irst Approach In ASP.NET Core MVC With EF ligration

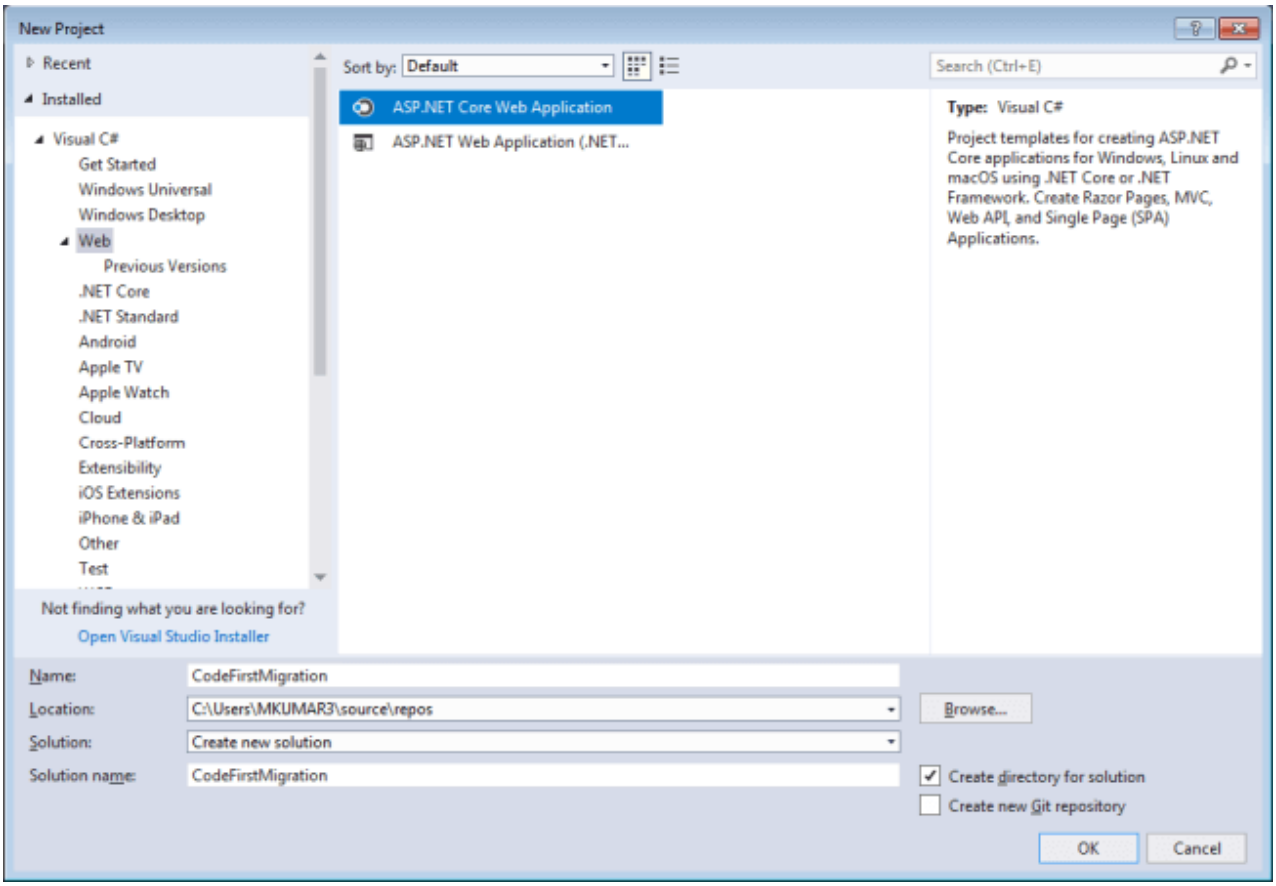
 Mukesh Kumar Updated date Sep 26, 2018

ique which helps us to create a database, migrate and maintain the database e code. From the code, means that you directly maintain the database and les from the .NET Code. It is helpful when you don't have a database ready working with new fresh project and want to create a database and maintain the database directly from your code.

