Entity Framework Core - Introduction

The ORM Concept

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg/

Table of Contents

Have a Question?

Software University

sli.do

#csharp-db

Entity Framework Core

Overview and Features

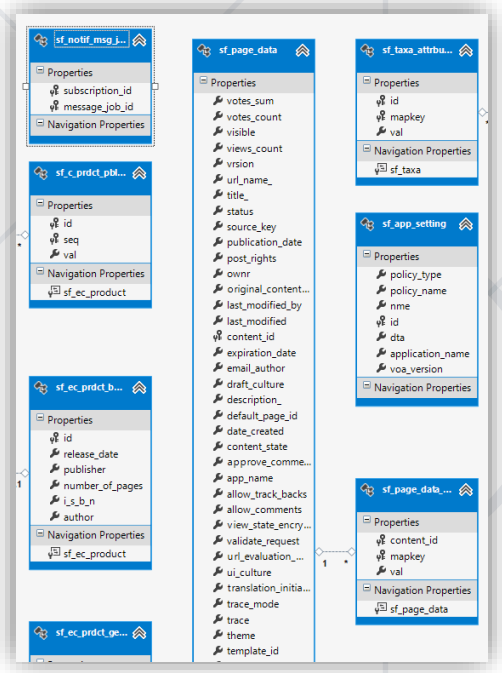
Entity Framework Core: Overview

- The standard **ORM framework** for **.NET** and **.NET Core**
- Provides LINQ-based data queries and **CRUD** operations
- Automatic **change tracking** of in-memory objects
- Works with many relational databases (with different providers)
- Open source with independent release cycle

EF Core: Basic Workflow (1)

Software University

1. Define the data model (**Code First** or **Scaffold from DB**)



2. Write & execute query over **IQueryable**

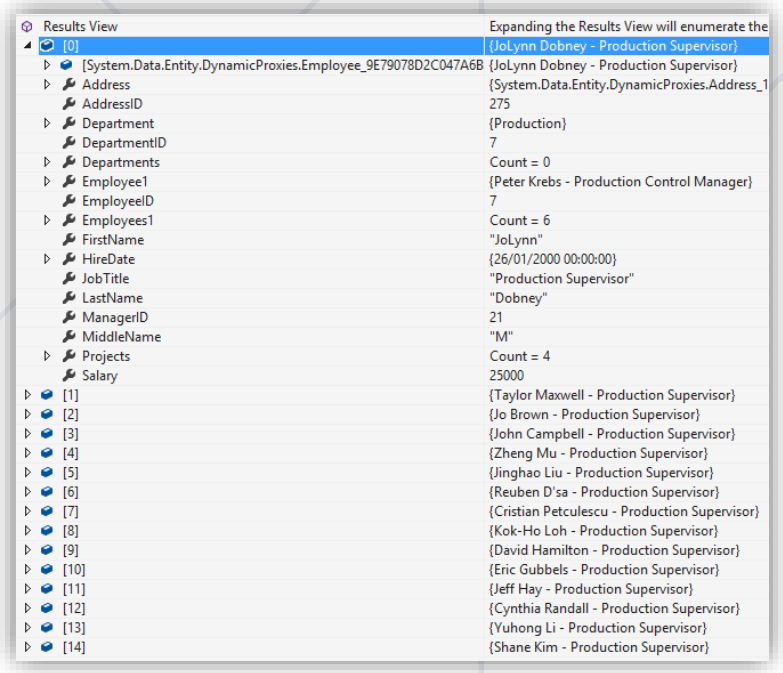
```
var toolName = "";

var snippetOptions = DefaultToolGroup
    .Tools
    .OfType<EditorListToll>()
    .Where(t =>
        t.Name == toolName &&
        t.Items != null &&
        t.Items.Any())
    .SelectMany(
        (t, index) =>
            t.Items
            .Select(item =>
                new
                {
                    text = item.Text,
                    value = item.Value
                }));

if(snippetOptions.Any())
{
    options[toolName] = snippetOptions;
}
```

