Code First with EF Core

Overview

Introduction to Code First with EF Core

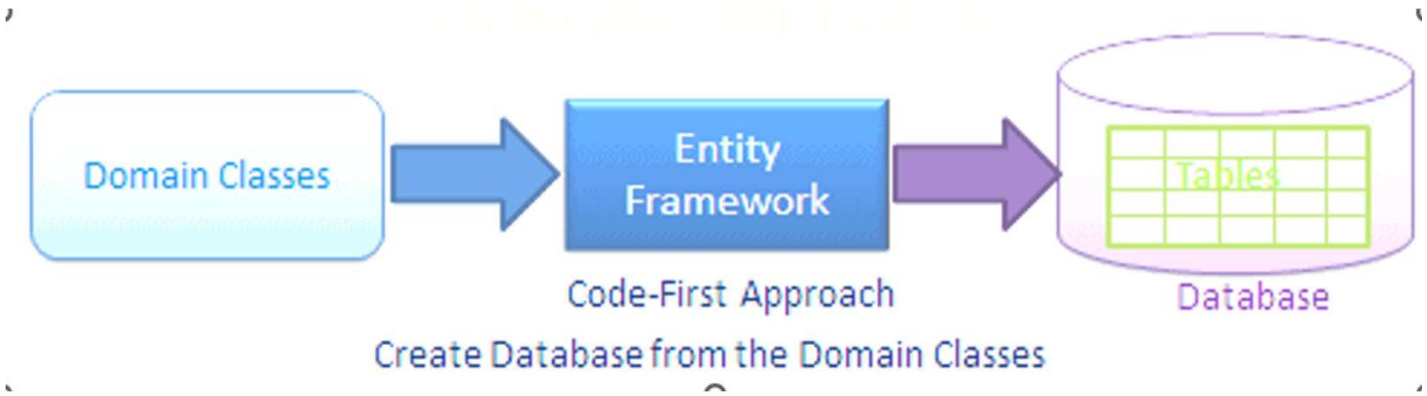
How to use Code First Approach?

Introduction to Code First

Creating a Database from an Existing Domain Classes in Entity Framework Core

- Code-first approach in Entity Framework Core (EF Core) allows you to design your application's domain model using C# classes, and then generate the corresponding database schema based on that model.
- It emphasizes writing code first and letting EF Core handle the database creation and management.
- So, we need to do reverse engineering using the Scaffold-DbContext command.
- The Code-first approach provides flexibility and control over the database schema from within your application code.
- It allows for rapid development and iteration as you can focus on designing the domain model and relationships in code without worrying about database details upfront.

Creating a Database using Existing Model in Entity Framework Core



How to use Code First Approach?

- **To Install EF Core DB Provider**

- There are different EF Core DB providers available for the different databases. These providers are available as NuGet packages.
- we need to install the NuGet package for the provider of the database we want to access.
- we want to access MS SQL Server database, so we need to install ***Microsoft.EntityFrameworkCore.SqlServer*** NuGet package.

- To Install Package using**.NET Core CLI**

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

- **To Install EF Core Tools**

- Along with the DB provider package, you also need to install EF tools to execute EF Core commands.
- These make it easier to perform several EF Core-related tasks in your project at design time, such as migrations, scaffolding, etc.
- In order to execute EF Core commands from Package Manager Console, search for the ***Microsoft.EntityFrameworkCore.Tools*** package from NuGet

- To install pakage using.**NET Core CLI**

```
dotnet add package Microsoft.EntityFrameworkCore.Tools
```

- **To Create a Project and Write Domain Classes.**

- Open a Visual Studio and Create a Console Application.
- Create Models folder and Add domain classes as given below.

```
using System.ComponentModel.DataAnnotations.Schema;
namespace DAL.Models
{
    [Table("DeptMaster")]
    public class DeptMaster
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Column("DeptCode")]
        [Range(minimum: 1, maximum: 10)]
        public int DeptCode { get; set; }

        [Required]
        [StringLength(maximumLength:50,MinimumLength =2)]
        [Column("DeptName")]
        public string DeptName { get; set; }

   }
}
```

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace DAL.Models
{
    [Table("EmpProfile")]
    //[Keyless]
    public class EmpProfile
    {
        [Key]
        [Column("EmpCode")]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Range(minimum: 1, maximum: 500)]
        public int EmpCode { get; set; }

        [Required]
        [StringLength(maximumLength:50,MinimumLength =2)]
        [Column("EmpName")]
        public string EmpName { get; set; }

```