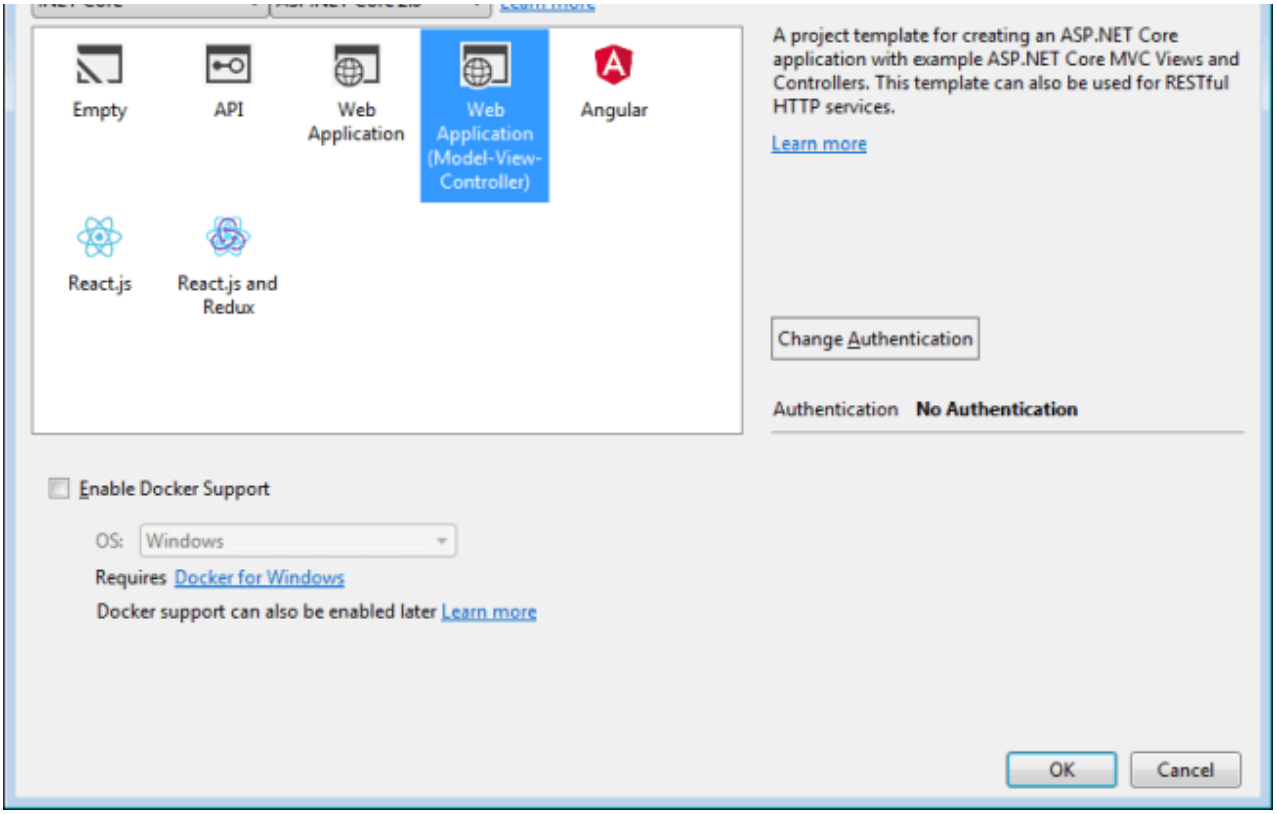
This article will help you to understand what the Code First approach is and how we can achieve it in ASP.NET Core MVC applications using Entity Framework Core migration. Migration always helps us to create, update and sync the database with your model classes. In this demonstration, we will understand the Entity Framework Core migration step by step practically. So, let's create a new application for the demonstration.

Let us jump to Visual Studio 2017 and create a new ASP.NET Core MVC application. You can follow the below steps while creating an ASP.NET Core MVC application in Visual Studio 2017.