3. EF generates & executes an **SQL query** in the **DB**

```
exec sp_executesql N'SELECT
[Filter2].[UserInCourseId] AS [UserInCourse
[Filter2].[UserId] AS [UserId],
[Filter2].[CourseInstanceId1] AS [CourseIns
[Filter2].[FirstCourseGroupId] AS [FirstCou
[Filter2].[SecondCourseGroupId] AS [SecondC
[Filter2].[ThirdCourseGroupId] AS [ThirdCou
[Filter2].[FourthCourseGroupId] AS [FourthC
[Filter2].[FifthCourseGroupId] AS [FifthCou
[Filter2].[IsLiveParticipant] AS [IsLivePar
[Filter2].[Accommodation] AS [Accommodatior
[Filter2].[ExcellentResults] AS [ExcellentR
[Filter2].[Result] AS [Result],
[Filter2].[CanDoTestExam] AS [CanDoTestExam
[Filter2].[CourseTestExamId] AS [CourseTest
[Filter2].[TestExamPoints] AS [TestExamPoir
[Filter2].[CanDoPracticalExam] AS [CanDoPra
[Filter2].[CoursePracticalExamId1] AS [Cour
[Filter2].[PracticalExamPoints] AS [Practic
[Filter2].[AttendancesCount] AS [Attendance
[Filter2].[HomeworkEvaluationPoints] AS [Ho
FROM   (SELECT [Extent1].[UserInCourseId] A
AS [SecondCourseGroupId], [Extent1].[ThirdC
[IsLiveParticipant], [Extent1].[Accommodati
[CourseTestExamId], [Extent1].[TestExamPoir
[PracticalExamPoints], [Extent1].[Attendanc
        FROM  [courses].[UsersInCourses] AS
        INNER JOIN [courses].[CoursePractic
        WHERE ( EXISTS (SELECT
                1 AS [C1]
                FROM [courses].[CoursePract
                WHERE [Extent1].[UserInCour
        )) AND ([Extent2].[AllowExamFilesEv
INNER JOIN [courses].[CoursePracticalExams]
WHERE ([Filter2].[UserId] = @p__linq__0) AN
```

EF Core: Basic Workflow (2)

4. EF transforms the query results into .NET objects



Results View	Expanding the Results View will enumerate the
[0]	{JoLynn Dobney - Production Supervisor}
[System.Data.Entity.DynamicProxies.Employee_9E79078D2C047A6B	{JoLynn Dobney - Production Supervisor}
Address	{System.Data.Entity.DynamicProxies.Address_1
AddressID	275
Department	{Production}
DepartmentID	7
Departments	Count = 0
Employee1	{Peter Krebs - Production Control Manager}
EmployeeID	7
Employees1	Count = 6
FirstName	"JoLynn"
HireDate	{26/01/2000 00:00:00}
JobTitle	"Production Supervisor"
LastName	"Dobney"
ManagerID	21
MiddleName	"M"
Projects	Count = 4
Salary	25000
[1]	{Taylor Maxwell - Production Supervisor}
[2]	{Jo Brown - Production Supervisor}
[3]	{John Campbell - Production Supervisor}
[4]	{Zheng Mu - Production Supervisor}
[5]	{Jinghao Liu - Production Supervisor}
[6]	{Reuben D'sa - Production Supervisor}
[7]	{Cristian Petculescu - Production Supervisor}
[8]	{Kok-Ho Loh - Production Supervisor}
[9]	{David Hamilton - Production Supervisor}
[10]	{Eric Gubbels - Production Supervisor}
[11]	{Jeff Hay - Production Supervisor}
[12]	{Cynthia Randall - Production Supervisor}
[13]	{Yuhong Li - Production Supervisor}
[14]	{Shane Kim - Production Supervisor}

5. Modify data with C# code and call **Save Changes()**

```
private void ChangeBlogPostName(int id,
    string newName)
{
    var db = new Context();
    var post = db.Posts
        .FirstOrDefault(x => x.Id == id);

    if(post == null)
    {
        throw new ArgumentNullException(
            "Item with that id was not found");
    }

    post.Name = newName;
    db.SaveChanges();
}
```