```
        [Required]
        [Column("DateOfBirth")]
        [DataType(DataType.Date)]
        public DateTime DateOfBirth { get; set; }

        [Required]
        [Column("Email")]
        [DataType(DataType.EmailAddress)]
        public string Email { get; set; }

        [Required]
        [Column("DeptCode")]
        [RegularExpression(pattern: @"^[1-
9]\d*(\.\d+)?$")]
        public int DeptCode { get; set; }
```

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
namespace DAL.Models
{
    [Table("SalaryInfo")]
    public class SalaryInfo
    {
        [Key]
        [Column("SalarySheetNo")]
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int SalarySheetNo { get; set; }

        [Required]
        [Column("EmpCode")]
        [RegularExpression(pattern: @"^[1-9]\d*(\.\d+)?$")]
        [Range(minimum:1,maximum:500)]
        public int EmpCode { get; set; }

        [Required]
        [Column("Basic")]
        public decimal Basic { get; set; }

        [Column("Hra")]
        public decimal? Hra { get; set; }

        [Column("Da")]
        public decimal? Da { get; set; }
    }
}
```

- **To Add a Connection String.**

- Add appSettings.json file to the solution and write a connection strings section to add connection string.

```
{
  "ConnectionStrings": {
    "EmpDbCon": "server=(localdb)\\MSSQLLocalDB;Initial catalog=EmpData;integrated security=true;"
  }
}
```

- After executing migration command, Database will be created with the name (EmpData).

• To Add DbContext Class

- Add a DbContext Class inside a Models folder and write below code.

```
using Microsoft.EntityFrameworkCore;
namespace DAL.Models
{
   public class EmpContext:DbContext
   {
      public DbSet<DeptMaster> DeptMasters { get; set; }

      public DbSet<EmpProfile> EmpProfiles { get; set; }

      public DbSet<SalaryInfo> SalaryInfos { get; set; }


protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
      {
         if (!optionsBuilder.IsConfigured)
         {
            optionsBuilder.UseSqlServer(DatabaseHelper.GetConnectionString());
         }
      }
protected  override void OnModelCreating(ModelBuilder modelBuilder)
      {
         //This function contains Fluent API code to configure domain classes.

         //one-To-Many
         modelBuilder.Entity<EmpProfile>().HasOne<DeptMaster>().WithMany().HasForeignKey(d =>
d.DeptCode);

         //one-To-Many

modelBuilder.Entity<SalaryInfo>().HasOne<EmpProfile>().WithMany().HasForeignKey(emp=>emp.EmpCode
);

         //To apply Unique Key Constraint
         modelBuilder.Entity<EmpProfile>(entity => { entity.HasIndex(e => e.Email).IsUnique(); });

         //Adding Seed Data
         //modelBuilder.Entity<DeptMaster>().HasData(
         //    new DeptMaster { DeptCode=1,DeptName="Hr"},
         //    new DeptMaster { DeptCode=2,DeptName="Sales"},
         //    new DeptMaster { DeptCode=3,DeptName="Accounts"}
         //    );

}}
```

- **Migration Commands**

In Entity Framework Core (EF Core), you can use migration commands to manage the database schema changes when following the Code-first approach. Here are the commonly used migration commands in EF Core:

Add Migration: This command generates a new migration file based on the changes detected in your entity classes. It captures the differences between the current state of the entity classes and the previous migration.

To add a migration, open the Package Manager Console in Visual Studio and run the following command:

```
Add-Migration <MigrationName>
```

Replace <MigrationName> with a meaningful name that describes the purpose of the migration.

Update Database: This command applies the pending migrations to the database, updating the schema accordingly.

To update the database, run the following command in the Package Manager Console:

```
Update-Database
```

This command applies all pending migrations to the database specified in the DbContext connection string.

You can also specify a target migration to update the database up to a specific migration. For example:

```
Update-Database -Migration <TargetMigration>
```

Replace <TargetMigration> with the name of the target migration you want to update the database up to.

Remove Migration: If you need to revert a previously applied migration, you can use the remove migration command. This command removes the latest migration files and updates the database schema accordingly.

To remove the latest migration run the command in the Package Manager Console:

```
Remove-Migration
```

This command removes the latest migration and updates the database schema to the previous migration state.

These are some of the essential migration commands in EF Core. Additional commands, such as **Script-Migration**, **Get-Migration**, and **Drop-Database**, provide more advanced capabilities for working with migrations.

