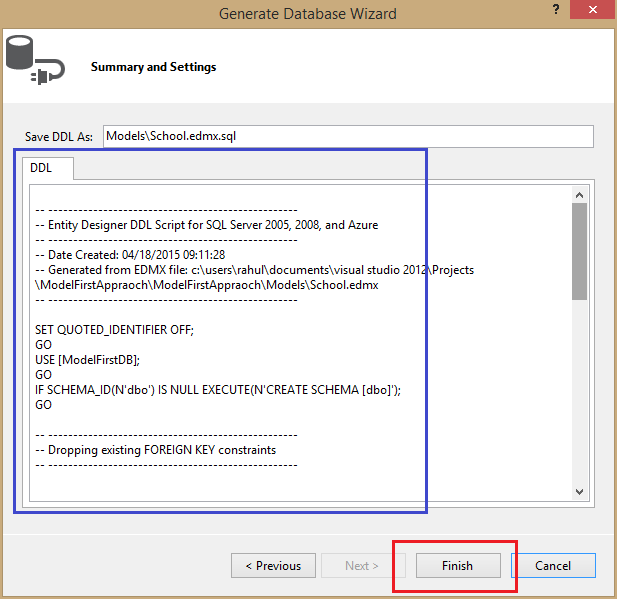
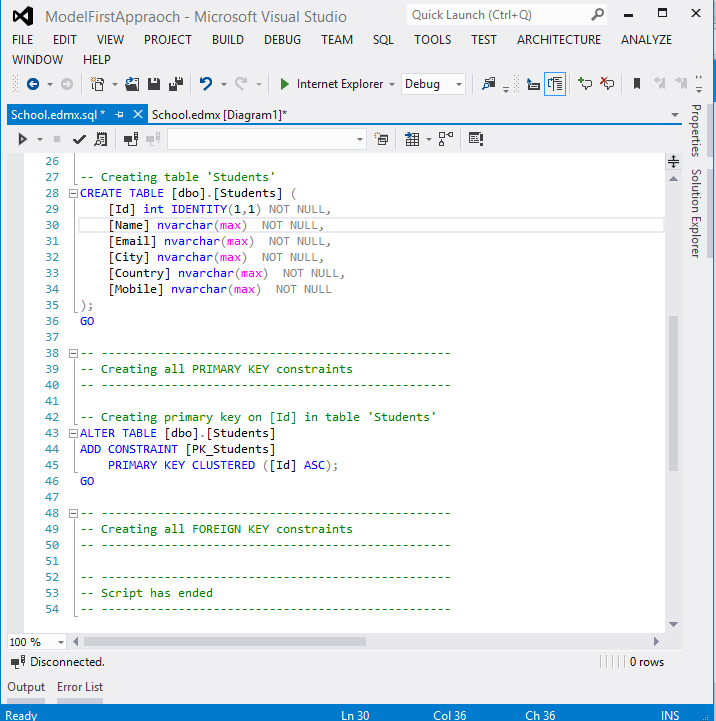
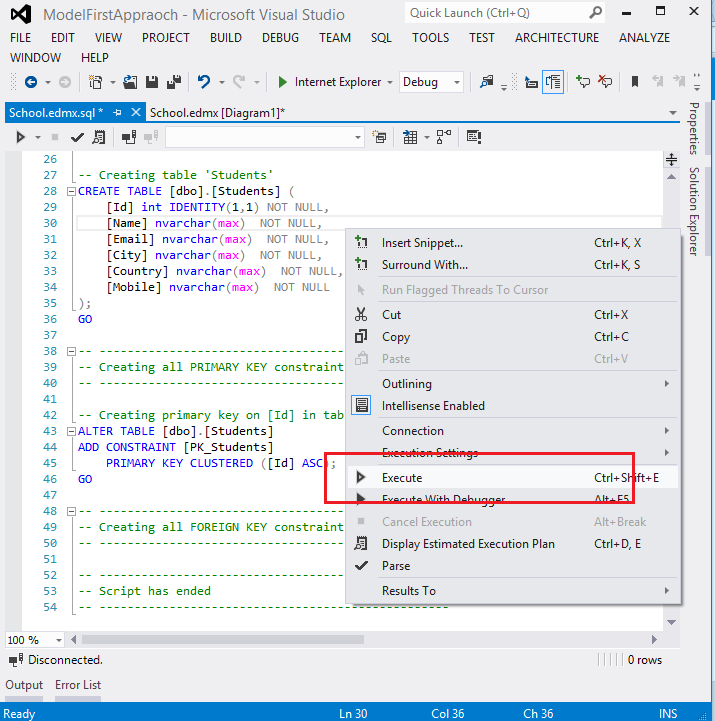
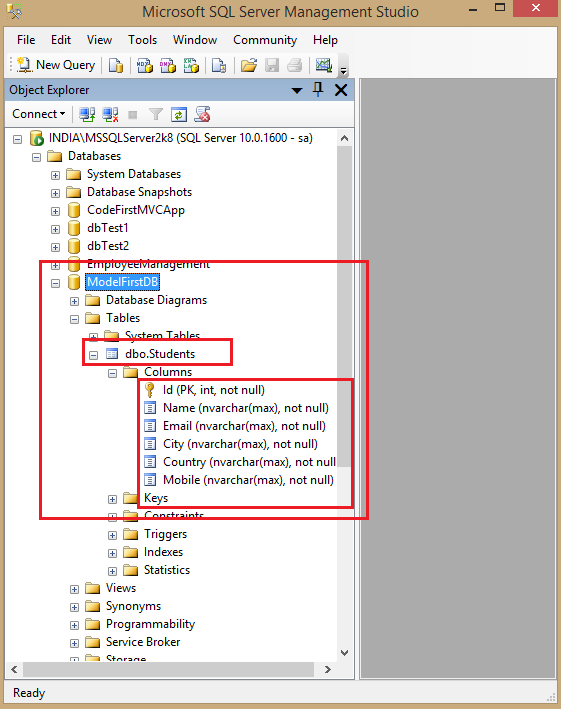