6. Entity Framework generates & executes SQL command to modify the DB

```
exec sp_executesql N'SET NOCOUNT ON;
DELETE FROM [Categories]
WHERE [CategoryID] = @p0;
SELECT @@ROWCOUNT;
UPDATE [Categories] SET [CategoryName] = @p1
WHERE [CategoryID] = @p2;
SELECT @@ROWCOUNT;
INSERT INTO [Categories] ([CategoryID], [CategoryName])
VALUES (@p3, @p4),
(@p5, @p6);
```

Entity Framework Core: Setup

- To add **EF Core support** to a project in **Visual Studio**
  - Install it from **NuGet** using Visual Studio or **dotnet CLI**

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

  - EF Core is modular – any **data providers** must be installed too

```
Microsoft.EntityFrameworkCore.SqlServer
```

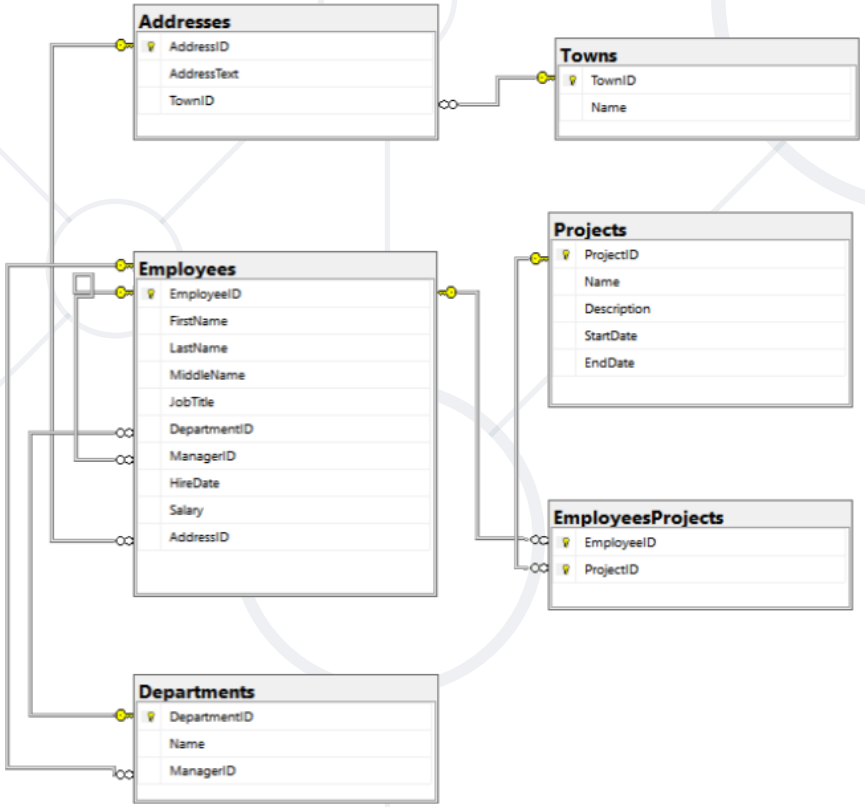
  - To use the Entity Framework Core CLI tools

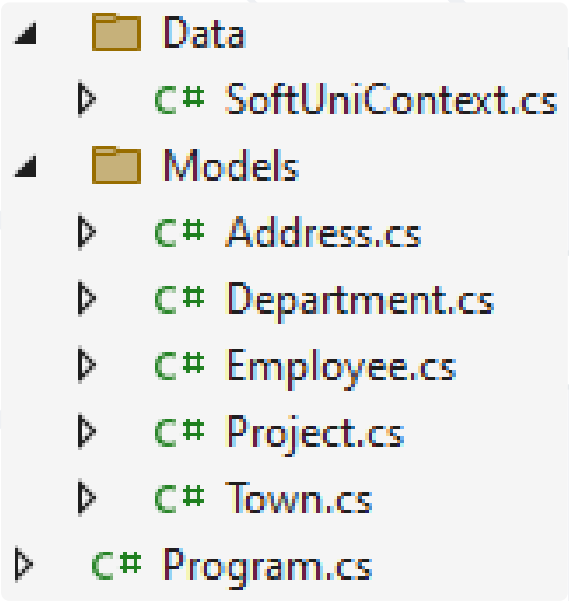
```
dotnet tool install --global dotnet-ef
```

