Entity Relations

Customizing Entity Models

SoftUni Team

Technical Trainers

Software University

SoftUni Foundation

Software University

http://softuni.bg

Table of Contents

Have a Question?

SoftUni Foundation

sli.do

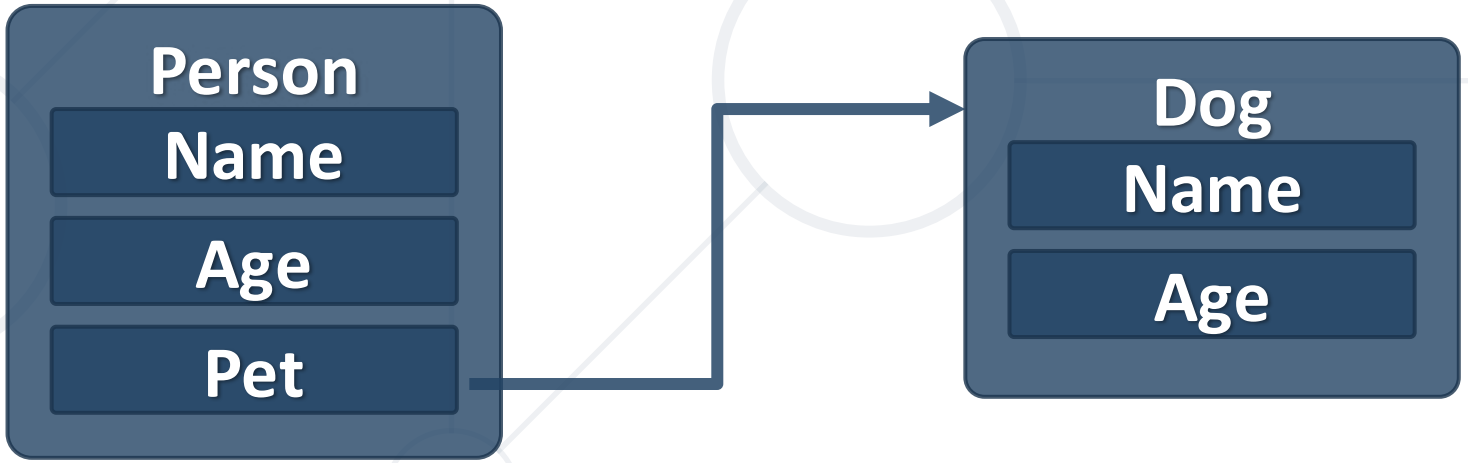
#csharp-db

Object Composition

Describing Database Relationships

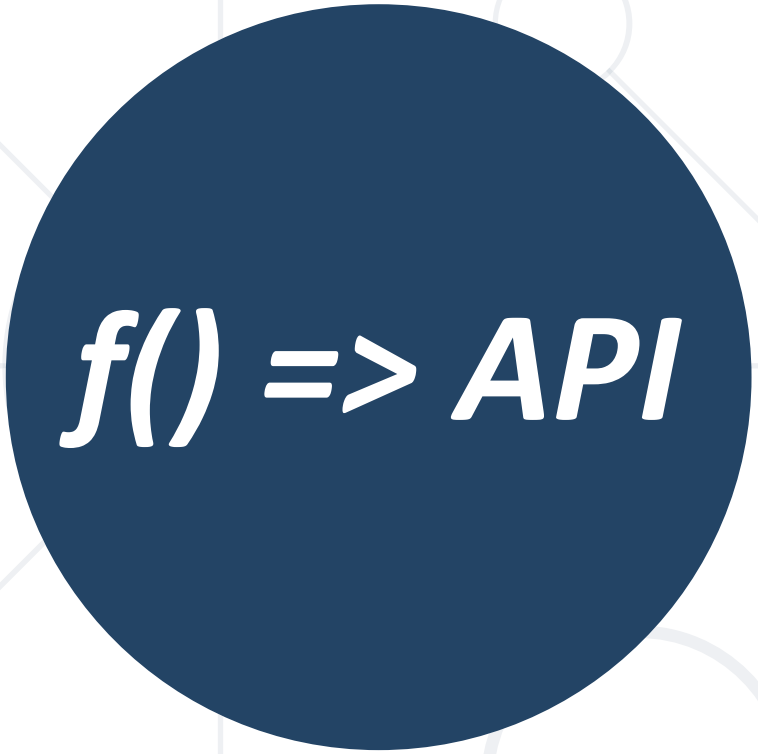
Object Composition

- Object composition denotes a "**has-a**" relationship
  - E.g. the **car** has an **engine**
- Defined in C# by one object having a property that is a reference to another



Navigation Properties

- Navigation properties create a **relationship** between entities
- Is either an **Entity Reference** (one to one or zero) or an **ICollection** (one to many or many to many)
- They provide **fast querying** of related records
- Can be **modified** by **directly** setting the reference



Fluent API

Working with Model Builder

Fluent API

SoftUni Foundation

- **Code First** maps your POCO classes to tables using a **set of conventions**
  - E.g. property named "**Id**" maps to the **Primary Key**
- Can be customized using **annotations** and the **Fluent API**
- Fluent API (Model Builder) allows **full control** over DB mappings
  - Custom names of objects (columns, tables, etc.) in the DB
  - Validation and data types
  - Define complicated entity relationships

Working with Fluent API

SoftUni Foundation

- Custom mappings are placed inside the **OnModelCreating** method of the DB context class

```
protected override void OnModelCreating(DbModelBuilder builder)
{
  builder.Entity<Student>().HasKey(s => s.StudentKey);
}
```

Fluent API: Renaming DB Objects

- Specifying Custom Table name

```
modelBuilder.Entity<Order>()
        .ToTable("OrderRef", "Admin");
```

- Custom Column name/DB Type

```
modelBuilder.Entity<Student>()
   .Property(s => s.Name)
   .HasColumnName("StudentName")
   .HasColumnType("varchar");
```

Optional schema name

Fluent API: Column Attributes

SoftUni Foundation

- Explicitly set Primary Key

```
modelBuilder
      .Entity<Student>().HasKey("StudentKey");
```

- Other column attributes

```
modelBuilder.Entity<Person>()
  .Property(p => p.FirstName)
  .IsRequired()
  .HasMaxLength(50)
```

```
modelBuilder.Entity<Post>()
  .Property(p => p.LastUpdated)
  .ValueGeneratedOnAddOrUpdate()
```

Fluent API: Miscellaneous Config

SoftUni Foundation

- Do not include property in DB (e.g. business logic properties)

```
modelBuilder
      .Entity<Department>().Ignore(d => d.Budget);
```

- Disabling cascade delete
  - If a FK property is non-nullable, cascade delete is **on by default**

```
modelBuilder.Entity<Course>()
  .HasRequired(t => t.Department)
  .WithMany(t => t.Courses)
  .HasForeignKey(d => d.DepartmentID)
  .OnDelete(DeleteBehavior.Restrict);
```

Throws exception on delete

Specialized Configuration Classes

- Mappings can be placed in entity-specific classes

```
public class StudentConfiguration
   : IEntityTypeConfiguration<Student>
{
   public void Configure(EntityTypeBuilder<Student> builder)
   {
      builder.HasKey(c => c.StudentKey);
   }
}
```

Specify target model

- Include in **OnModelCreating**:

```
builder.ApplyConfiguration(new StudentConfiguration());
```

Table Relationships

Expressed as Properties and Attributes

One-to-Zero-or-One

- Expressed in SQL Server as a shared primary key
- Relationship direction must be explicitly specified with a **ForeignKey** attribute
- ForeignKey is placed above the key property and contains the name of the navigation property and vice versa

One-to-Zero-or-One: Implementation

- Using the **ForeignKey** Attribute

```
public class Student
{
    [Key]
    public int Id { get; set; }
    public string Name { get; set; }

    public int AddressId { get; set; }
    [ForeignKey("Address")]
    public Address Address { get; set; }
}
```

Attributes

One-to-Zero-or-One: Implementation (2)

SoftUni Foundation

- Using the **ForeignKey** Attribute

```
public class Address
{
  public int Id { get; set; }
  public string Text { get; set; }

  public int StudentId { get; set; }
  [ForeignKey(nameof(Student))]
  public Student Student { get; set; }
}
```

One-to-Zero-or-One: Fluent API

- HasOne → WithOne

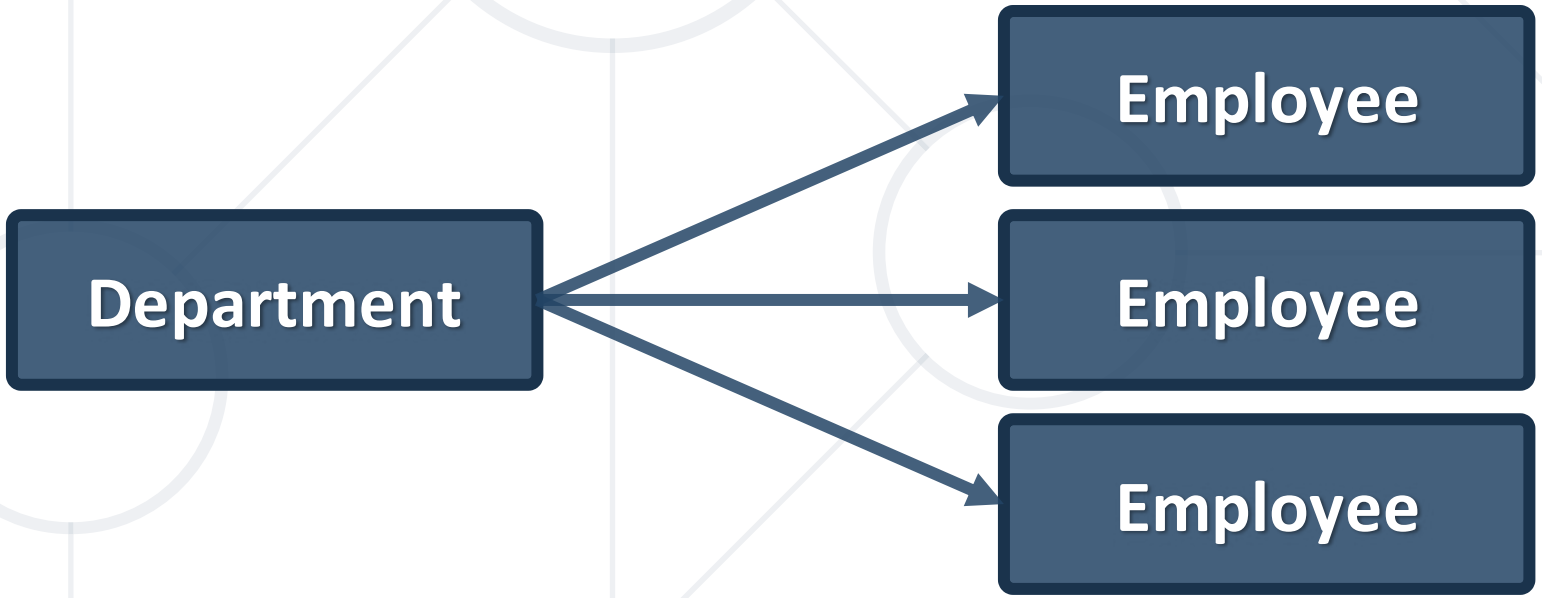
```
modelBuilder.Entity<Address>()
   .HasOne(a => a.Student)
   .WithOne(s => s.Address)
   .HasForeignKey(a => a.StudentId);
```

Address contains FK to Student

- If StudentId property is nullable (int?), relation becomes One-To-Zero-Or-One

One-to-Many

- Most common type of relationship
- Implemented with a **collection** inside the **parent entity**
  - The collection should be **initialized** in the **constructor**!



One-to-Many: Implementation

SoftUni Foundation

- Department has many employees

```
public class Department
{
  public int Id { get; set; }
  public string Name { get; set; }

  public ICollection<Employee> Employees { get; set; }
}
```

One-to-Many: Implementation (2)

SoftUni Foundation

- **Employees** have one **department**

```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}
```

One-to-Many: Fluent API

SoftUni Foundation

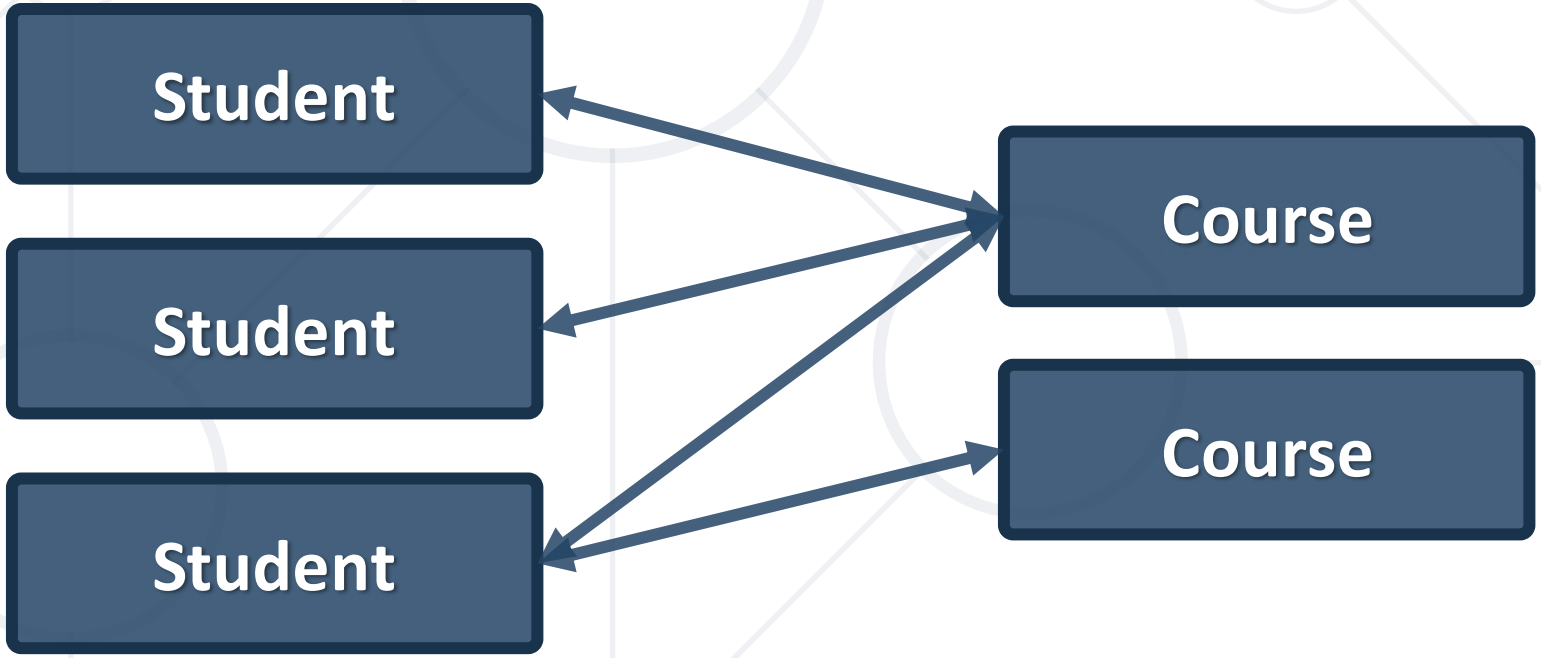
- HasMany → WithOne

```
modelBuilder.Entity<Post>()
   .HasMany(p => p.Comments)
   .WithOne(c => c.Post)
   .HasForeignKey(c => c.PostId);
```

```
modelBuilder.Entity<Employee>()
   .HasMany(e => e.Addresses)
   .WithOne(a => a.Employee)
   .HasForeignKey(a => a.EmployeeId);
```

Many-to-Many

- Requires a **join entity (separate class)** in EF Core
- Implemented with collections in each entity, referring the other



Many-to-Many Implementation (1)

SoftUni Foundation

```
public class Course
{
   public string Name { get; set; }

   public ICollection<StudentCourse> StudentsCourses { get; set; }
}
```

```
public class Student
{
   public string FirstName { get; set; }
   public string LastName { get; set; }

   public ICollection<StudentCourse> StudentsCourses { get; set; }
}
```

Many-to-Many Implementation (2)

- EF Core requires a **Join Entity**

```
public class StudentCourse
{
    public int StudentId { get; set; }
    public Student Student { get; set; }

    public int CourseId { get; set; }
    public Course Course { get; set; }
}
```

Many-to-Many: Fluent API

SoftUni Foundation

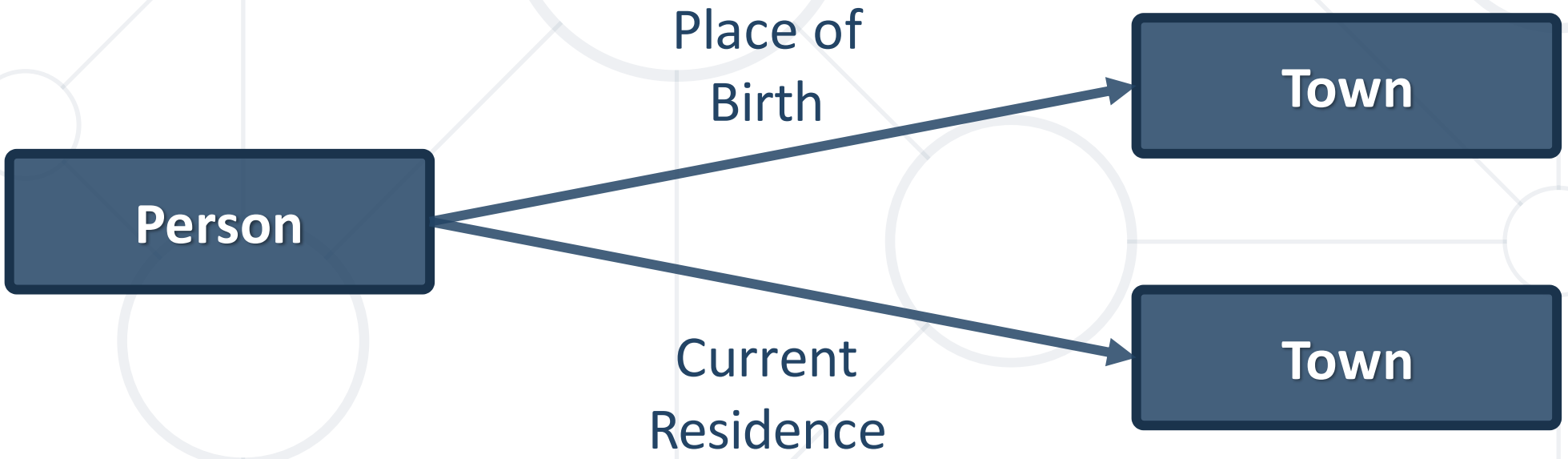
- Mapping **both sides** of relationship

```
modelBuilder.Entity<StudentCourse>()
   .HasKey(sc => new { sc.StudentId, sc.CourseId });

builder.Entity<StudentCourse>()
   .HasOne(sc => sc.Student)
   .WithMany(s => s.StudentCourses)
   .HasForeignKey(sc => sc.StudentId);

builder.Entity<StudentCourse>()
   .HasOne(sc => sc.Course)
   .WithMany(s => s.StudentCourses)
   .HasForeignKey(sc => sc.CourseId);
```

Composite Primary Key

Multiple Relations

- When two entities are related by more than one key
- Entity Framework needs help from **Inverse Properties**



Multiple Relations Implementation

SoftUni Foundation

- **Person** Domain Model – defined as usual

```
public class Person
{
  public int Id { get; set; }
  public string Name { get; set; }

  public Town PlaceOfBirth { get; set; }
  public Town CurrentResidence { get; set; }
}
```

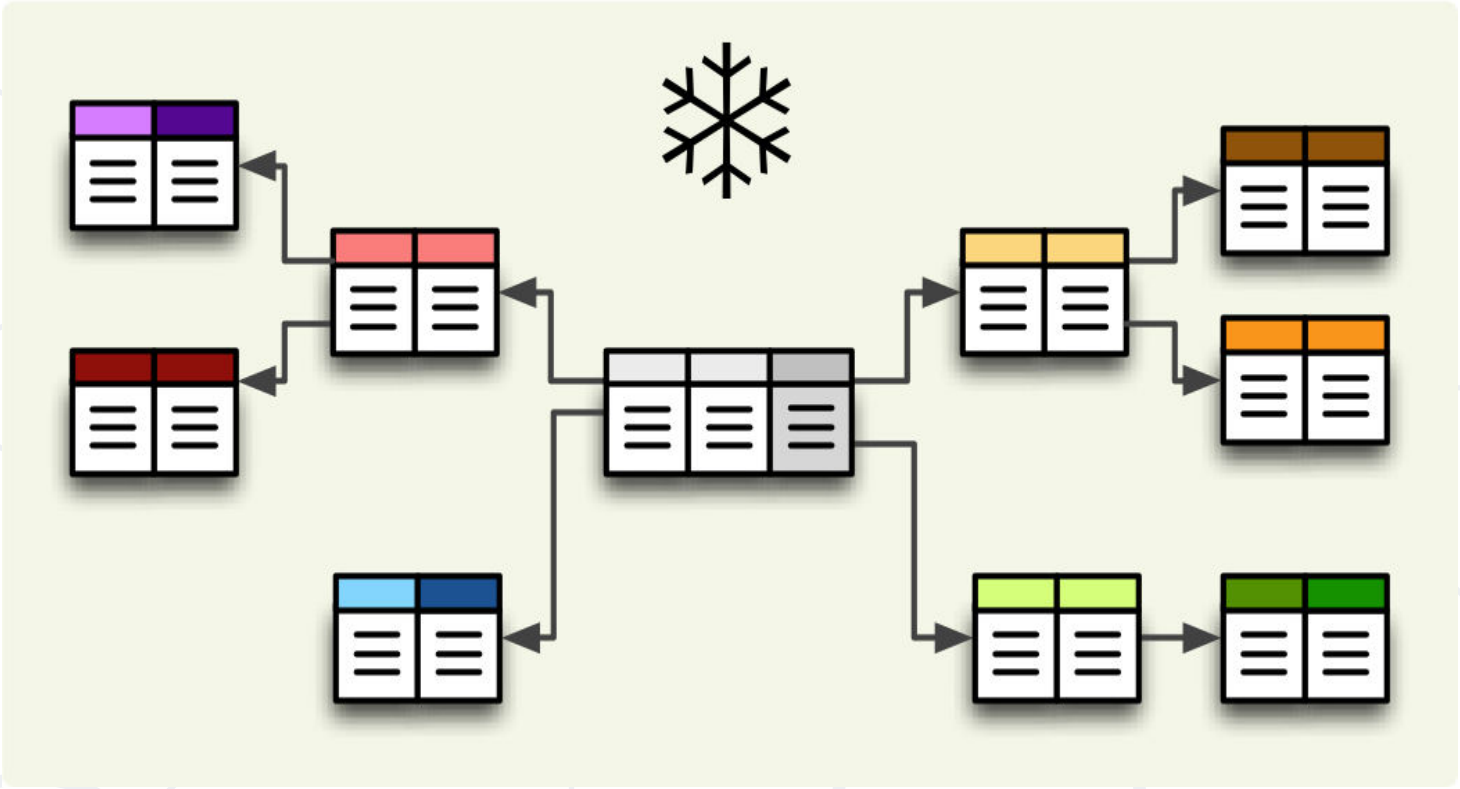
Multiple Relations Implementation (2)

SoftUni Foundation

- **Town** Domain Model

```
public class Town
{
  public int Id { get; set; }
  public string Name { get; set; }
  [InverseProperty("PlaceOfBirth")]
  public ICollection<Person> Natives { get; set; }
  [InverseProperty("CurrentResidence")]
  public ICollection<Person> Residents { get; set; }
}
```

Point towards related property



Filtering and Aggregating Tables

Select, Join and Group Data Using LINQ

Good Reasons to Use Select

SoftUni Foundation

- Limit network traffic by reducing the queried columns
- Syntax:

```
var employeesWithTown = context
   .Employees
   .Select(employee => new
   {
      EmployeeName = employee.FirstName,
      TownName = employee.Address.Town.Name
   });
```

- SQL Server Profiler

```
   SELECT [employee].[FirstName] AS [EmployeeName], [employee.Address.Town].[Name] AS [TownName]
     FROM [Employees] AS [employee]
LEFT JOIN [Addresses] AS [employee.Address] ON [employee].[AddressID] = [employee.Address].[AddressID]
LEFT JOIN [Towns] AS [employee.Address.Town] ON [employee.Address].[TownID] =
          [employee.Address.Town].[TownID]
```

Good Reasons not to Use Select

SoftUni Foundation

- Data that is selected is **not** of the **initial entity type**
  - **Anonymous type**, generated at runtime

```
(local variable) System.Collections.Generic.List<'a> employeesWithTown

Anonymous Types:
    'a is new { string EmployeeName, string TownName }

Local variable 'employeesWithTown' is never used
```

- **Data cannot be modified** (updated, deleted)
  - Entity is of a **different type**
  - Not associated with the **context** anymore

Joining Tables in EF: Using Join()

- Join tables in EF with **LINQ** / **extension methods** on **IEnumerable<T>** (like when joining collections)

```
var employees =
  softUniEntities.Employees.Join(
    softUniEntities.Departments,
    (e => e.DepartmentID),
    (d => d.DepartmentID),
    (e, d) => new {
      Employee = e.FirstName,
      JobTitle = e.JobTitle,
      Department = d.Name
    }
  );
```

Grouping Tables in EF

SoftUni Foundation

- Grouping also can be done by LINQ
  - The same way as with collections in LINQ
- Grouping with LINQ:

```
var groupedEmployees =
  from employee in softUniEntities.Employees
  group employee by employee.JobTitle;
```

- Grouping with extension methods:

```
var groupedCustomers = softUniEntities.Employees
  .GroupBy(employee => employee.JobTitle);
```

Result Models

Simplifying Models

Result Models

- **Select()**, **GroupBy()** can work with **custom classes**
  - Allows you to **pass them** to methods and use them as a return type
  - Requires some **extra code** (class definition)
- Sample Result Model:

```
public class UserResultModel
{
   public string FullName { get; set; }
   public string Age { get; set; }
}
```

Result Models (2)

- Assign the fields as you would with an anonymous object:

```
var currentUser = context.Users
  .Where(u => u.Id == 8)
  .Select(u => new UserResultModel
  {
    FullName = u.FirstName + " " + u.LastName,
    Age = u.Age
  })
  .SingleOrDefault();
```

- The new type can be used in a method signature:

```
public UserResultModel GetUserInfo(int Id) { … }
```

Attributes

Custom Entity Framework Behavior

Attributes

SoftUni Foundation

- EF Code First provides a set of **DataAnnotation attributes**
  - You can override default Entity Framework behavior
- To access nullability and size of fields:

```
using System.ComponentModel.DataAnnotations;
```

- To access schema customizations:

```
using System.ComponentModel.DataAnnotations.Schema;
```

- For a full set of configuration options you need the **Fluent API**

Key Attributes

- [Key] – explicitly specify **primary key**
  - When your PK column doesn't have an "Id" suffix

```
[Key]
public int StudentKey { get; set; }
```

  - **Composite key** is only defined using **Fluent API** for now

```
builder.Entity<EmployeesProjects>()
       .HasKey(k => new { k.EmployeeId, k.ProjectId });
```

Key Attributes (2)

- **ForeignKey** – explicitly **link** navigation property and foreign key property within the same class
- Works in **either direction** (FK to navigation property or navigation property to FK)

```
public class Client
{
  …
  [ForeignKey("Order")]
  public int OrderRefId { get; set; }
  public Order Order { get; set; }
}
```

Renaming Objects

- **Table** – manually specify the name of the table in the DB

```
[Table("StudentMaster")]
public class Student
{
  …
}
```

```
[Table("StudentMaster", Schema = "Admin")]
public class Student
{
  …
}
```

Renaming Objects (2)

- **Column** – manually specify the name of the column in the DB
  - You can also specify order and explicit data type

```
public class Student
{
  …
  [Column("StudentName", Order = 2, TypeName="varchar(50)")]
  public string Name { get; set; }
}
```

Optional parameters

Entity Validation

SoftUni Foundation

- **Required** – mark a nullable property as **NOT NULL** in the DB
  - Will throw an exception if not set to a value
  - Non-nullable types (e.g. **int**) will **not throw** an exception (will be set to language-specific default value)
- **MinLength** – specifies min length of a string (client validation)
- **MaxLength** / **StringLength** – specifies max length of a string (both client and DB validation)
- **Range** – set lower and/or upper limits of numeric property (client validation)

Other Attributes

- **Index** – create index for column
  - Primary key will always have an index

```
builder.Entity<Car>()
        .HasIndex(u => u.RegistrationNumber)
        .IsUnique();
```

- **NotMapped** – property will not be mapped to a column
  - For business logic properties

Validation Method

- Add using:

```
using System.ComponentModel.DataAnnotations;
```

- Create following method to validate entities:

```
private bool IsValid(object obj)
{
        var validationContext = new ValidationContext(obj);
        var validationResults = new List<ValidationResult>();

        return Validator.TryValidateObject(obj, validationContext,
                                           validationResults, true);
}
```

Shadow Properties

Shadow Properties

- Shadow properties are **not defined** in your **.NET entity class**
  - They are **useful** when there is data in the database that **should not be exposed** on the mapped entity types.
- **Configure** shadow property:

```
builder.Entity<Project>()
       .Property<DateTime>("LastUpdated");
```

- **Change** value of shadow property:

```
context.Entry(Project).Property("LastUpdated")
       .CurrentValue = DateTime.Now;
```

Summary

- Objects can be composed from other objects to represent complex relationships
- Navigation properties speed up the traversal of related entities

Summary

- The **Fluent API** gives us full control over Entity Framework object mappings
- Information overhead can be limited by **selecting** only the needed properties
- **ResultModels** can be used to move aggregated data between methods
- **Attributes** can be used to express special table relationships and to customize entity behaviour

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Trainings @ Software University (SoftUni)

SoftUni Foundation

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

INDEAVR
Serving the high achievers

SUPERHOSTING.BG

SoftUni Organizational Partners

SoftUni Foundation

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license