1. Open Visual Studio 2017
2. Click to File> New > Project from the Menu
3. In New Project windows, from the left panel, select Installed > Visual C#> Web
4. Select the NET Core Web Application project template from the middle panel



1. Next dialog will appear for the New ASP.NET Core Web Application.
2. Choose the target framework as .NET Core and select the version from the drop-down as NET Core 2.0
3. Select Web Application (Model-View-Controller) as a template
4. Select the Authentication as 'No Authentication'
5. Click OK

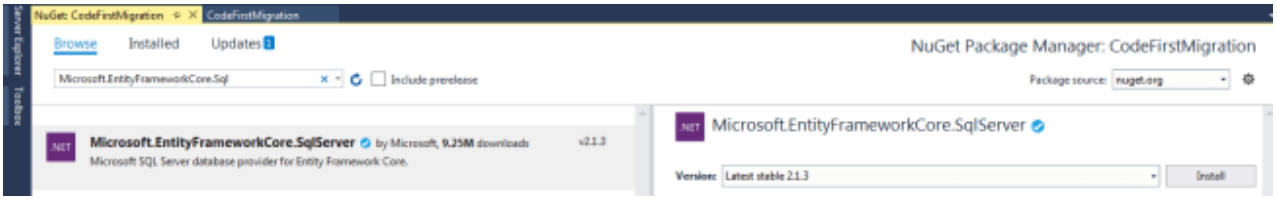


MORE ARTICLES ON ASP.NET CORE WHICH YOU MAY LIKE,

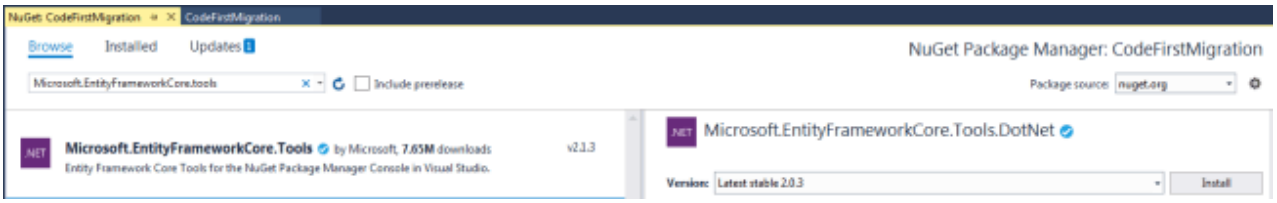
1. First Application in Asp.Net Core MVC 2.0
2. 10 New Features of Asp.Net Core 2.0
3. Publish Asp.Net Core 2.0 Application on IIS
4. Getting started with Razor Pages in Asp.Net Core 2.0
5. NET Core Web API with Oracle Database and Dapper

Now we have the project ready. You can check it to run the application using F5. If everything is well then we can move forward.

I hope the new application is running fine. Therefore, next, we will install some of the required Entity Framework Core packages from the NuGet Package Manager for performing database operations from the code. First, we will install the package like Microsoft.EntityFrameworkCore.SqlServer which will provide classes to connect with SQL Server for CRUD Operation to Entity Framework Core.



update database etc.



So far, we have created one Asp.Net Core MVC application and installed some required Entity Framework Core packages which are required for Code First migration or we can say, these will help us to use Entity Framework Core functionality for working with SQL Server.

So, let's move and create a folder name as 'Context' and create a Model class as 'Employee' in this with the following properties as follows.

```
01. namespace CodeFirstMigration.Context
02. {
03.     public class Employee
04.     {
05.         public int EmployeeId { get; set; }
06.         public string Name { get; set; }
07.         public string Address { get; set; }
08.         public string CompanyName { get; set; }
09.         public string Designation { get; set; }
10.     }
11. }
```

We will create another class inside the Context folder as 'EmployeeDbContext' that will inherit to DbContext class. This class will contain all the model's information which are responsible for creating the tables in the databae. Here we will define our Employee class as DbSet.

```
01. using Microsoft.EntityFrameworkCore;
02.
03. namespace CodeFirstMigration.Context
04. {
05.     public class EmployeeDbContext : DbContext
06.     {
07.         public EmployeeDbContext(DbContextOptions options) : base(options)
08.         {
09.         }
10.
11.         DbSet<Employee> Employees { get; set; }
12.     }
13. }
```

As w, all know that in the Code First approach, we first write the code, which means Model classes, and on the basis of these classes our tables are auto-generated inside

using SQL Windows Authentication but you can use Mixed Authentication and pass the username and password with the connection string. So, you can write connection string inside the appsetting.json file as follows.