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Database First Model

- **Database First** model models the **entity classes after** the **database**





Database First Model: Setup

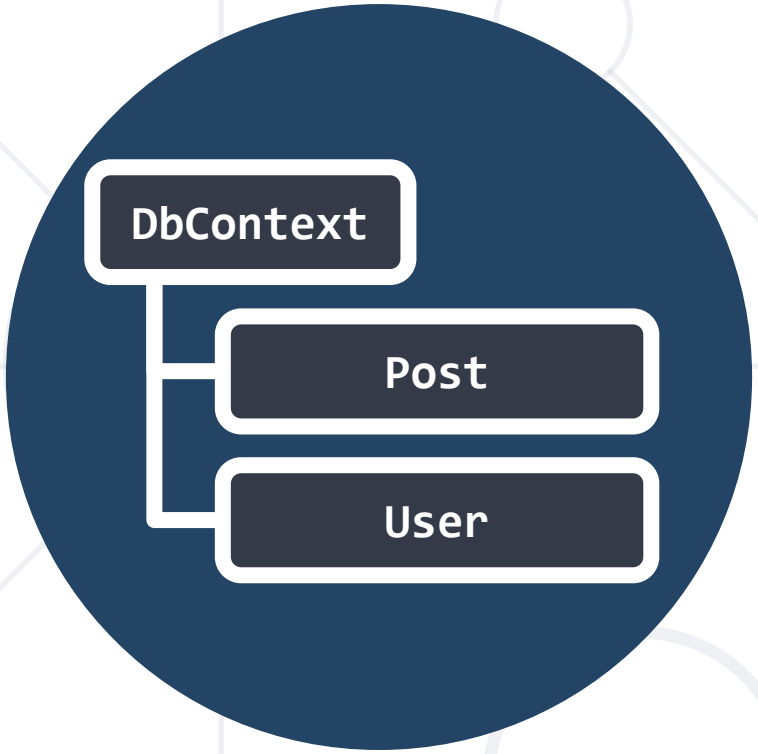
- Scaffolding **DbContext** from DB with **EF Core CLI Tools**

```
dotnet ef dbcontext scaffold "Server=…;Database=…;Integrated
Security=true" Microsoft.EntityFrameworkCore.SqlServer -o Models
```

- To update with the latest database changes, use the **-f** flag
  - To use attributes for configuring the model use the **-d** flag

```
dotnet ef dbcontext scaffold "…" Microsoft… -o Models -f -d
```

- Scaffolding requires the following NuGet packages installed
  - **Microsoft.EntityFrameworkCore.SqlServer**
  - **Microsoft.EntityFrameworkCore.Design**



EF Core Components

Overview of System Objects

Domain Classes (Models)

- Bunch of normal C# classes (POCO)
  - May contain **navigation properties** for **table relationships**

```
public class PostAnswer
{
    public int Id { get; set; }
    public string Content { get; set; }
    public int PostId { get; set; }
    public Post Post { get; set; }
}
```

Primary key

Foreign key

Navigation property

- Recommended to be in a **separate class library**

DbSet Type

- Maps a **collection** of **entities** from a **table**
- Set operations: **`Add`**, **`Attach`**, **`Remove`**, **`Find`**
- **`DbContext`** contains multiple **`DbSet<T>`** properties

```
public class DbSet<TEntity> :
System.Data.Entity.Infrastructure.DbQuery<TEntity>
        where TEntity : class
    Member of System.Data.Entity
```

```
public DbSet<Post> Posts { get; set; }
```

The DbContext Class

- Usually named after the database, e.g., `BlogDbContext`, `ForumDbContext`
- Inherits from `DbContext`
- Manages model classes using `DbSet<T>` type
- Implements **identity tracking**, **change tracking**
- Provides **API** for **CRUD** operations and **LINQ-based** data access
- Recommended to be in a separate class library
  - Don't forget to reference the EF Core library + any providers
- Use several `DbContext` if you have too much models

Defining DbContext Class - Example

Software University

EF Reference

Models Namespace

```
using Microsoft.EntityFrameworkCore;
using CodeFirst.Data.Models;

public class ForumDbContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
}
```

