There are 3 approaches through which Entity framework is implemented.

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

Of these Database first and Code First  are the most used ones. In this article I will not discuss model first approach.

First let us understand what are code first and database first

**Code First Approach**In code first approach we will first create entity classes with properties defined in it. Entity framework will create the database and tables based on the entity classes defined. So database is generated from the code. When the dot net code is run database will get created.

**Advantages**

1. You can create  the database and tables from your business objects.
2. You can specify which related collections are to be eager loaded, or not be serialized at all.
3. Database version control.
4. Good for small applications.

**Disadvantages**

1. You have to write everything related to database in the visual studio code.
2. For stored procedures you have to map stored procedure using Fluent API and write Stored Procedure inside the code.
3. If you want to change anything in the database tables you to make changes in the entity classes in the code file and run the update-database from the  package manager console.
4. Not preferred for Data intensive applications.

**Database First Approach**In this approach Database and tables are created first. Then you create entity Data Model using the created database.

**Advantages**

1. Simple to create the data model
2. Graphical  user interface.
3. Mapping and creation of keys and relationships are easy as you need not have to write any code .
4. Preferred for data intense and large applications

**Disadvantages**

1. Using an existing database to generate a .edmx model file and the associated code models results in a giant pile of auto generated code.
2. When you need to add any functionality to generated model you have to extend the model class generated.

Choosing the appropriate approach is purely based on the applications you are developing.

**Entity Framework 6**

Entity Framework 6 (EF6) is a tried and tested data access technology. It was first released in 2008, as part of .NET Framework 3.5 SP1 and Visual Studio 2008 SP1. Starting with the 4.1 release it has shipped as the [EntityFramework](https://www.nuget.org/packages/EntityFramework/) NuGet package. EF6 runs on the .NET Framework 4.x, which means it runs only on Windows.

EF6 continues to be a supported product, and will continue to see bug fixes and minor improvements.

**Entity Framework Core**

Entity Framework Core (EF Core) is a complete rewrite of EF6 that was first released in 2016. It ships in Nuget packages, the main one being [Microsoft.EntityFrameworkCore](https://www.nuget.org/packages/Microsoft.EntityFrameworkCore/). EF Core is a cross-platform product that can run on .NET Core or .NET Framework.

EF Core was designed to provide a developer experience similar to EF6. Most of the top-level APIs remain the same, so EF Core will feel familiar to developers who have used EF6.

--------------------------------------------------------------------------------------------

## Introduction

We all know Databases is heart of any application, and some time its too complex to manage right? Yes. That’s why Microsoft come up with the **Entity Framework** :  which reduce complexity and make easy for developers to write great program without having much more idea of Database. Thats why now-a-days most of Microsoft .Net developers use the .Net Entity Framework for database operation.

### ****What is the Entity framework?****

The Entity framework is an **object-relation mapper**, means - it takes the structure of the database and turns it into objects that the .Net framework can understand. Developers use those object to interact with the database instead of interacting with database directly. It’s possible to perform the full set of Create, Read, Update, and Delete (CRUD) operations using the Entity Framework features.

It has three work flows

1. **Code – first,**
2. **Model-first and**
3. **database first.**

Choosing and using the right approach can save developers a lot of time and effort, especially when interacting with complex database design.

In this Article we will work through the **Code-first work flow** with simple Demo Application.

**Cover part:**

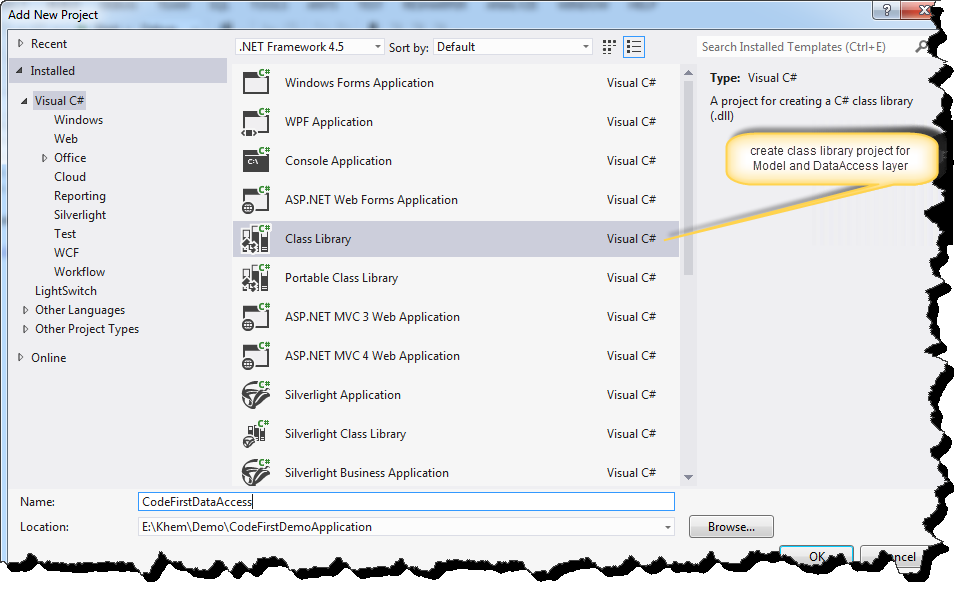
1. Understanding the Code First Work-flow
2. Simple Demonstration on code-frist approach with MVC Application.

## ****Understanding the Code First Work-flow****

The code first approach is introduced in **Entity Framework 4.1**, and is a latest workflow Microsoft has introduced. It lets us transform our coded classes into a database application, which means code first lets us to define our domain model using POCO (plan old CLR object) class rather than using an xml - based EDMX files, which has no dependency with entity framework. Our model classes becomes the domain model, there for we most have to be very conscious in designing our model classes. And the rest work will be done by entity-frame work. This is the beauty of the code first approach where our model classes are become the data models on which Entity framework relies.

**Simple Demonstration on code-frist approach with Asp.Net MVC Application**

1) Create a class library project Named as “**CodeFirstModel**” and solution Named as “**CodeFirstDemoApplication**”. Open VS -> File ->New Project -> Class library project. Refer screen shot below.



And new class named as “Employee.cs”. Project name->right click-> Add-> class.