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionStrings": {
    "myconn": "server=ABC\\SQLEXPRESS2012; database=EmployeeDB;Trusted_Conne
  }
}
```

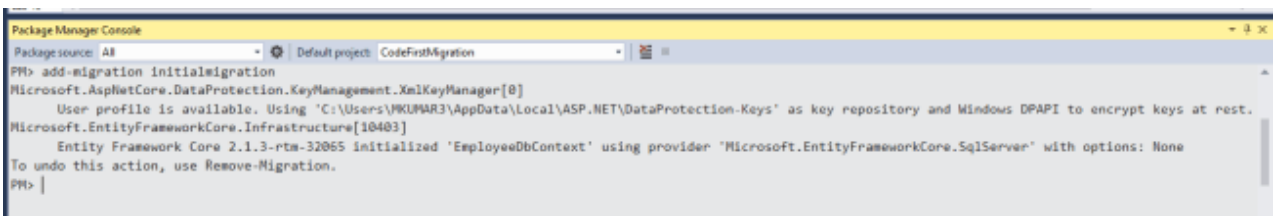
To use this connection string, first, we will change the ConfigureServices method in Startup.cs class as follows.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddDbContext<EmployeeDbContext>
(item => item.UseSqlServer(Configuration.GetConnectionString("myconn")));
}
```

So far, we have done most of the things, like project creation, installing NuGet packages, creating Model classes and setting connection string. So, let's generate the database using Entity Framework Core Migrations.

Open Package Manager Console from the Tools Menu and select the Default project for which you would like to generate migrations code. For this demonstration, we have only a single project as CodeFirstMigration. Therefore, by default, it is a default project.

For creating the migration code, we use ‘add-migration MigrationName’ command. So, let’s perform this operation and see what happens. Therefore, in the Package Manager Console, just type ‘add-migration initialmigration’ command and press Enter.



while executing add migration command with some name. Here you can see the table structure based on your Model (Employee), which is ready to generate the database

```
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;

namespace CodeFirstMigration.Migrations
{
    public partial class initialmigration : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Employees",
                columns: table => new
                {
                    EmployeeId = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy", Sq
                    Name = table.Column<string>(nullable: true),
                    Address = table.Column<string>(nullable: true),
                    CompanyName = table.Column<string>(nullable: true),
                    Designation = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Employees", x => x.EmployeeId);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Employees");
        }
    }
}
```

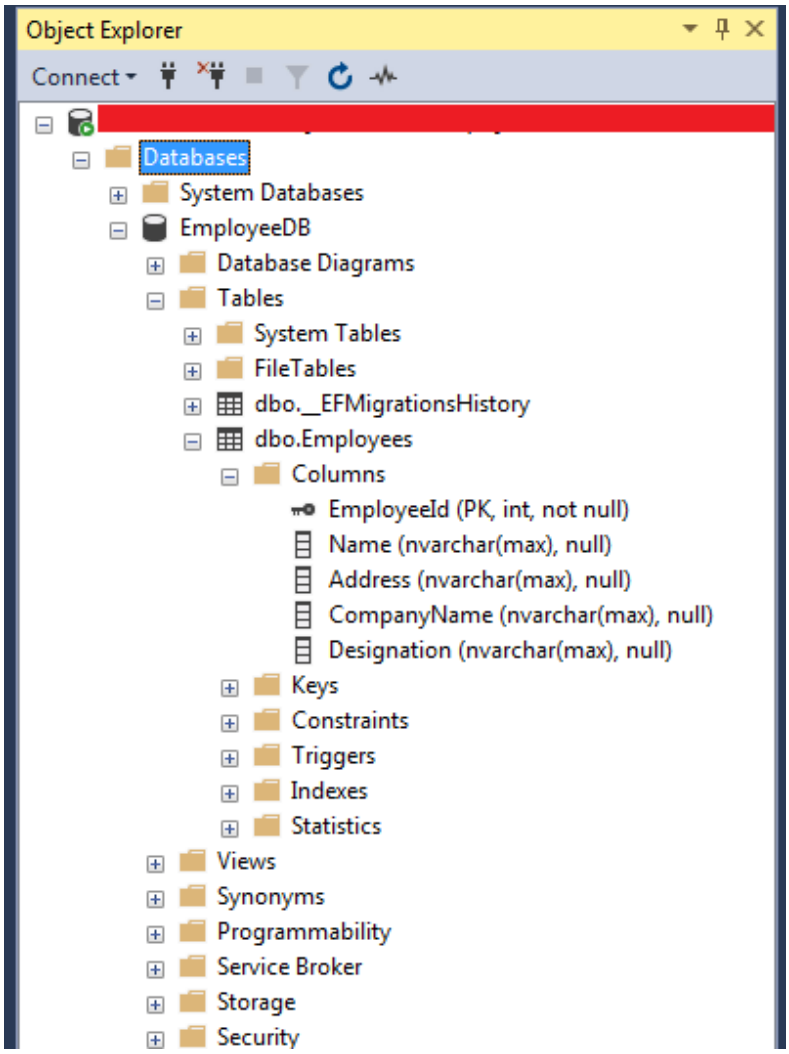
We have only created the migration script which is responsible for creating the database and its table. But we've not created the actual database and tables. So, let's execute the migration script and generate the database and tables. Therefore, executing the migration scripts we have to execute 'update-database' command. So, let's perform it as follows.