Reading Data

Querying the DB Using Entity Framework

The DbContext Class

- `DbContext` provides
  - CRUD Operations
    - A way to **access entities**
    - Methods for **creating** new entities (`Add()` method)
    - Ability to **manipulate database data by** modifying **objects**
- Easily navigate through **table relations**
- Executing **LINQ queries** as native **SQL queries**
- Managing database **creation**/**deletion**/**migration**

Using DbContext Class

- First create instance of the **DbContext**

```
var context = new SoftUniDbContext();
```

- In the constructor you can pass a database connection string
- **DbContext** properties
  - **Database** - **EnsureCreated**/**Deleted** methods, DB Connection
  - **ChangeTracker** - Holds info about the **automatic change tracker**
  - All entity classes (tables) are listed as properties
    - e.g., `DbSet<Employee> Employees { get; set; }`

Reading Data with LINQ Query (1)

- Executing **LINQ-to-SQL** query over EF entity

```
using (var context = new SoftUniEntities())
{
  var employees = context.Employees
    .Where(e => e.JobTitle == "Design Engineer")
    .ToArray();
}
```

EF translates this to an SQL query

- **Employees** property in the **DbContext**

```
public partial class SoftUniEntities : DbContext
{
  public DbSet<Employee> Employees { get; set; }
  public DbSet<Project> Projects { get; set; }
  public DbSet<Department> Departments { get; set; }
}
```

Reading Data with LINQ Query (2)

- We can also use **extension methods** for constructing the query

```
using (var context = new SoftUniEntities())
{
  var employees = context.Employees
      .Where(c => c.JobTitle == "Design Engineering")
      .Select(c => c.FirstName)
      .ToList();
}
```

`ToList()` materializes the query

- Find element by **ID**

```
using (var context = new SoftUniEntities())
{
  var project = context.Projects.Find(2);
  Console.WriteLine(project.Name);
}
```

LINQ: Simple Operations (1)

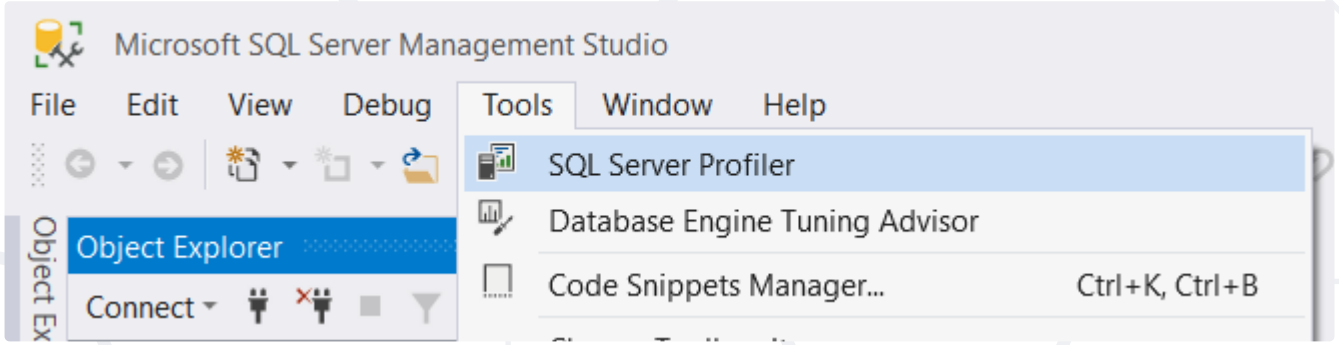
- `Where()`
  - Searches by given condition
- `First()/Last()` / `FirstOrDefault()` / `LastOrDefault()`
  - Gets the **first**/**last** element which matches the condition
  - Throws `InvalidOperationException` without `OrDefault`
- `Select()`
  - Projects (conversion) collection to another type
- `OrderBy()` / `ThenBy()` / `OrderByDescending()`
  - Orders a collection

LINQ: Simple Operations (2)

- `Any()`
  - Checks if any element matches a condition
- `All()`
  - Checks if all elements match a condition
- `Distinct()`
  - Returns only unique elements
- `Skip()` / `Take()`
  - Skips or takes X number of elements

