Entity Framework

Fluent API Configurations

Lesson Objectives

- Fluent API Configurations
- Entity Mappings
- Property Mappings
- Relationship in Entity Framework

Section 1

FLUENT API CONFIGURATIONS

Fluent API Configurations

- Entity Framework Fluent API is used to configure domain classes to override conventions.
  - ✓ In other words, we can use both Data Annotation attributes and Fluent API at the same time
  - ✓ Fluent API override Data Annotations attributes.

Fluent API Configurations

- EF Fluent API is based on a Fluent API design pattern (a.k.a Fluent Interface) where the result is formulated by method chaining.
- The DbModelBuilder class acts as a Fluent API.
- It provides more options of configurations than Data Annotation attributes.

Fluent API Configurations

- To write Fluent API configurations, override the OnModelCreating() method of DbContext in a context class
  - ✓ protected override void OnModelCreating(DbModelBuilder modelBuilder)

Fluent API Configurations

```
public class SchoolContext: DbContext
{

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

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        //Write Fluent API configurations here

    }
}
```

Fluent API Configurations

- Model-wide Configuration: Configures the default Schema, entities to be excluded in mapping, etc.
- Entity Configuration: Configures entity to table and relationship mappings
  - ✓ e.g. PrimaryKey, Index, table name, one-to-one, one-to-many, many-to-many etc.
- Property Configuration: Configures property to column mappings
  - ✓ e.g. column name, nullability, Foreignkey, data type, concurrency column, etc.

Entity Mappings

- Configure Default Schema
  - ✓ Specifies the default database schema.
  - ✓ modelBuilder.HasDefaultSchema("Admin");

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    //Configure default schema
    modelBuilder.HasDefaultSchema("Admin");
}
```

Entity Mappings

- Map Entity to Table
  - ✓ Specifies the table name instead of default convention.
  - ✓ modelBuilder.Entity<Product>().ToTable(“NewProduct");
  - ✓ modelBuilder.Entity<Product>().ToTable("NewProduct", "Admin");

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    //Configure default schema
    modelBuilder.HasDefaultSchema("Admin");

    //Map entity to table
    modelBuilder.Entity<Student>().ToTable("StudentInfo");
    modelBuilder.Entity<Standard>().ToTable("StandardInfo","dbo");
}
```

Map Entity to Multiple Tables

- Map set of properties to each table
- Used when:
  - ✓ Has many properties/columns
  - ✓ Separate table for difference used purposes

```
modelBuilder.Entity<Product>().Map(m =>
{
    m.Properties(p => new { p.ID, p.Name });
    m.ToTable("ProductBasic");
}).Map(m => {
    m.Properties(p => new { p.Weight, p.Height,
        p.Long, p.Width });
    m.ToTable("ProductDetails");
});
```

Property Mappings

- Configure Primary Key and Composite Primary Key
  - ✓ To configure a key property.
  - ✓ Override default convention, ID or <Entity>Id becomes normal column in SQL
  - ✓ modelBuilder.Entity<Product>().HasKey(p => p.ProductKey);

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    //Configure primary key
    modelBuilder.Entity<Student>().HasKey<int>(s => s.StudentKey);
    modelBuilder.Entity<Standard>().HasKey<int>(s => s.StandardKey);

    //Configure composite primary key
    modelBuilder.Entity<Student>().HasKey<int>(s => new { s.StudentKey, s.StudentName });
}
```

Property Mappings

- Configure Column Name, Type and Order
  - ✓ To configure column name, order, data type in SQL

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    //Configure Column
    modelBuilder.Entity<Student>()
                .Property(p => p.DateOfBirth)
                .HasColumnName("DoB")
                .HasColumnOrder(3)
                .HasColumnType("datetime2");
}
```

Property Mappings

- Configure Null or NotNull Column
  - ✓ To configure Null or NotNull Column

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
        //Configure Null Column
    modelBuilder.Entity<Student>()
            .Property(p => p.Heigth)
            .IsOptional();

        //Configure NotNull Column
        modelBuilder.Entity<Student>()
            .Property(p => p.Weight)
            .IsRequired();
}
```

Property Mappings

- Configure Column Size
  - ✓ HasMaxLength: to configure maxlength
  - ✓ IsFixedLength: to change datatype from nvarchar to nchar
  - ✓ HasPrecision: to set size decimal

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    //Set StudentName column size to 50
    modelBuilder.Entity<Student>()
            .Property(p => p.StudentName)
            .HasMaxLength(50);

    //Set StudentName column size to 50 and change datatype to nchar
    //IsFixedLength() change datatype from nvarchar to nchar
    modelBuilder.Entity<Student>()
            .Property(p => p.StudentName)
            .HasMaxLength(50).IsFixedLength();

    //Set size decimal(2,2)
        modelBuilder.Entity<Student>()
            .Property(p => p.Height)
            .HasPrecision(2, 2);
}
```

Property Mappings

- Configure Concurrency Column
  - ✓ IsConcurrencyToken: to set column as concurrency column so that it will be included in the where clause in update and delete commands.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    //Set StudentName as concurrency column
    modelBuilder.Entity<Student>()
            .Property(p => p.StudentName)
            .IsConcurrencyToken();
}
```

Section 2

CONFIGURE RELATIONSHIPS

Configure One-to-Many Relationships

- Step 1/4: start configuring with any one entity class.
- Step 2/4: use **.HasRequired()** to specifies required property. This will create a NotNull foreign key column in the DB.

Configure One-to-Many Relationships

- Step 3/4: use **.WithMany()** to specifies that the parent entity class includes many children entities. Here, many infers the ICollection type property.
- Step 4/4: use .**HasForeignKey()** to specify the name of the foreign key

Configure One-to-One Relationship

- Configure a One-to-Zero-or-One relationship
  - ✓ Get Entity from any side
  - ✓ Use HasOptional() to another entity
  - ✓ Use WithRequired() this entity
- Configure a One-to-One relationship
  - ✓ Get Entity from any side
  - ✓ Use HasRequired() to another entity
  - ✓ Use WithRequiredPrincipal() this entity

Configure One-to-One Relationship

- Configure a One-to-Zero-or-One relationship
  - ✓ Get Entity from any side
  - ✓ Use HasOptional() to another entity
  - ✓ Use WithRequired() this entity

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    // Configure StudentId as FK for StudentAddress
    modelBuilder.Entity<Student>()
                .HasRequired(s => s.Address)
                .WithRequiredPrincipal(ad => ad.Student);

}
```

Configure a One-to-Many Relationship

- *Generally, the one-to-many relationship in entity framework because one-to-many relationship conventions cover all combinations.*

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    // configures one-to-many relationship
    modelBuilder.Entity<Student>()
        .HasRequired<Grade>(s => s.CurrentGrade)
        .WithMany(g => g.Students)
        .HasForeignKey<int>(s => s.CurrentGradeId);
}
```

Configure a One-to-Many Relationship

- Configure the NotNull ForeignKey
  - ✓ Use: HasRequired method
- Configure Cascade Delete
  - ✓ Use: WillCascadeOnDelete

Configure a Many-to-Many Relationship

- Get Entity from any side
- Use HasMany() to another entity
- Use WithMany() to this entity
- [Optional] Use Map
  - ✓ Use MapLeftKey
  - ✓ Use MapRightKey
  - ✓ Use ToTable

Configure a Many-to-Many Relationship

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{

    modelBuilder.Entity<Student>()
                .HasMany<Course>(s => s.Courses)
                .WithMany(c => c.Students)
                .Map(cs =>
                        {
                            cs.MapLeftKey("StudentRefId");
                            cs.MapRightKey("CourseRefId");
                            cs.ToTable("StudentCourse");
                        });

}
```

Thank you