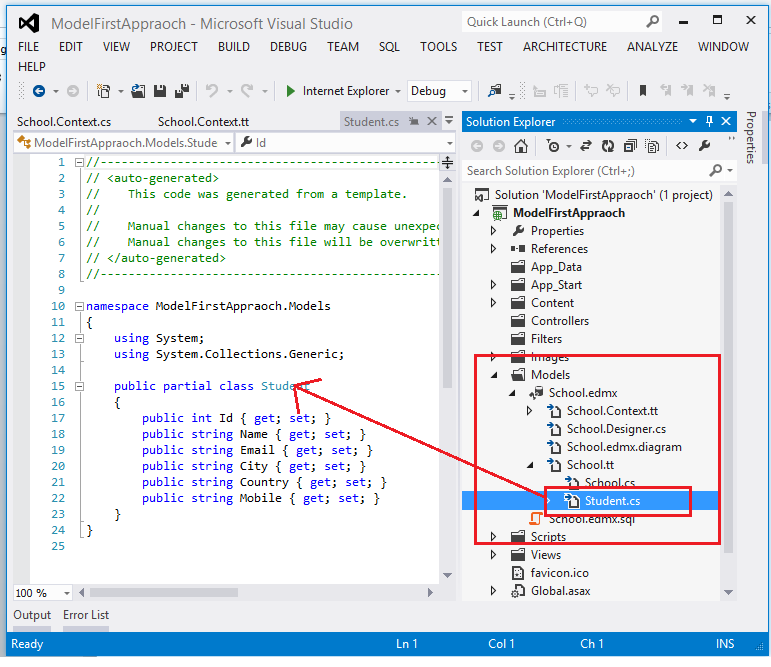
Now, you will see a much cleaner code, as we saw in the "Code First" article. Look at the screen:

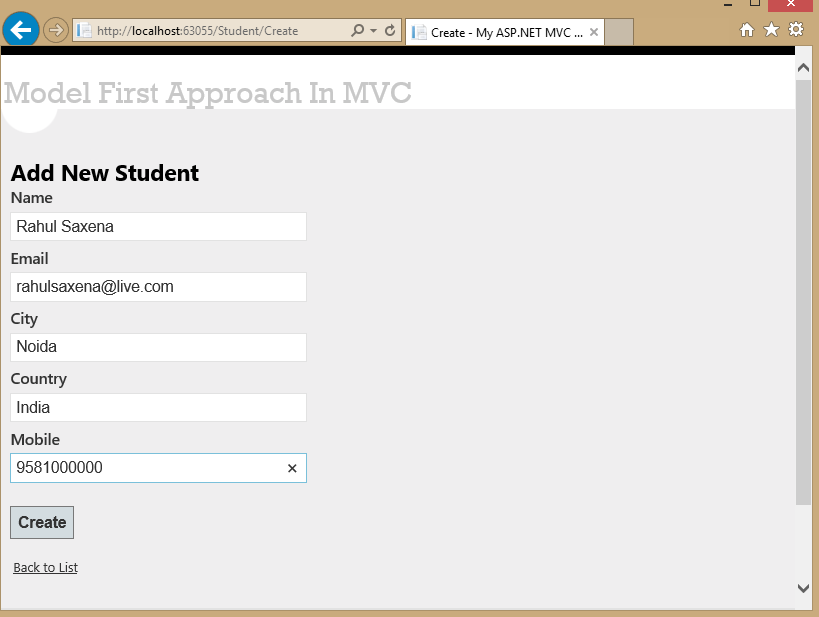
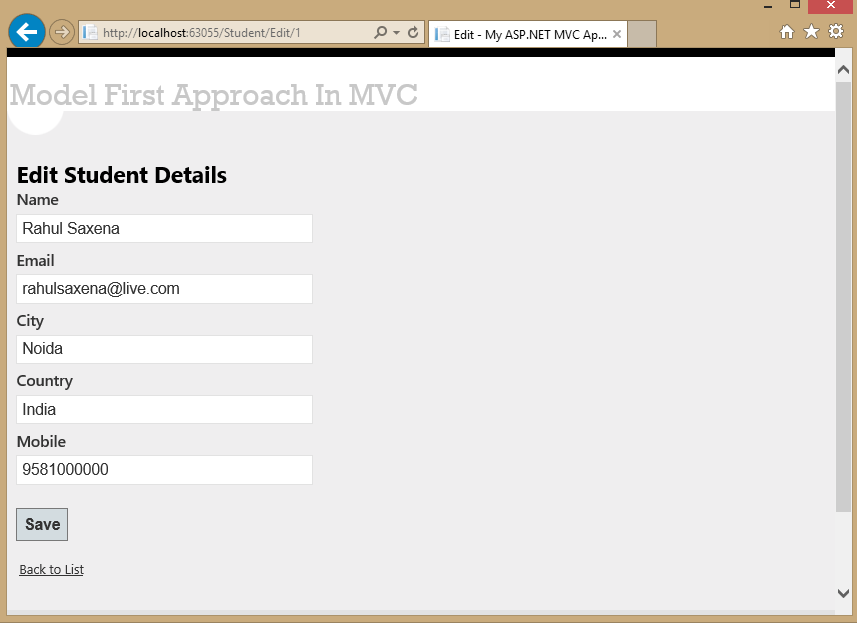
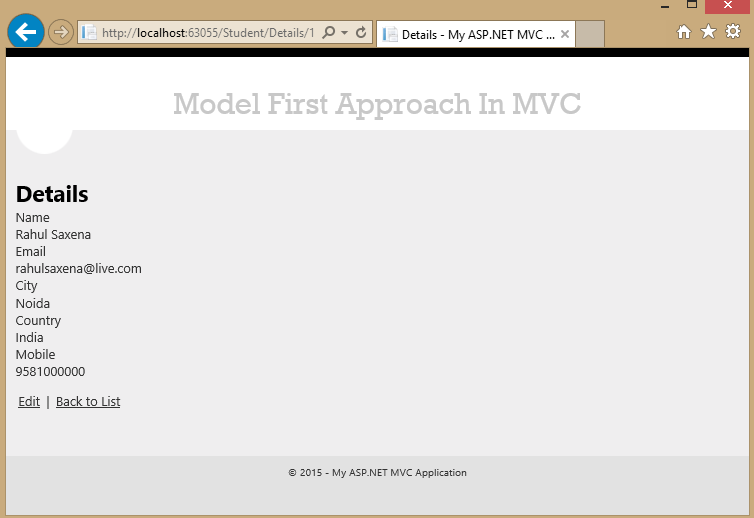
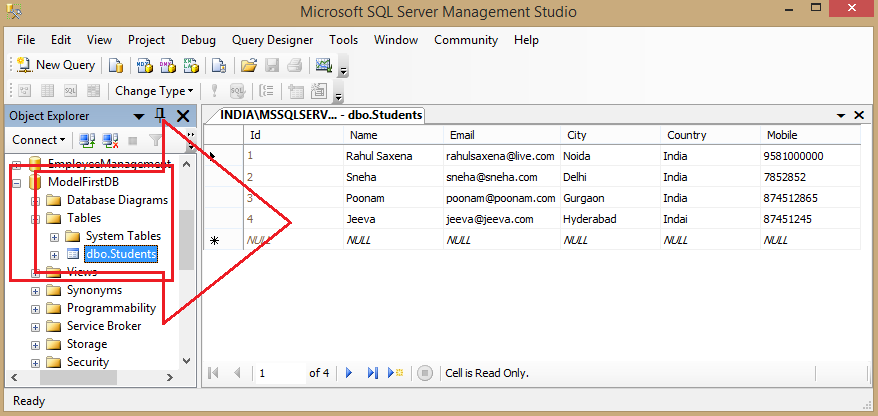