- **To Execute Add-Migration Command**

- In Visual Studio, select menu Tools -> NuGet Package Manger -> Package Manger Console and run the following command:

```
PM> Add-Migration CreateDB
```

- *This will create migration class with the schema defined in the DeptMaster, EmpProfile and SalaryInfo domain classes .*

- **To Execute Update-Database Command**

- In Visual Studio, select menu Tools -> NuGet Package Manger -> Package Manger Console and run the following command:

```
PM> Update-database
```

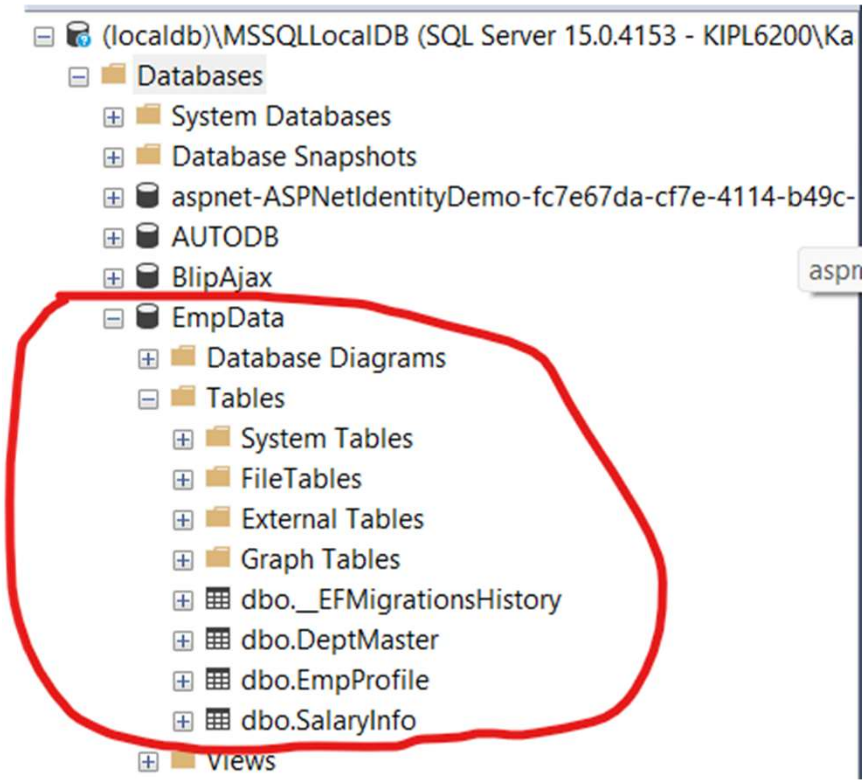
- *This will execute a script script created in migration class (CreateDB) and apply changes to create target database (EmpDB) mentioned in the connection string.*

appSettings.json

```
{
  "ConnectionStrings": {
    "EmpDbCon": "server=(localdb)\\MSSQLLocalDB;Initial catalog=EmpData;integrated security=true;"
  }
}
```

- **Notice the Changes made in form of the Database**

- Open a SQL Server Management Studio and see changes.



```
⊟ (localdb)\MSSQLLocalDB (SQL Server 15.0.4153 - KIPL6200\Ka
  ⊟ Databases
    ⊞ System Databases
    ⊞ Database Snapshots
    ⊞ aspnet-ASPNetIdentityDemo-fc7e67da-cf7e-4114-b49c-
    ⊞ AUTODB
    ⊞ BlipAjax
    ⊟ EmpData
      ⊞ Database Diagrams
      ⊟ Tables
        ⊞ System Tables
        ⊞ FileTables
        ⊞ External Tables
        ⊞ Graph Tables
        ⊞ dbo.__EFMigrationsHistory
        ⊞ dbo.DeptMaster
        ⊞ dbo.EmpProfile
        ⊞ dbo.SalaryInfo
      ⊞ Views
```

References

https://www.entityframeworktutorial.net/efcore/create-model-for-existing-database-in-ef-core.aspx

https://www.c-sharpcorner.com/article/entity-framework-database-first-in-asp-net-core2/

https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

https://www.c-sharpcorner.com/article/entity-framework-database-first-in-asp-net-core2/

https://www.c-sharpcorner.com/article/entity-framework-core-6-with-database-first/

https://docs.devart.com/dotconnect/postgresql/EFCore-Database-First-NET-Core.html

https://dotnettutorials.net/lesson/entity-framework-database-first-approach/