Logging the Native SQL Queries

- Queries sent to SQL Server can be **monitored** with the SQL Server Profiler
  - Included with the **SQL Server** installation



- Queries can be gotten using the built-in `ToQueryString()` method

```
db.Courses.Where(x => x.Title == "EF Core").ToQueryString()
```

CRUD Operations

With Entity Framework

Creating New Data

- To create a new database table row use the method **Add(…)** of the corresponding **DbSet**

```
var project = new Project()
{
  Name = "Judge System",
  StartDate = new DateTime(2023, 1, 26),
};

context.Projects.Add(project);
context.SaveChanges();
```

Create a new **Project** object

Add the object to the **DbSet**

Execute SQL statements

Cascading Inserts

- We can also add cascading entities to the database

```
Employee employee = new Employee();
employee.FirstName = "John";
employee.LastName = "Doe";
employee.Projects.Add(new Project { Name = "SoftUni Conf"} );
softUniEntities.Employees.Add(employee);
softUniEntities.SaveChanges();
```

- The **Project** will be added when the **Employee** entity (employee) is inserted to the database

Updating Existing Data

- `DbContext` allows modifying entity properties and persisting them in the database
  - Just load an entity, modify it and call `SaveChanges()`
- The `DbContext` automatically tracks all changes made on its entity objects

```
Employees employee =
  softUniEntities.Employees.First();
employees.FirstName = "Alex";
context.SaveChanges();
```

SELECT the first order

Execute an SQL UPDATE

Deleting Existing Data

- Delete is done by **Remove()** on the specified entity collection
- **SaveChanges()** method performs the delete action in the database

```
Employees employee =
  softUniEntities.Employees.First();
softUniEntities.Employees.Remove(employee);
softUniEntities.SaveChanges();
```