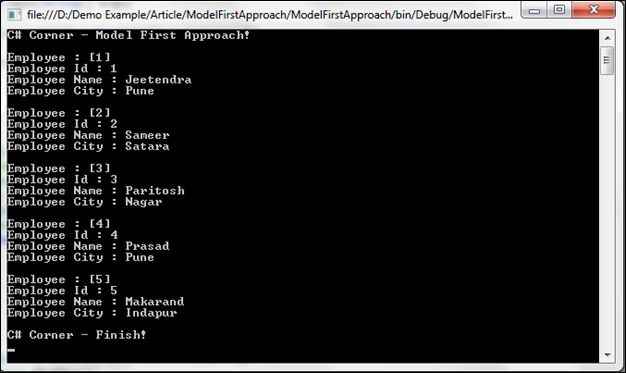
Now we are ready to add Controllers and appropriate Views in the project which will be exactly the same as we did in Step 3 to Step 7 of the article "[Code First Approach in Entity Framework](https://www.c-sharpcorner.com/UploadFile/abhikumarvatsa/code-first-approach-in-entity-framework/)". So, go through the article and do the remaining code. I know I'm quitting here because it is better to write another article than duplicating something.

# Model First Approach in MVC Using Entity Framework

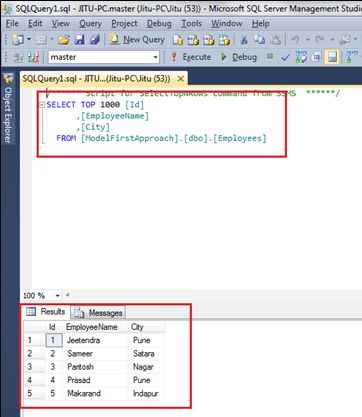
**Entity Framework**  
  
The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write.  
  
The following Image will explain Entity Framework architecture.  
  
  
Image 1.  
  
**O/RM**  
  
ORM is a tool for storing data from domain objects to relational database like Microsoft SQL Server, in an automated way, without much programming. O/RM includes three main parts: Domain class objects, Relational database objects and Mapping information on how domain objects map to relational database objects (tables, views and Stored Procedures). ORM allows us to keep our database design separate from our domain class design.  
  
**Model First Approach step-by-step**  
Model first is the domain modelling approach in Entity Framework. It enables you to create a model's Entities, relationships and inheritance hierarchies on the design surface of an empty model (.edmx file) by using entity designer and then create the database from it.  
  
Now open Visual Studio then select "File" -> "New" -> "Project...".  
  
  
Image 2.  
  
  
Image 3.  
  
Now right-click on the Model Folder then select "Add" -> "ADO.NET Entity Data Model".  
  
  
Image 4.  
  
Graphical user interface, text, application

Description automatically generated  
Image 5.  
  
Select Empty model.  
  
  
Image 6.  
  
Now right-click then select "Add New" -> "Entity...".  
  
  
Image 7.  
  
Provide a name then select a Data Type and Set Key.  
  
  
Image 8.  
  
Click OK.  
  
  
Image 9.  
  
Now add more columns to your Entity.  
  
  
Image 10.  
  
It will add a new property or column to your entity. Now you can set the data type and other properties of this newly added column as in the following.  
  
  
Image 11.  
  
Now add all the columns (properties) that you want. After this our ADO.NET Entity Data Model Will look as in the following.  
  
  
Image 12.  
  
Now to generate the database from this model. So right-click in your edmx as in the following.  
  
  
Image 13.  
  
  
Image 14.  
  
This will give the following Message then click Yes.  
  
  
Image 15.  
  
This will show the connection string in the next step as in the following.  
  
  
Image 16.  
  
Click Next. That will generate a DB Script.  
  
  
Image 17.  
  
  
Image 18.  
  
  
Image 19.  
  
After the command is executed successfully check your database.  
  
  
Image 20.  
  
Now generate a class. So click on Models then select School.edmx then right-click on School.tt then select Run Custom Tool.  
  
  
Image 21.  
  
  
Image 22.  
  
Now to create CRUD operations. So right-click on Controller then select Add -> Controller.  
  
Graphical user interface, text, application, email

Description automatically generated  
Image 23.  
  
It will CRUD Views also as in the following.  
  
  
Image 24.  
  
Now run your application.  
  
Add New Student.  
  
  
Image 25.  
  
Showing All Students Records.  
  
  
Image 26.  
  
Edit any Student record.  
  
  
Image 27.  
  
Showing details of Student.  
  
  
Image 28.  
  
Delete any Student record.  
  
  
Image 29.  
  
Now see your database.  
  
  
Image 30.



Checking the data is saved in the database or not, see the below image.



Great, your second Entity Framework approach - Model First Approach created successfully!