Entity Framework

Notes
rajeeshvengalapurath@gmail.com
17 Jul 2019

Reference

https://www.youtube.com/watch?v=Z7713GBhi4k

Introduction

Entity Framework is an ORM framework

Object Relational Mapping Framework

Object Relational Mapping framework automatically creates classes based on database tables, and vice versa is also true, that is, it can also automatically generate necessary SQL to create database tables based on classes.

Approaches

1. Schema First Approach

Done based on existing database tables

2. Model First Approach

In this, we first create the Entity Model. That is we create
a. Entities
b. Relationship between entities
c. Inheritance hierarchies etc.
All this is done directly on the design surface of the EDMX file
Foreign key is created using new>association
Foreign key name (DesignationID) is automatically set
This approach is rare as per Mosh
(https://www.youtube.com/watch?v=ZX7_12fwQLU&t=1212s)

3. Code First Approach

Advantages

- Full versioning of database
- Done more quickly
- https://youtu.be/ZX7_12fwQLU (reference)

Make classes in the following way

```
public class ProductType
{
	public int ID { get; set; }
```

```
        public string Name { get; set; }
        public List<Product> Products { get; set; }
}

public class Product
{
        public int ID { get; set; }
        public string Name { get; set; }
        public int ProductTypeID { get; set; }  //Include1
        [ForeignKey("ProductTypeID")]           //Include2
        public ProductType ProductType { get; set; }
}
```

Include1 & Include2 should be specified to get a foregin key name ProductTypeID. Remove both lines and the system will automatically generate the foregin key name as ProductType_ID

```
using System.Data.Entity; //For DbContext and DbSet<>
public class ProductDBContext : DbContext
{
        public DbSet<ProductType> ProductTypes { get; set; }
        public DbSet<Product> Products { get; set; }
}
```

Name of the connection string in config file must be “ProductDBContext”

```
public List<ProductType> GetProductTypes()
{
        ProductDBContext productDBContext = new ProductDBContext();
        return productDBContext.ProductTypes.ToList();
}
```

Sample Insert, Update and Delete

```
Using System.ComponentModel.DataAnnotations
public class ProductType
{
        public int ID { get; set; }
        [StringLength(50)]
        public string Name { get; set; }
}
using System.Data.Entity;
public class ProductDBContext : DbContext
{
        public DbSet<ProductType> ProductTypes { get; set; }
}
public class ProductRepository
{
        public List<ProductType> GetProductTypes()
        {
                ProductDBContext productDBContext = new ProductDBContext();
                return productDBContext.ProductTypes.ToList();
```

```
        }
        public void InserProductType(ProductType productType)
        {
               ProductDBContext productDBContext = new ProductDBContext();
               productDBContext.ProductTypes.Add(productType);
               productDBContext.SaveChanges();
        }
        public void UpdateProductType(ProductType productType)
        {
               ProductDBContext productDBContext = new ProductDBContext();
               ProductType productTypeToUpdate =
               productDBContext.ProductTypes.FirstOrDefault(x=>x.ID==productType.ID);
               productTypeToUpdate.Name = productType.Name;
               productDBContext.SaveChanges();
        }
        public void DeleteProductType(ProductType productType)
        {
               ProductDBContext productDBContext = new ProductDBContext();
               ProductType productTypeToDelete =
                      productDBContext.ProductTypes.FirstOrDefault(x => x.ID ==
                      productType.ID);
               productDBContext.ProductTypes.Remove(productTypeToDelete);
               productDBContext.SaveChanges();
        }
}
```

Delete

```
using (SampleDBContext db = new SampleDBContext())
{
       Product product = new Product { Id = 4 };
       db.Products.Attach(product);
       db.Products.Remove(product);
       db.SaveChanges();
}
```

Code First Migrations

1. Run the Enable-Migrations command in Package Manager Console
   Result:
   PM> Enable-Migrations
   Checking if the context targets an existing database…
   Detected database created with a database initializer. Scaffolded migration '201907200644298_InitialCreate' corresponding to existing database. To use an automatic migration instead, delete the Migrations folder and re-run Enable-Migrations specifying the -EnableAutomaticMigrations parameter.
   Code First Migrations enabled for project EFTestWinforms.
   This command has added a Migrations folder to our project. This new folder contains two files:
2. The Configuration class. This class allows you to configure how Migrations behaves for your context. For this walkthrough we will just use the default configuration.

Because there is just a single Code First context in your project, Enable-Migrations has automatically filled in the context type this configuration applies to.

3. Generating & Running Migrations
   Code First Migrations has two primary commands that you are going to become familiar with.
   - Add-Migration will scaffold the next migration based on changes you have made to your model since the last migration was created
   - Update-Database will apply any pending migrations to the database

General guidelines when deciding on the lifetime of the context

- When working with Web applications, use a context instance per request.
- When working with Windows Presentation Foundation (WPF) or Windows Forms, use a context instance per form. This lets you use change-tracking functionality that context provides.
- If the context instance is created by a dependency injection container, it is usually the responsibility of the container to dispose the context.
- If the context is created in application code, remember to dispose of the context when it is no longer required.
- When working with long-running context consider the following:
  - As you load more objects and their references into memory, the memory consumption of the context may increase rapidly. This may cause performance issues.
  - The context is not thread-safe, therefore it should not be shared across multiple threads doing work on it concurrently.
  - If an exception causes the context to be in an unrecoverable state, the whole application may terminate.
  - The chances of running into concurrency-related issues increase as the gap between the time when the data is queried and updated grows.

Multiple DBContext

https://stackoverflow.com/questions/21843751/splitting-multiple-dbcontexts

Actually you don't need have to create one DbContext for each Entity. But you can have multiple DbContext for a single database. For an example let's say you want to separate your business aspect and security aspect of the application into two different modules. Then of course you can have two different context such as SecurityContext which has all the entities related to Security and BusinessContext consists of Business related entities. Ex:

```
public class SecurityContext : DbContext
{
       public PersonsContext() : base("name=securitycontext"){ }
       public DbSet<User> User { get; set; }
```

```
        public DbSet<Role> Role { get; set; }
        public DbSet<Group> Group { get; set; }
}

public class BusinessContext : DbContext
{
        public OrderContext() : base("name=businesscontext"){ }
        public DbSet<Order> Order { get; set; }
        public DbSet<OrderLine> OrderLine { get; set; }
        public DbSet<Customer> Customer { get; set; }
}
```

EoF