Hide   Copy Code

public class Employee

{

    public int EmployeeId { get; set; }

    public string Name { get; set; }

    public string Description { get; set; }

    public bool IsEmployeeRetired { get; set; }

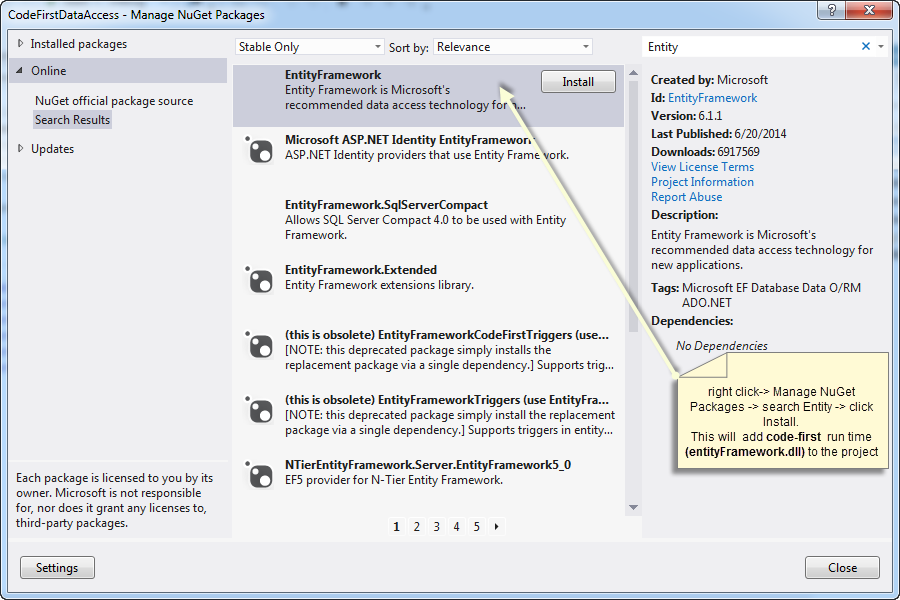
    public string Country { get; set; }

    public string Company { get; set; }

}

Code-first will use this model class to define data model on which entity framework relies. It has convention (we will discuss more in next article) that if it finds a property named **Id**or a property with the combined name of **type name and Id**  (i.e EmployeeId) that property will be automatically configured as the Primary key and not null and also this key marked as auto-increment identity in the table. If it cannot find any property that matches this convention, it will throw an exception at run-time telling there is no key (I.e EntityType “class name” has no key defied. Define the key for this entity type).

2) Create another Class library project and named as “**CodeFirstDataAccess**” : Right click project solution ->  Add -> new Project -> Class library project and **Install Entity framework**form NuGet Package manager. Select Project -> right click -> Manage NuGet Packages -> Search Entity FrameWork -> Install. refer below screen shot.



i) Add the reference of **entity framework dll** and “**CodeFirstModel**” to the current project "**i.e CodeFirstDataAccess**".  
ii) Add a class named as “**DemoEntityContext.cs**” - Then implement **DBContext**and **DBSet**to interact with database and its operations.

Hide   Copy Code

public class DemoEntityContext : DbContext

{

    public DemoEntityContext() : base("name=DbConnectionString") {    }

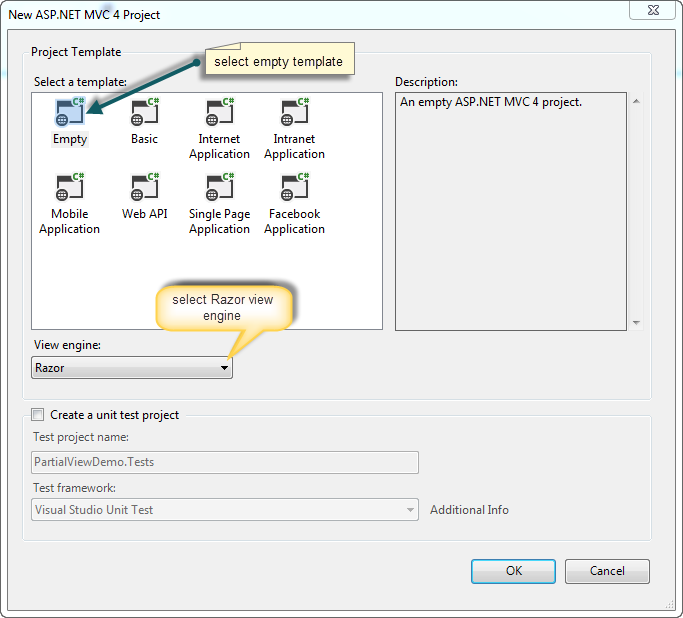
    public DbSet<Employee> Employees { get; set; }

}

This simple class represents the whole data layer for  our application. ”**DBConnectionString**” is the connection string name defined in web.config file(discuss later).

**DBContext**– In simple this class is responsible to interact with database, and also to manage the entity objects during run time, which includes populating objects with data from a database, change tracking, and persisting data to the database.  
  
**DBSet**- This class represents an entity set that is used for the create, read, update, and delete operations.

3) Add an Empty MVC web application named as “CodeFirstWebApp” which will use the above datalayer for db operation. Right click on project solutions -> Add -> new Project -> ASP.NET MVC4 Web application. refer screen shot



Enable-migration

Add-migration filename –context

Update-database