**Introduction to Entity Framework Core**

**Introduction to Entity Framework Core**

In this article, I am going to give you an overview of **Entity Framework Core**. The Entity Framework Core which is also known as EF Core is the latest version of Entity Framework and completely rewrites from the ground up. As part of this article, we are going to discuss the following pointers.

1. **What is Entity Framework Core?**
2. **What is ORM?**
3. **Why do we need to use an ORM?**
4. **EF Core Development Approaches.**
5. **EF Core Code First Approach**
6. **Entity Framework Core Database First Approach**
7. **EF Core Database Providers**
8. **Why need to use Entity Framework Core over EF 6.x?**

**What is Entity Framework Core?**

Entity Framework (EF) Core is an ORM (Object-Relational Mapper) Framework for data access in .Net. It was released along with .NET Core and is an extensible, lightweight, Open Source, and cross-platform version of Entity Framework data access technology. It works on multiple operating systems like Windows, Mac, and Linus.

**What is ORM?**

The term ORM stands for Object-Relational Mapper and it automatically creates classes based on database tables and the vice versa is also true. That is, it can also generate the necessary SQL to create the database based on the classes.

As a developer, we mostly work with the application business objects and the ORM Framework generates the necessary SQL (to perform the CRUD operation) that the underlying database can understand. So, in simple words, we can say that the ORM Framework eliminates the need for most of the data access code that as a developer we generally write.

**Why do we need to use an ORM?**

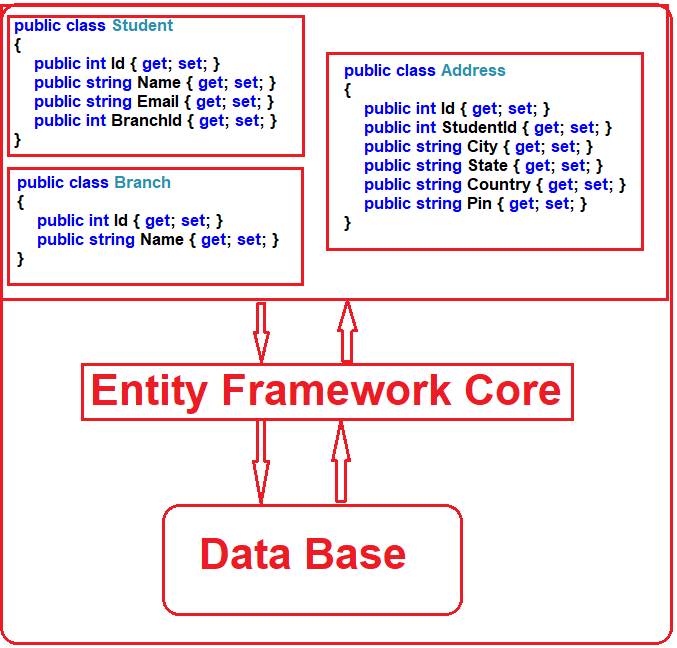
Let us understand why we need to use the ORM Framework with an example.

Suppose we want to develop an application to manage the students of a college. In order to do this, we may need to create classes such as Student, Department, Address, etc. Technically we called these classes as Domain classes.

Without using ORM Framework like Entity Framework or EF Core, we have to write lots of data access code to perform the CRUD operations i.e. store and retrieve the Student, Department and Address data from the underlying database tables.

For example, in order to perform CRUD operations i.e. read, insert, update or delete from a database table, we generally need to write custom code in our application to generate the required SQL statements which can be understood by the underlying database. Again, when we want to read the data from the database into our application, then also we have to write some custom code to map the database data to our model classes like Student, Department, Address, etc. This is a very common task as a developer for us that we do almost in every application.

An ORM Framework like Entity framework or EF Core can do all of the above for us and saves a lot of time if we provide the necessary required information to the ORM Framework. The ORM Framework sits between our application code and the Database. It eliminates the need for most of the custom data-access code that we usually write without an ORM.