Mark the entity for deleting at the next save

Execute the SQL DELETE command

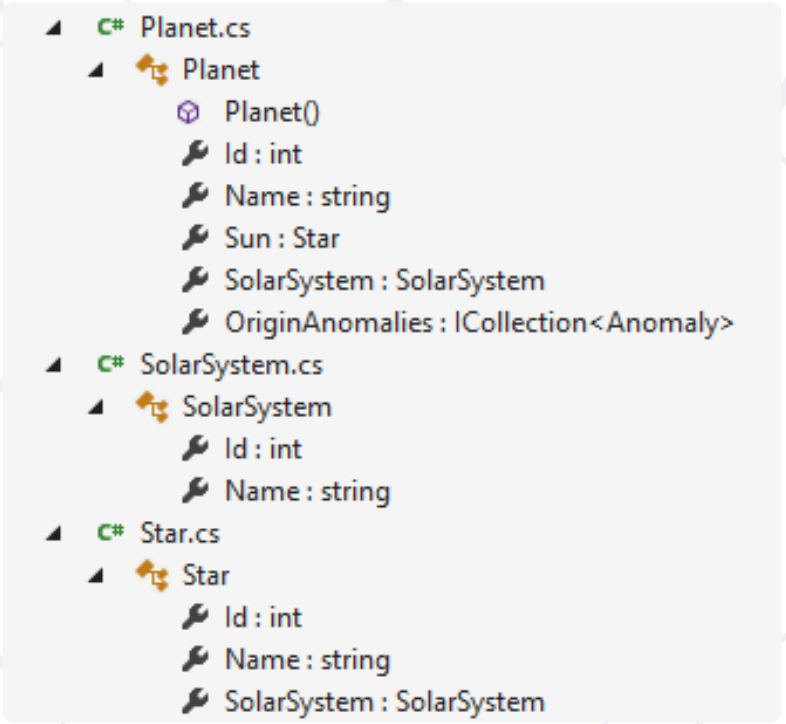
EF Core Configuration

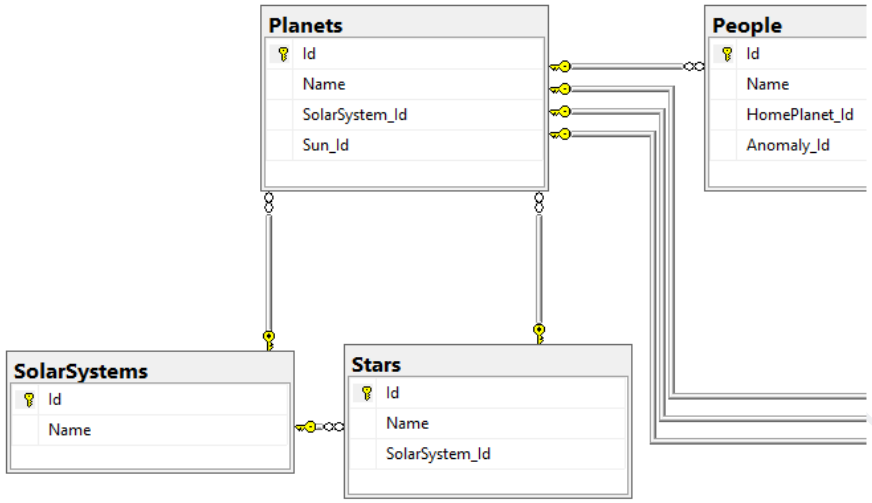
NuGet Packages, Configuration

What is the Code First Model?

Software University

- **Code First** means to write the .NET classes and let EF Core create the **database** from the **mappings**





Why Use Code First?

- Write code **without** having to define **mappings** in XML or **create** database **tables**
- Define objects in **C# format**
- Enables database persistence with no configuration
- Changes to code can be **reflected** (migrated) in the schema
- **Data Annotations** or **Fluent API** describe properties
  - `Key`, `Required`, `MinLength`, etc.

Code First with EF Core: Setup

- To add EF Core support to a project in Visual Studio
  - Install it from **Package Manager Console**

```
Install-Package Microsoft.EntityFrameworkCore
```

  - Or using **.NET Core CLI**

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

- EF Core is modular – any **data providers** must be installed too

```
Microsoft.EntityFrameworkCore.SqlServer
```

How to Connect to SQL Server?

Software University

- One way to connect is to create a **Configuration** class with your connection string

```
public static class Configuration
{
  public const string ConnectionString = "Server=.;Database=…;";
}
```

- Then add the connection string in the **OnConfiguring** method in the **DbContext** class

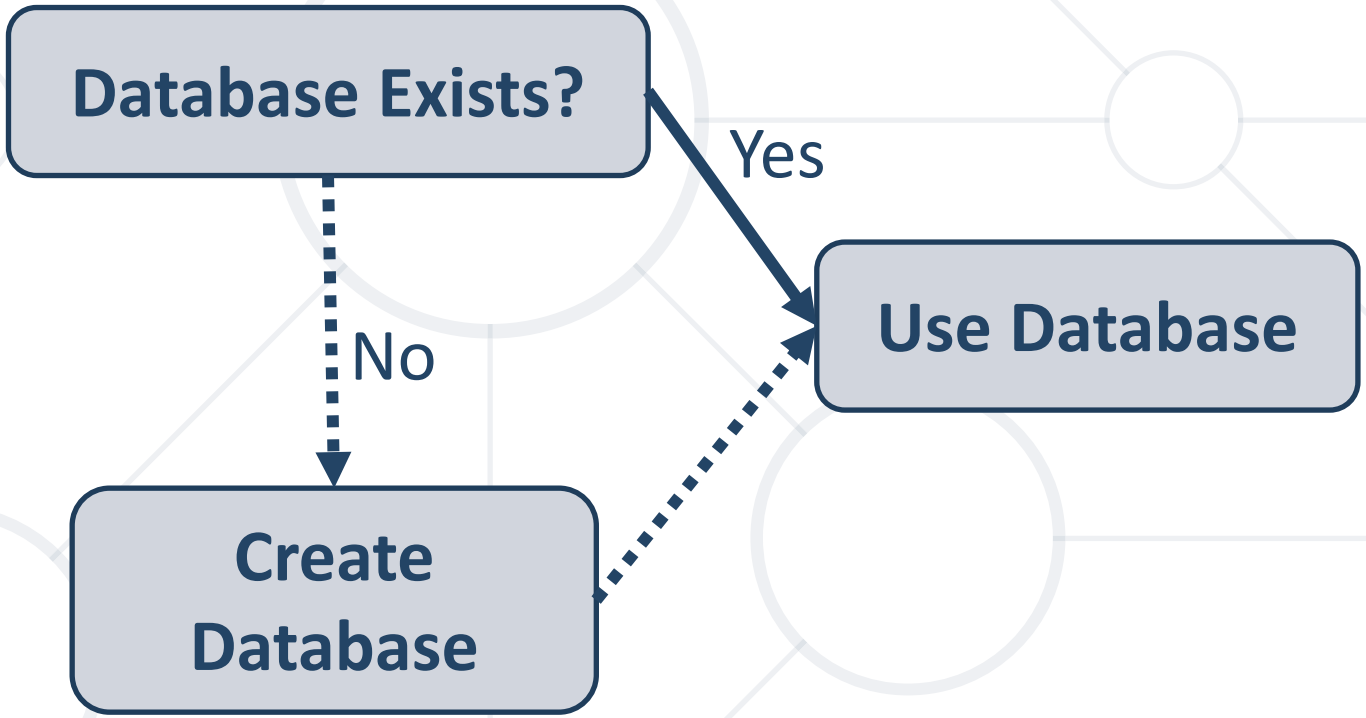
```
protected override void OnConfiguring(DbContextOptionsBuilder builder)
{
  if (!builder.IsConfigured)
    builder.UseSqlServer(Configuration.ConnectionString);
}
```