For now, we have only one migration script available that is why we are not providing the name of the migration. If we have multiple migration scripts, then we have to provide the name along with command as follows.

update-database migrationname

```
      User profile is available. Using 'C:\Users\MKUMAR3\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.1.3-rtm-32065 initialized 'EmployeeDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (307ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
      CREATE DATABASE [EmployeeDB];
Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (181ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
```

Once the above command executes successfully, we just need to go to SQL Server Management Studio and login with Windows Authentication and see the Database. You will find the database, table and Entity Framework Migration history table as follows.



Now, let's modify the Employee model and add a new property as Salary with float type as follows.

```
01. namespace CodeFirstMigration.Context
02. {
03.     public class Employee
04.     {
05.         public int EmployeeId { get; set; }
06.         public string Name { get; set; }
07.         public string Address { get; set; }
08.         public string CompanyName { get; set; }
```

```
    }
}
```

Move to package manager console and run the following command to add migration, this time we have given the name of migration as 'addedsalary'.

add-migration addedsalary

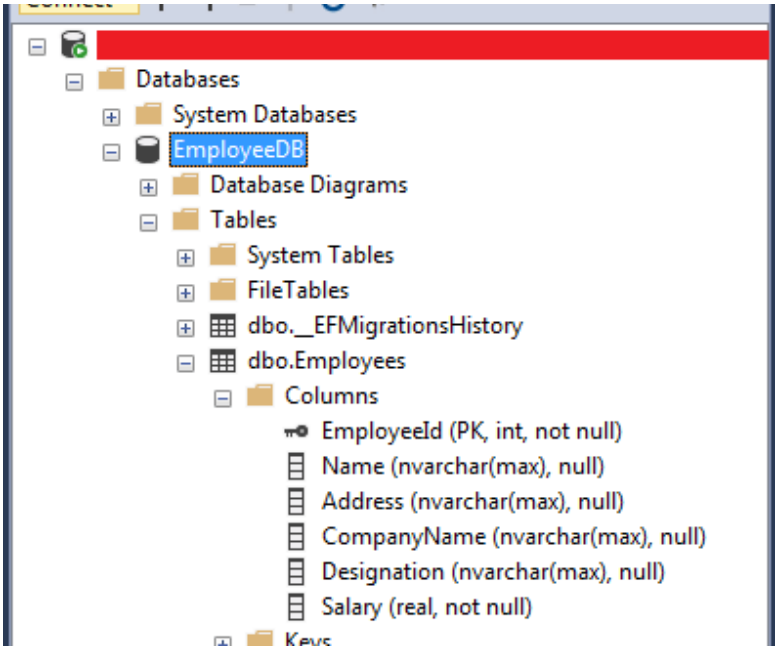
Once the above command execution will be completed, it will create a new class for migration name as follows. Here we can see, migration builder has the configuration for adding a new column as salary.

```
using Microsoft.EntityFrameworkCore.Migrations;

namespace CodeFirstMigration.Migrations
{
    public partial class addedsalary : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<float>(
                name: "Salary",
                table: "Employees",
                nullable: false,
                defaultValue: 0f);
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "Salary",
                table: "Employees");
        }
    }
}
```

For updating the table in the database with the new column as salary, we have to run following command for updating the database.

update-database addedsalary

Once above update database command will execute successfully, just check the database's table. You will find the new column has been added to the Employees table as salary.



Now, let's see how to seed some dummy data into the table. So, move to EmployeeDbContext class and override the DbContext method 'OnModelCreating'. Using the help of the ModelBuilder, we can add some dummy data for the Employees table as follows.

```
using Microsoft.EntityFrameworkCore;

namespace CodeFirstMigration.Context
{
    public class EmployeeDbContext : DbContext
    {
        public EmployeeDbContext(DbContextOptions options) : base(options)
        {
        }

        DbSet<Employee> Employees { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Employee>().HasData(
                new Employee() { EmployeeId = 1, Name = "John", Designation
                new Employee() { EmployeeId = 2, Name = "Chris", Designatior
                new Employee() { EmployeeId = 3, Name = "Mukesh", Designatic
        }
    }
}
```

So, above we have already prepared the data for seeding. Now, let's add it to migration using the following command.

add-migration seeddata

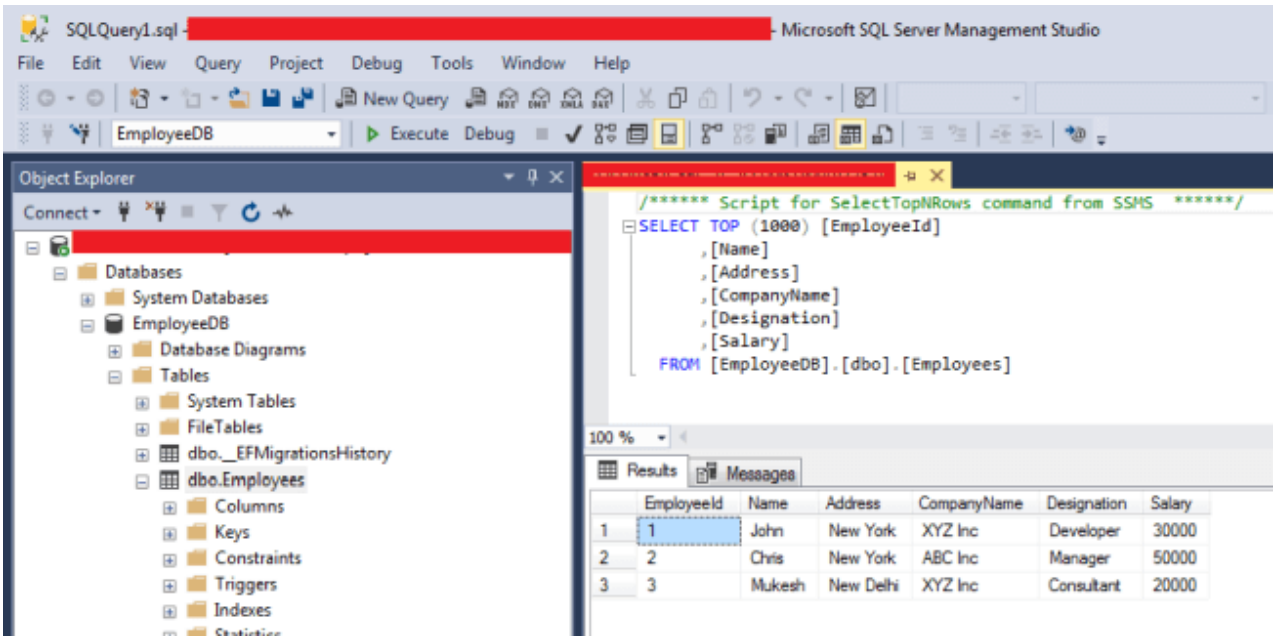
should add, columns and their respective values.

```
using Microsoft.EntityFrameworkCore.Migrations;

namespace CodeFirstMigration.Migrations
{
    public partial class seeddata : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.InsertData(
                table: "Employees",
                columns: new[] { "EmployeeId", "Address", "CompanyName", "De
                values: new object[] { 1, "New York", "XYZ Inc", "Developer'

            migrationBuilder.InsertData(
                table: "Employees",
                columns: new[] { "EmployeeId", "Address", "CompanyName", "De
                values: new object[] { 2, "New York", "ABC Inc", "Manager",

            migrationBuilder.InsertData(
                table: "Employees",
                columns: new[] { "EmployeeId", "Address", "CompanyName", "De
                values: new object[] { 3, "New Delhi", "XYZ Inc", "Consultar
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DeleteData(
                table: "Employees",
                keyColumn: "EmployeeId",
                keyValue: 1);

            migrationBuilder.DeleteData(
                table: "Employees",
                keyColumn: "EmployeeId",
                keyValue: 2);

            migrationBuilder.DeleteData(
                table: "Employees",
                keyColumn: "EmployeeId",
                keyValue: 3);
        }
    }
}
```

So, let's update the database with generated migration class using the following command.

update-database seeddata

migration has been updated into the table.



Conclusion

So, today we have learned how to implement the code first approach in Asp.Net Core MVC project using Entity Framework Core Migration.

I hope this post will help you. Please put your feedback using comment which helps me to improve myself for next post. If you have any doubts please ask your doubts or query in the comment section and If you like this post, please share it with your friends. Thanks

ASP.NET Core | Code First Approach | Code First Approach In ASP.NET Core | EF Core Migration | MVC

> Next Recommended Reading
> ASP.NET Core 2.1 - Implement Entity Framework Core In A Code First Approach

OUR BOOKS

Mukesh Kumar TOP 100

Full Stack Developer | Microsoft MVP | C# Corner MVP | Writer | C# | .Net Core | SQL | Javascript | Angular | Python | Django | Azure DevOps | DataStructure & Algorithm

http://www.mukeshkumar.net https://in.linkedin.com/in/mukeshkumartech

100 **12.2m** **4** **1**

View Previous Comments

17 **20**

Type your comment here and press Enter Key (Minimum 10 characters)

Tks. You save my life

Thành Nguyễn

2032 **14** **0**

Jan 31, 2021

0 0 Reply

Can you add new link to download please ? Thank you

Adam Kocourek

2009 **37** **0**

Jan 21, 2021

1 0 Reply

Its wonderful. So easy to learn.

Muhammad Inaam

2042 **4** **0**

Nov 26, 2020

0 0 Reply

Thanks, very helpful.

Yunus Olalekan

2037 **9** **0**

Oct 15, 2020

0 0 Reply

Hy mukesh,i am getting error value cannot be null,paramter:connectionstring.can you please explain why this error occuring

Rohan Goyal

2023 **23** **0**

Jul 11, 2020

0 0 Reply

Awesome Article

Vivek Kumar

190 **11.6k** **3.6m**

Feb 11, 2020

0 0 Reply

Nice and helpful. Thumbs up!

Jason Chiew

Feb 02

Very Help fullllll thnx

shakib patel

Jan 08, 2020

2000 **46** **0**

0 0 Reply

Thank you for your good example.

MohammadHossein KamaliNezhad

Jan 03, 2020

2026 **20** **0**

0 0 Reply

Thank you so much for explaining in a simple language. I just want to know how we can have the same table's primary key as a foreign key with seed method?

Mahavirsinh Padhiyar

Dec 17, 2019

1895 **151** **1.7k**

0 0 Reply

FEATURED ARTICLES

What Is NFT? Everything You Need To Know About NFTs

Unit Testing With xUnit And Moq In ASP.NET Core

Using Certificates For API Authentication In .NET 5

List Of Commonly Used GitHub Commands

View All

TRENDING UP

01 What's New In Java 16?

02 Azure Function - An Serverless Architecture

03 Data Science - A Skill Needed For Future

04 Introduction To Clean Architecture And Implementation With ASP.NET Core

05 Creating And Training Custom ML Model to Read Sales Receipts Using AI-Powered Azure Form Recognizer

06 Create And Validate JWT Token In .NET 5.0

07 RabbitMQ Designs

08 Power Apps Add in to Excel for Creating and Updating records in Dynamics 365 CRM by Srinath Pega

09 Distributed Caching (Redis)

10 A Diagnosis Of Parallel.Foreach

View All