**EF Core Development Approaches:**

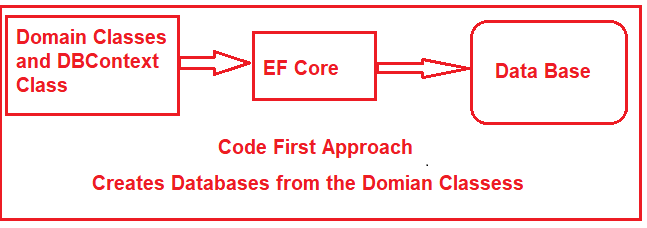
Entity Framework Core supports two development approaches. They are as follows:

1. **Code-First Approach**
2. **Database-First Approach**

EF Core mainly targets the code-first approach and provides little support for the database-first approach at the moment.

**EF Core Code First Approach:**

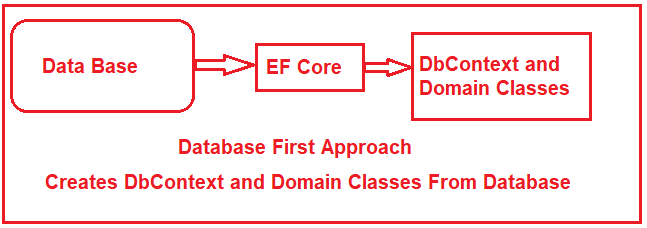
In the EF Core Code First Approach, first, we need to create our application domain classes such as Student, Branch, Address, etc. and a special class that derives from Entity Framework DbContext class. Then based on the application domain classes and DBContext class, the EF Core creates the database and related tables.



Out of the box, in the code-first approach, the EF Core API creates the database and tables using migration based on the default conventions and configuration. If you want then you can also change the default conventions used to create the database and its related tables. This approach is useful in Domain-Driven Design (DDD).

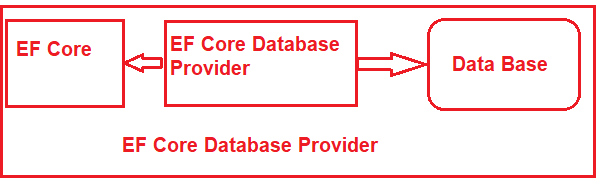
**EF Core Database First Approach:**

If you have an existing database and database tables are already there, then you need to use the EF Core Database First Approach. In the database-first approach, the EF Core creates the DBContext and Domain classes based on the existing database schema using EF Core Command.



**EF Core Database Providers:**

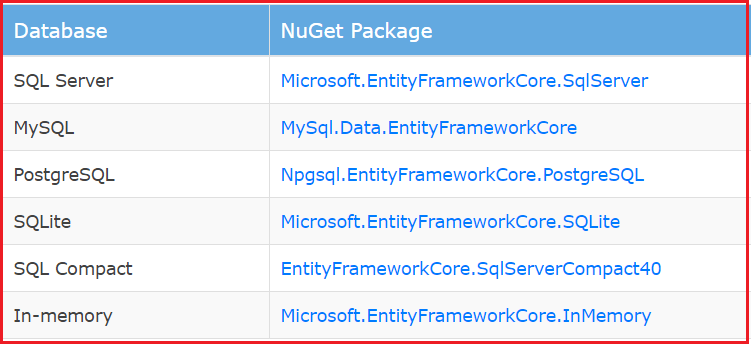
The EF Core supports both relational and non-relational databases and this is possible due to database providers. The database providers are available as NuGet packages. The Database Provider sits between the EF Core and the database it supports.



The EF Core database provider usually contains the functionality specific to the database it supports.

1. Functionality that is common to all the databases is in the EF Core component.
2. Functionality that is specific to a database, for example, Microsoft SQL Server-specific functionality is with-in the SQL Server provider for EF Core.

The following image shows the database providers and the NuGet packages for EF Core.



For the complete list of EF Core Database Providers, please visit the following URL.

[**https://docs.microsoft.com/en-us/ef/core/providers/**](https://docs.microsoft.com/en-us/ef/core/providers/)

We will discuss Database providers in detail in our upcoming articles.

**Why need to use Entity Framework Core over EF 6.x?**

The EF 6.x is a stable and fully tested ORM technology and used in many applications. EF Core also provides an experience similar to EF 6.x. However, it is built upon a new set of components. The application targets to .NET Core e.g. ASP.NET Core applications. Universal Windows Platform (UWP) Applications cannot use EF 6.x as this requires the complete .NET Framework. EF Core is more suitable for such applications. Although both of these releases provide similar features, EF Core does not have the following features:

1. Spatial Type
2. Complex/value types
3. EDMX Model format
4. Lazy loading
5. Stored Procedures

The complete list of features not available in EF Core and a side-by-side comparison with EF 6.x can be found at [Feature Comparison](https://docs.microsoft.com/en-us/ef/efcore-and-ef6/features). The EF Core does not support all the features provided by Entity Framework 6.x. However, following are some reasons to use EF Core:

1. Simple to get started
2. It supports NoSQL Databases along with relational databases
3. It fits nicely in ASP.NET MVC Core setup
4. Integrates well with LINQ
5. Possible to use on Linux and Mac since it is based on .NET Core along with Windows.

**How to Install Entity Framework Core**

**How to Install Entity Framework Core in Visual Studio**

In this article, I am going to discuss How to Install Entity Framework Core in Visual Studio step by step in different types of applications. Please read our previous article where we discussed the [**introduction part of Entity Framework Core**](https://dotnettutorials.net/lesson/entity-framework-core/).

**Note:** The Entity Framework Core can be used with .NET Core applications as well as .NET 4.6 based applications. Here, in this article, I will show you how to install EF Core in .NET Core applications using Visual Studio 2017.

The Entity Framework Core is not a part of .NET Core and standard .NET framework. It is available as a NuGet package. Depending on how you have set your project, you may have already installed the Entity Framework Core.

Depending on the project complexity or requirement, you can develop a web application as:

1. **Single Layer Web Application**
2. **Multi-Layer Web Application**

**Single Layer Web Application**

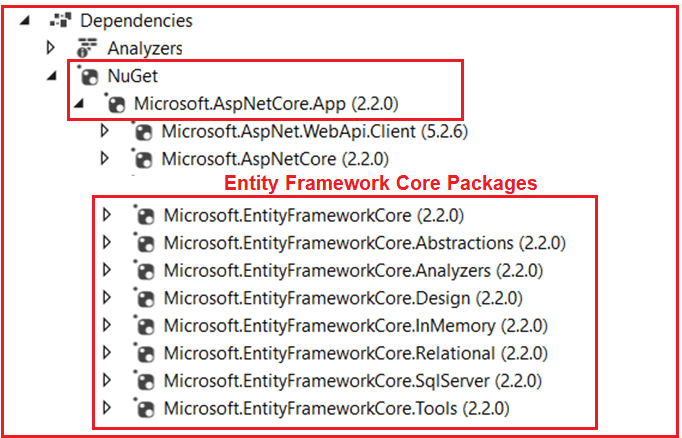
When you develop a small project, then you may have the presentation, business and data access layer all in one project. So when we created a web application project using ASP.NET Core 2.1 or higher, then by default the Entity Framework Core already installed.

**Entity Framework Core in ASP.NET Core Web Application Project**

When we create an ASP.NET Core Web application using ASP.NET Core 2.1 or higher version, then by default the following NuGet package installed.

**Microsoft.AspNetCore.App**

The above package is called a **Meta package**. A Meta package does not hold any content of its own rather it holds a list of other packages. You can find this Meta package, in the Solution Explorer under the NuGet section. When you expand the Meta package, then you can find all the dependencies i.e. other packages. In the dependencies, once you scrolled down then you will find the Entity Framework Core NuGet packages already installed as shown in the below image.



So the point that you need to remember is, when you create an ASP.Net Core web application using ASP.NET Core Version 2.1 or later, then by default you will have Entity Framework Core installed as part of the meta-package.

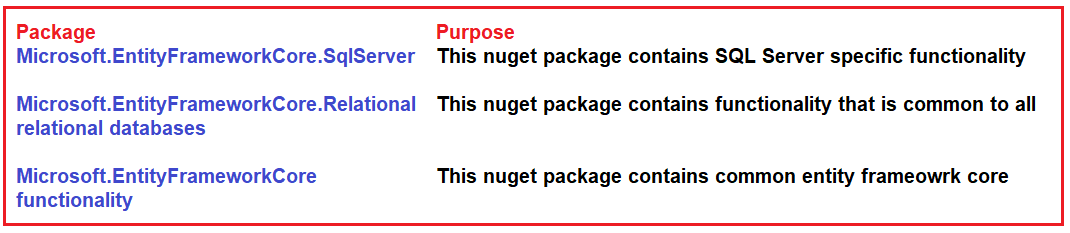
**Multi-Layer Web Application:**

When we want to develop a large application, we usually have at least the following 3 layers

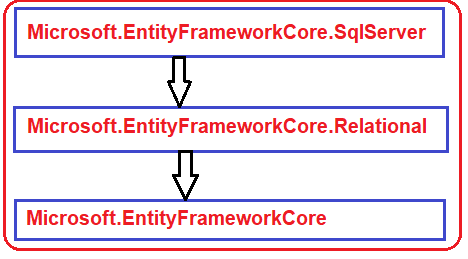
1. **Presentation Layer**
2. **Business Logic Layer**
3. **Data Access Layer**

These layers are implemented as separate projects. And the Entity Framework Core is usually required in the Data Access Layer project as this layer is going to interact with the database. The Data Access Layer project is usually a class library project and a class library project by default does not usually have the meta-package referenced. As it does not have the meta-package installed, so this means, the Entity Framework Core is not installed for the Data Access Layer project.

In order to install the Entity Framework Core to interact with the SQL server database, you need to install the following NuGet packages.



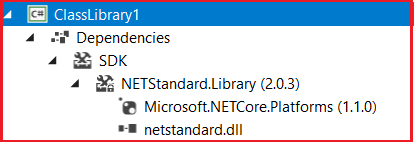
1. **Microsoft.EntityFrameworkCore.SqlServer** has a dependency on **Microsoft.EntityFrameworkCore.Relational** package.
2. Similarly, **Microsoft.EntityFrameworkCore.Relational** package has a dependency on **Microsoft.EntityFrameworkCore** package.
3. **Microsoft.EntityFrameworkCore**package has a dependency on several other packages.



**Note:** When you install **Microsoft.EntityFrameworkCore.SqlServer** package, then it also automatically installs all the other dependency NuGet packages.

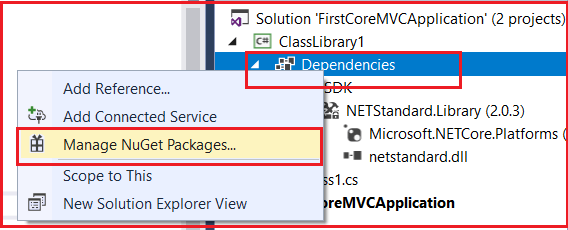
**Installing Entity Framework Core in the Class Library Project:**

Create a class library Project. Once you create the class library project expand the dependencies node as shown below.

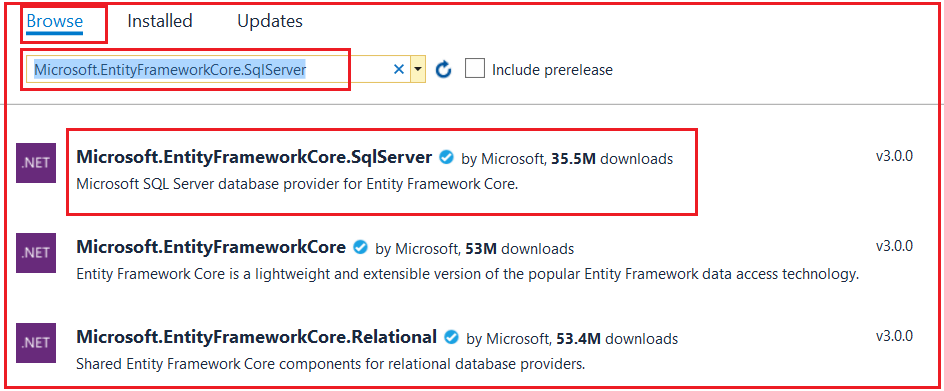


As you can see in the above image, we don’t have the meta-package installed and hence we don’t have the entity framework core installed in our application.

In order to install entity framework core, Right-click on the *“***Dependencies***“* node in Solution Explorer and then select *“***Manage NuGet Packages***“* from the context menu as shown in the below image.



This will open the NuGet Package Manager interface. Click on the Browse tab and search for **Microsoft.EntityFrameworkCore.SqlServe**r in the search box at the top left corner as shown in the below image. Based on the asp.net core version installed the entity framework core version compatibility with your .net core framework.



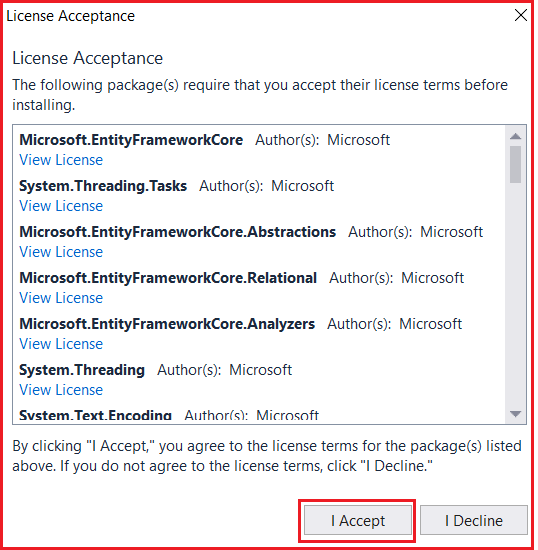
**Note:** We want to install Entity Framework Core to interact with the SQL Server database, so we installed the **Microsoft.EntityFrameworkCore.SqlServer** package. This package is usually called the Database provider package.

If you are using a different database other than SQL server, then you need to install that database provider-specific NuGet package. For example, if you want to use MySQL as your database, then install **Pomelo.EntityFrameworkCore.MySql** database provider package. Along the same lines, if you want to use PostgreSQL as your database, then use **Npgsql.EntityFrameworkCore.PostgreSQL** database provider package.

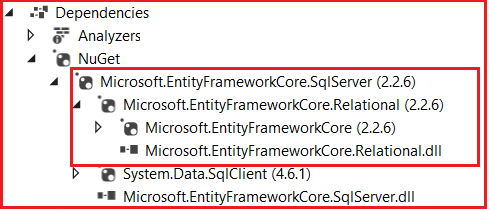
You can find all provider-specific NuGet packages on the following MSDN page.

[**https://docs.microsoft.com/en-us/ef/core/providers/**](https://docs.microsoft.com/en-us/ef/core/providers/)

When you are installing the entity framework it will ask you to accept the license terms associated with the packages that are going to be installed as shown in the image below. Accept the license.



Once the entity framework core installed, you will find the packages in the Meta package as shown in the below image.



**Alternatively,** you can also use the Package Manager Console to install the Entity Framework Core package. Go to **Tools -> NuGet Package Manager -> Package Manager Console** and execute the following command to install the SQL Server provider package.

**PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer**

**DbContext in Entity Framework Core**

**DbContext in Entity Framework Core**

In this article, I am going to discuss **DbContext in Entity Framework Core** in detail. Please read our previous article where we discussed [**how to install entity framework core in visual studio**](https://dotnettutorials.net/lesson/install-entity-framework-core/) step by step. DbContext class is one of the important classes in entity framework core and at the end of this article, you will understand the significance of the **DbContext** class in Entity Framework Core. As part of this, we will discuss the following pointers.

1. **What is a DbContext class?**
2. **How to create and use the DbContext class?**
3. **DbContextOptions class in Entity Framework Core**
4. **Entity Framework Core DbSet**

**What is a DbContext class?**

The DBContext class is basically used by our application to interact with the underlying database. That means this class is used to manage the database connection as well as also used to perform CRUD operation with the underlying database.

**How to create and use the DbContext class in Entity Framework Core?**

In order to use the **DbContext** class in your application, you need to create a class derives from the **DbContext** class. The **DbContext** class present is in **Microsoft.EntityFrameworkCore** namespace.

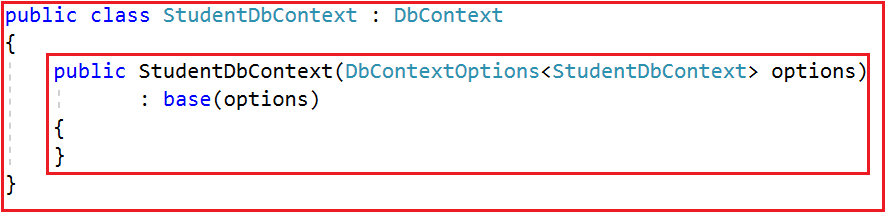
So, within the models’ folder create a class file with the name **StudentDbContext** and then inherit from the class from **DbContext** class as shown in the below image. You can provide any name for your **DbContext** class, as I am going to develop an application to manage the students, so I gave the name as **StudentDbContext**.



**DbContextOptions class in Entity Framework Core:**

In order to perform any useful task by the **DbContext** class, we need an instance of the **DbContextOptions**class. The instance of the **DbContextOptions** carries all the required configuration information such as the connection string, database provider, etc.

To pass the **DbContextOptions**instance we need to use the constructor of **StudentDbContext** class as shown in the image below.



In our next article, we will discuss the DbContextOptions class in detail.

**Entity Framework Core DbSet:**

The entity framework code **DbContext**class includes a property i.e. **DbSet<TEntity>** for each entity in your application. Let us understand this with an example. Create three entities such as **Student, Branch and Address** within the Models folder as shown below.

**namespace** *FirstCoreMVCApplication.Models*

**{**

**public** **class** Student

**{**

**public** **int** Id **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**public** string Email **{** **get**; **set**; **}**

**public** **int** BranchId **{** **get**; **set**; **}**

**}**

**public** **class** Branch

**{**

**public** **int** Id **{** **get**; **set**; **}**

**public** string Name **{** **get**; **set**; **}**

**}**

**public** **class** Address

**{**

**public** **int** Id **{** **get**; **set**; **}**

**public** **int** StudentId **{** **get**; **set**; **}**

**public** string City **{** **get**; **set**; **}**

**public** string State **{** **get**; **set**; **}**

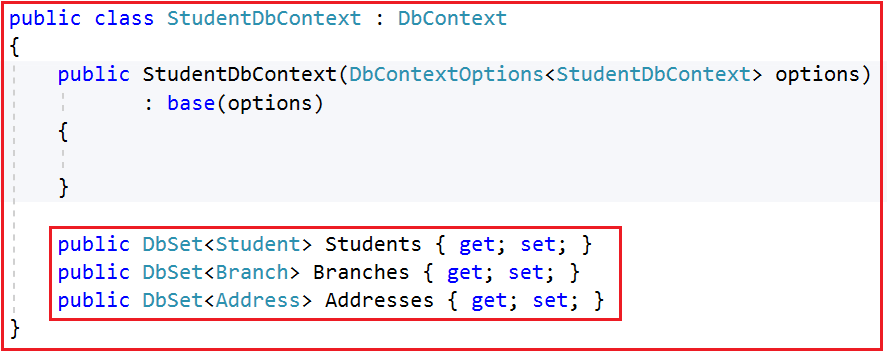
**public** string Country **{** **get**; **set**; **}**

**public** string Pin **{** **get**; **set**; **}**

**}**

**}**

As our application contains three entities, so in our **StudentDbContext**class, we need to have three **DbSet** Properties as shown below.



We will use the above DbSet properties such as **Students, Branches, and Addresses** to perform CRUD Operations.

So, the complete code of the **StudentDbContex** class is given below.

**using** *Microsoft.EntityFrameworkCore;*

**namespace** *FirstCoreMVCApplication.Models*

**{**

**public** **class** StudentDbContext : DbContext

**{**

**public** StudentDbContext**(**DbContextOptions**<**StudentDbContext**>** options**)**

: **base(**options**)**

**{**

**}**

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

**public** DbSet**<**Branch**>** Branches **{** **get**; **set**; **}**

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

**}**

**}**

**So, in Short,**

The **DbContext in Entity Framework Core** perform the following tasks:

1. **Manage database connection**
2. **Configure model & relationship**
3. **Querying database**
4. **Saving data to the database**
5. **Configure change tracking**
6. **Caching**
7. **Transaction management**

In order to connect to a database, we need the database connection string. So, in the next article, I am going to discuss, where to define the connection string and how to use it in Entity Framework Core to interact with the SQL Server Database. Here, in this article, I try to explain the need and use of the DbContext class in Entity Framework. I hope you enjoy this article.