Fluent API

- The **OnModelCreating** method let us use the Fluent API to describe our **table relations** to **EF Core**

```
protected override void OnModelCreating(ModelBuilder builder)
{
  builder.Entity<Category>()
    .HasMany(c => c.Posts)
    .WithOne(p => p.Category);

  builder.Entity<Post>()
    .HasMany(p => p.Replies)
    .WithOne(r => r.Post);

   builder.Entity<User>()
    .HasMany(u => u.Posts)
    .WithOne(p => p.Author);
}
```

Database Connection Workflow

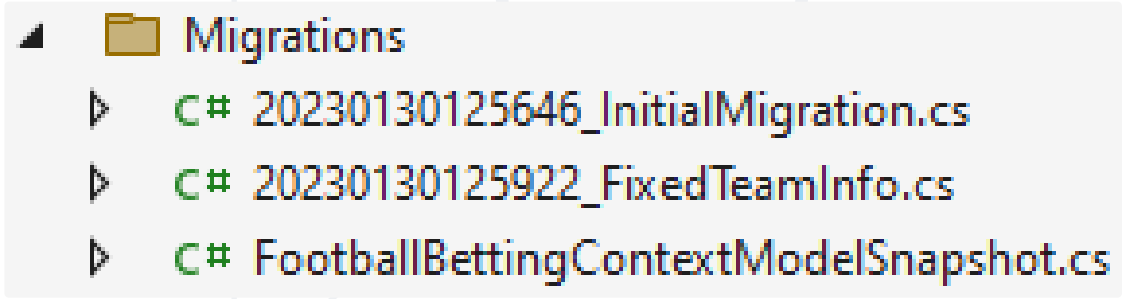


Database Migrations

What Are Database Migrations?

Software University

- Updating database schema **without losing data**
  - Adding/dropping tables, columns, etc.
- Migrations in EF Core keep their **history**
  - Entity Classes, DB Context versions are all **preserved**
- **Automatically** generated



Migrations in EF Core

- To use migrations in EF Core, we use the **`dotnet ef migrations add`** command from the EF CLI Tools

```
dotnet ef migrations add {MigrationName}
```

- To undo a migration, we use **`migrations remove`**

```
dotnet ef migrations remove {MigrationName}
```

- Commit changes to the database

```
dotnet ef database update
```

```
db.Database.Migrate()
```

Summary

- ORM frameworks maps database schema to objects in a programming language
- Entity Framework Core is the standard .NET ORM
- LINQ can be used to query the DB through the DB context

Questions?

SoftUni

Software University | SoftUni Svetlina | SoftUni Creative | SoftUni Digital | SoftUni Foundation | SoftUni Kids

SoftUni Diamond Partners

Software University

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg