Entity Framework Core – Introduction

The ORM Concept

SoftUni Team

Technical Trainers

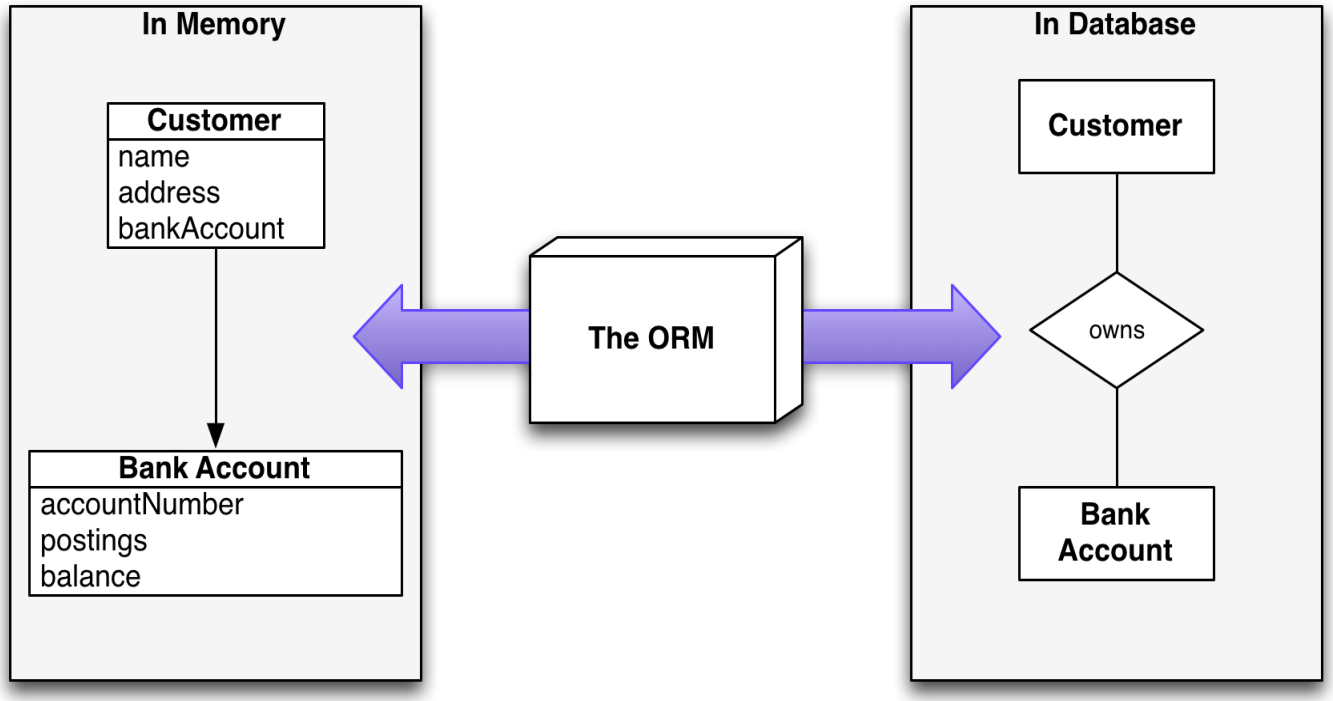
Software University

SoftUni

Software University

https://about.softuni.bg/

Table of Contents

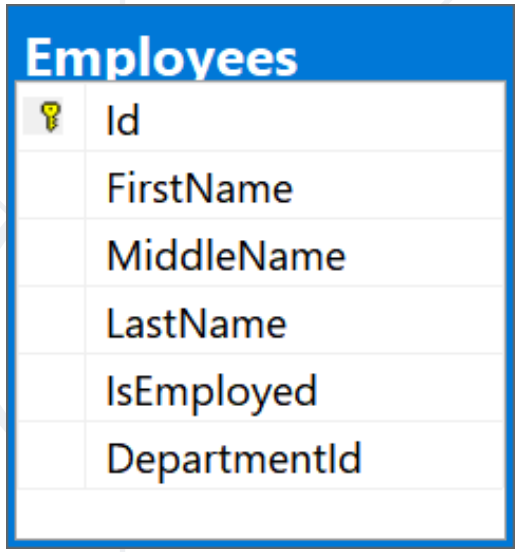


Introduction to ORM

Object-Relational Mapping

What is ORM?

- **Object-Relational Mapping** (ORM) allows manipulating databases **using common classes and objects**
  - **Database Tables → C#/Java/etc. classes**




```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string MiddleName { get; set; }
    public string LastName { get; set; }
    public bool IsEmployed { get; set; }
    public Department Department { get; set; }
}
```

ORM Frameworks: Features

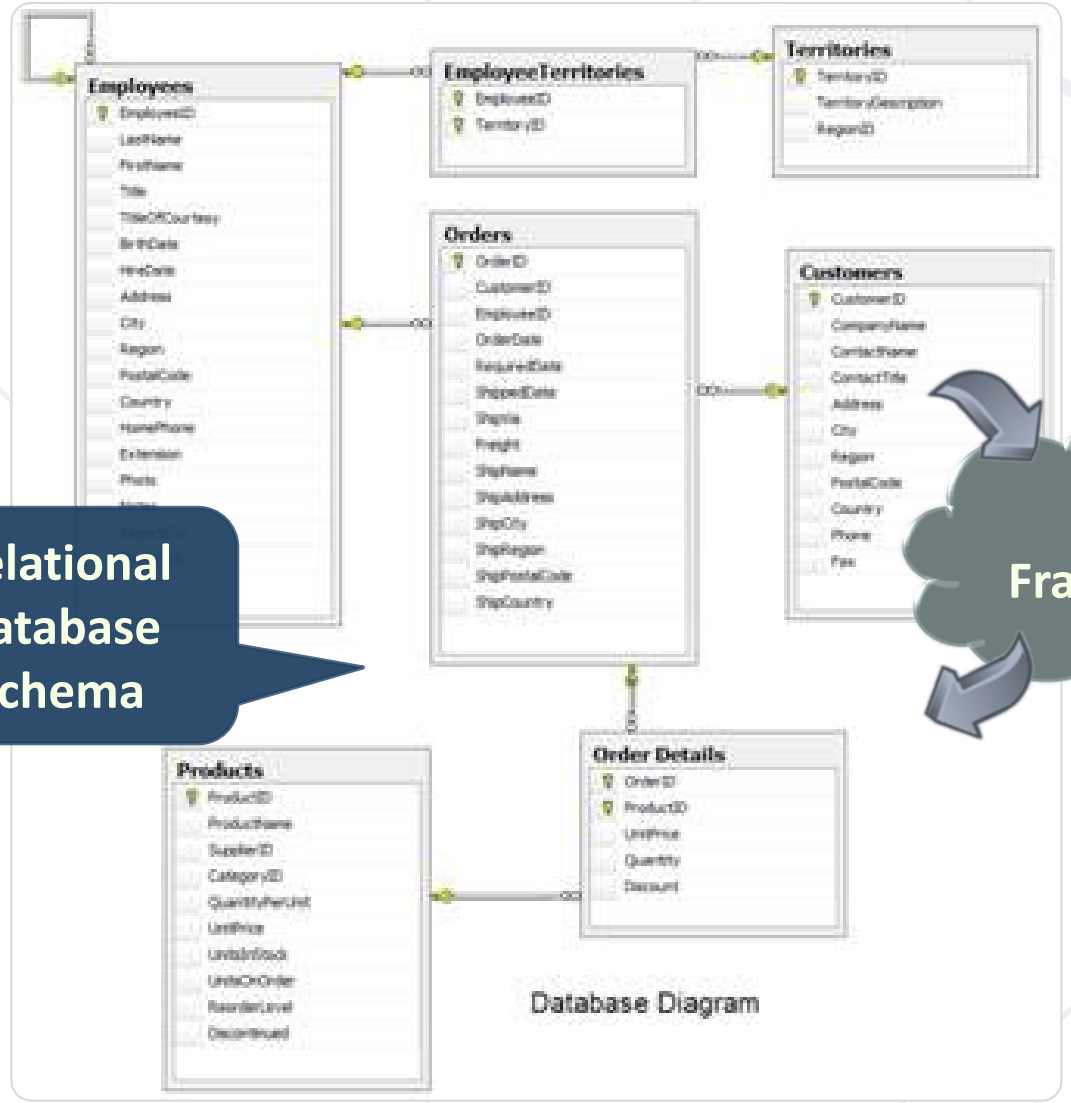
- **ORM frameworks** typically **provide** the following functionality:
  - **Automatically generate SQL** to perform data operations

```
database.Employees.Add(new Employee
{
    FirstName = "Gosho",
    LastName = "Ivanov",
    IsEmployed = true
});
```

```
INSERT INTO Employees
(FirstName, LastName, IsEmployed)
VALUES ('Gosho', 'Ivanov', 1)
```

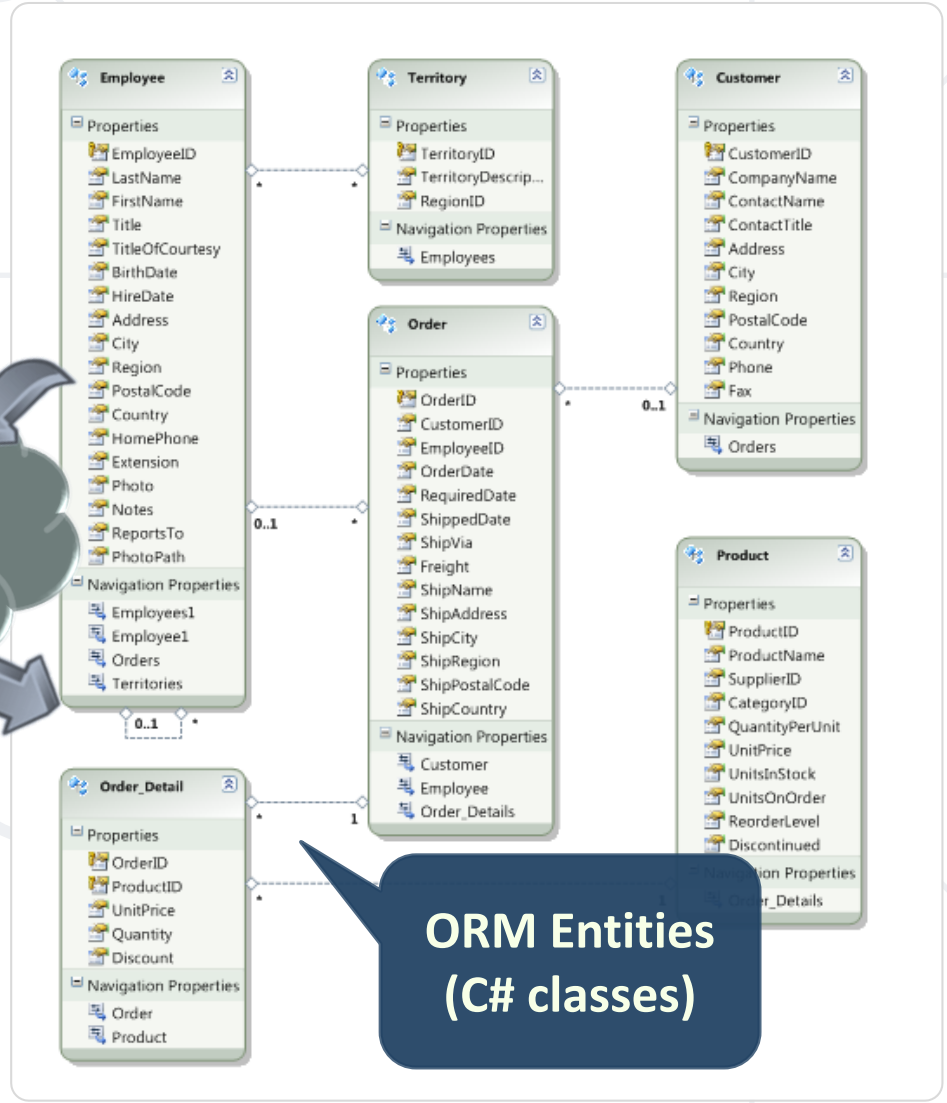
  - **Create database schema from object model** (Code First model)
  - **Create object model from database schema** (DB First model)
  - **Query data by object-oriented API** (e.g. LINQ queries)

ORM Mapping – Example



Relational database schema

ORM Framework



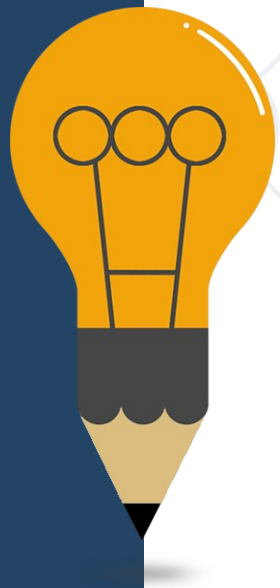
ORM Entities (C# classes)

Entity Framework Core

Overview and Features

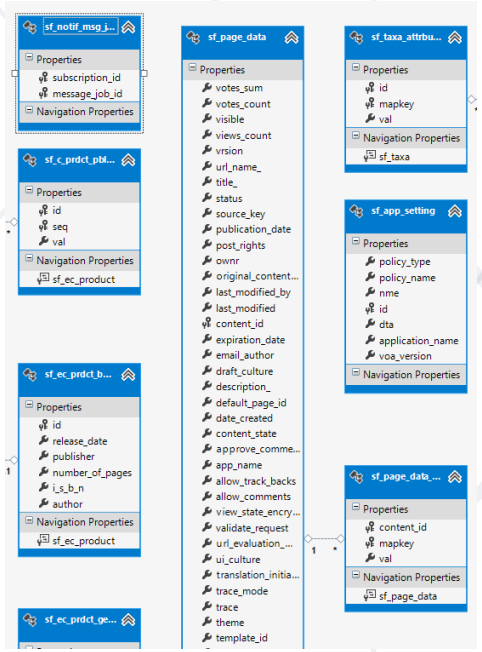
Entity Framework Core: Overview

- 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)

1. Define the data model (**Code First** or **Scaffold from DB**)
2. Write & execute query over **IQueryable**
3. EF generates & executes an **SQL query** in the **DB**




```
var toolName = "";

var snippetOptions = DefaultToolGroup
    .Tools
    .OfType<EditorListTool>()
    .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;
}
```

```
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 ArgumentException(
            "Item with that id was not fo
            "id");
    }

    post.Name = newName;

    db.SaveChanges();
}
```

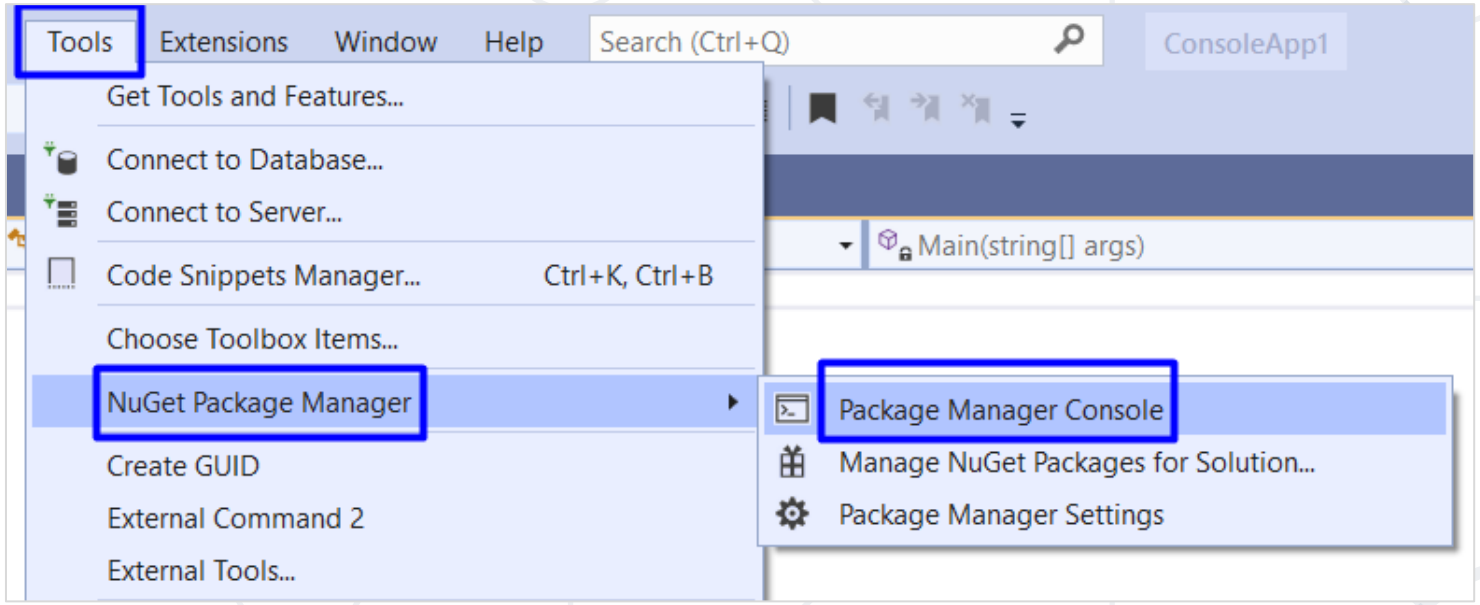
6. Entity Framework generates & executes SQL command to modify the DB

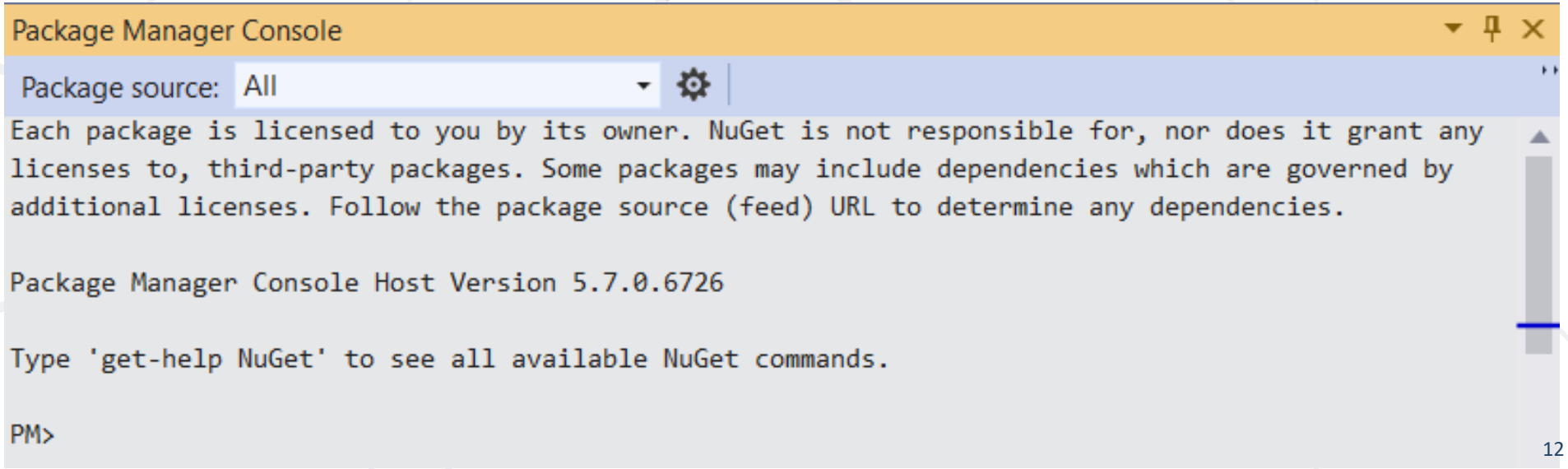
```
SELECT
[Extent1].[EmployeeID] AS [EmployeeID],
[Extent1].[FirstName] AS [FirstName],
[Extent1].[LastName] AS [LastName],
[Extent1].[MiddleName] AS [MiddleName],
[Extent1].[JobTitle] AS [JobTitle],
[Extent1].[DepartmentID] AS [DepartmentID],
[Extent1].[ManagerID] AS [ManagerID],
[Extent1].[HireDate] AS [HireDate],
[Extent1].[Salary] AS [Salary],
[Extent1].[AddressID] AS [AddressID]
FROM [dbo].[Employees] AS [Extent1]
WHERE N'Production Supervisor' = [Extent1].[JobTitle]
```

Database First with EF Core

Generating DBContext and Entity Classes from DB

Package Manager Console





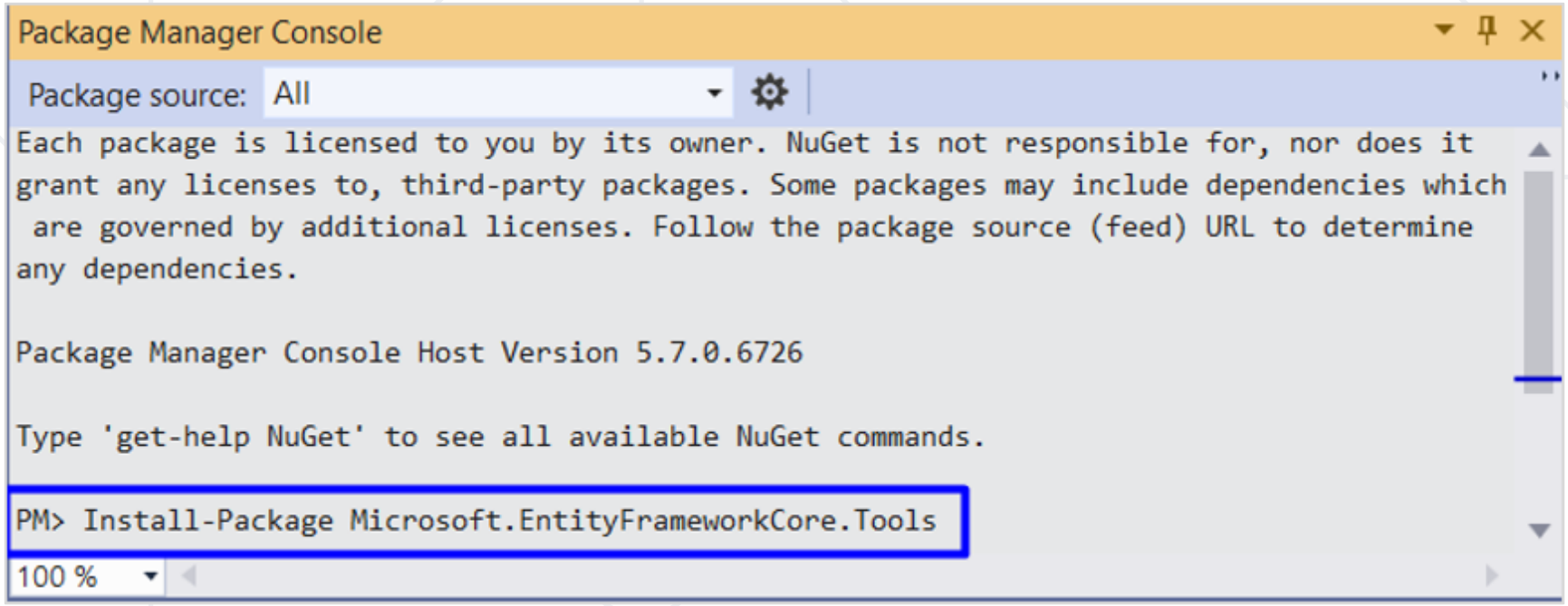
Install EF Packages

- Install the following commands **one by one**:

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

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

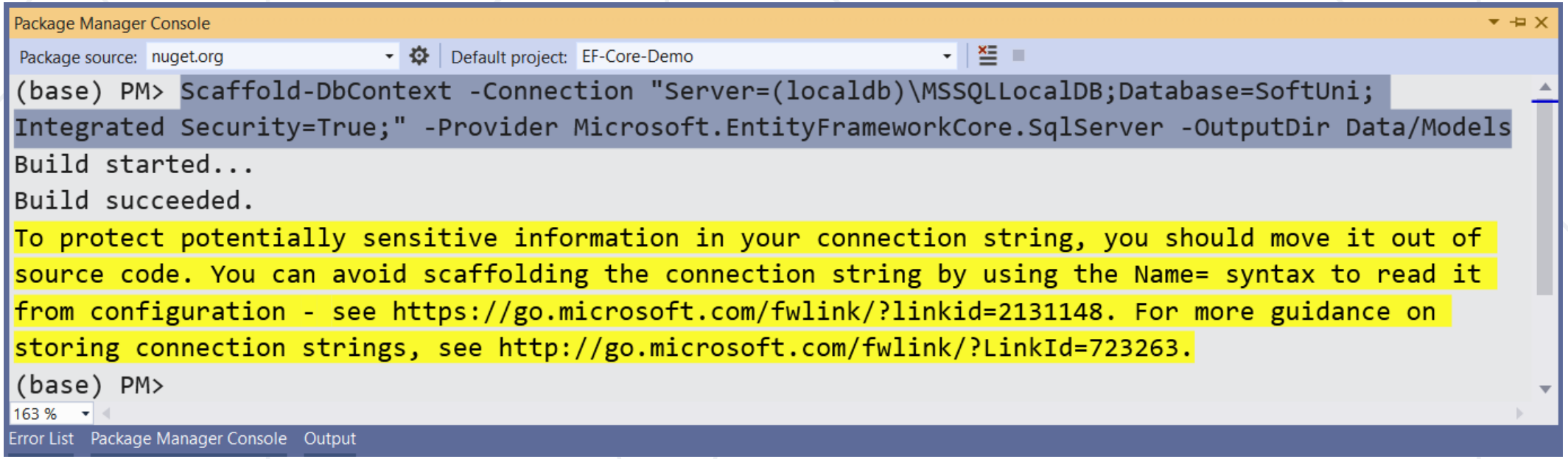
```
Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design
```



Scaffold the Context Class

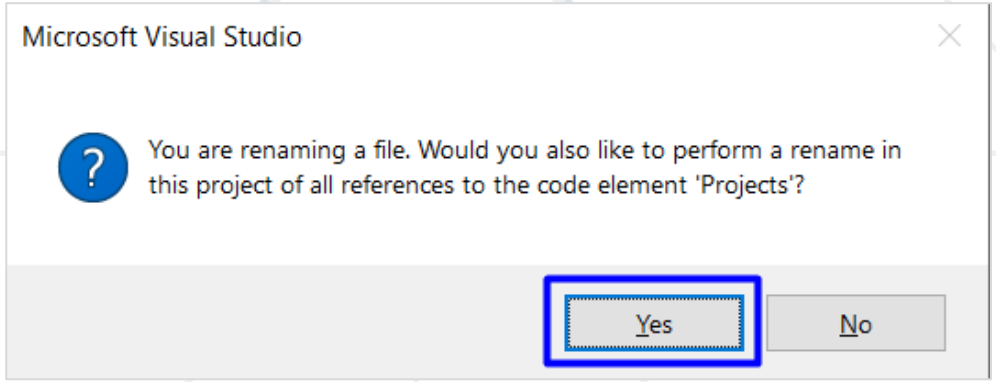
- Execute the following command to **scaffold** the **context class**

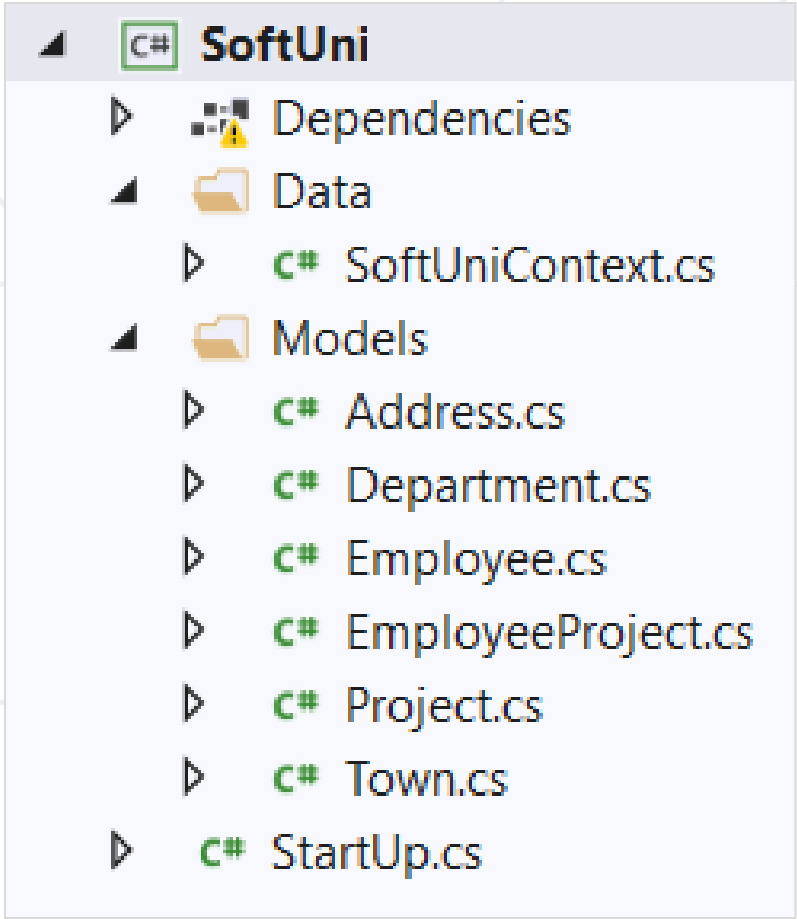
```
Scaffold-DbContext -Connection
"Server=(localdb)\MSSQLLocalDB;Database=SoftUni;Integrated
Security=True;" -Provider
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Data/Models
```



Change Class Structure

- Change file **names** and **structure** to look like this:





- Make sure that your **namespaces** are **exactly the same** as these:

```
SoftUni
SoftUni.Data
SoftUni.Models
```

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 SoftUniContext();
```

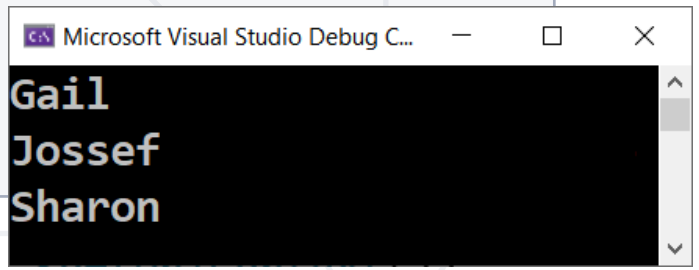
- In the constructor you can pass a database connection string
- **DbContext** properties:
  - **Database** - **EnsureCreated**/**Deleted** methods, DB Connection
  - **ChangeTracker** - Holds info about the **automatic change tracker**
  - All entity classes (tables) are listed as properties
    - e.g.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

Reading Data with LINQ Query

- Executing **LINQ-to-Entities** query over EF entity:

```
public static string FindEmployeesWithJobTitle(SoftUniContext context)
{
    var employees = context.Employees
        .Where(e => e.JobTitle == "Design Engineer")
        .Select(x => x.FirstName)
        .ToList();
    return string.Join(Environment.NewLine, employees);
}
```

EF translates this to an SQL query



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

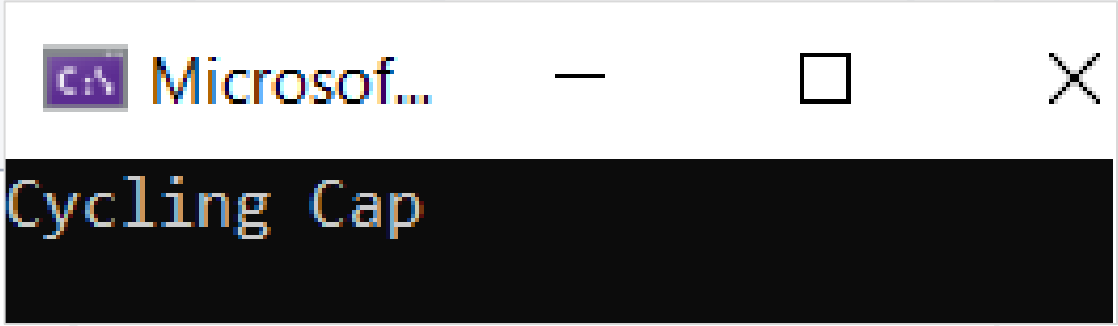
```
public partial class SoftUniContext : DbContext {
  public DbSet<Employee> Employees { get; set; }
  …
}
```

Reading Data with LINQ Query

- Find element by **ID**

```
public static string FindProjectWithId(SoftUniContext
context)
{
    var project = context.Projects.Find(2);

    return project.Name;
}
```



CRUD Operations

With Entity Framework

Creating New Data

- To create a new database row use **DbSet.Add(…)**:

```
public static void CreateNewProject(SoftUniContext context)
{
    var project = new Project()
    {
        Name = "Our Newest Project",
        StartDate = new DateTime(2021, 1, 1),
    };
    context.Projects.Add(project);
    context.SaveChanges();
}
```

Create a new **Project** object

Add the object to the **DbSet**

Execute SQL statements

Updating Existing Data

- **DbContext** allows modifying entities and persisting them in the DB
  - Just load an entity, modify it and call **SaveChanges()**
- The **DbContext** automatically tracks all changes on entity objects

```
public static string UpdateFirstEmployee(SoftUniContext context){
    Employee employee = context.Employees.FirstOrDefault();
    if (employee != null)
    {   employee.FirstName = "Alex";
        context.SaveChanges();
        return employee.FirstName;
    } return "";
}
```

SELECT the first in order

Execute an SQL UPDATE

Deleting Existing Data

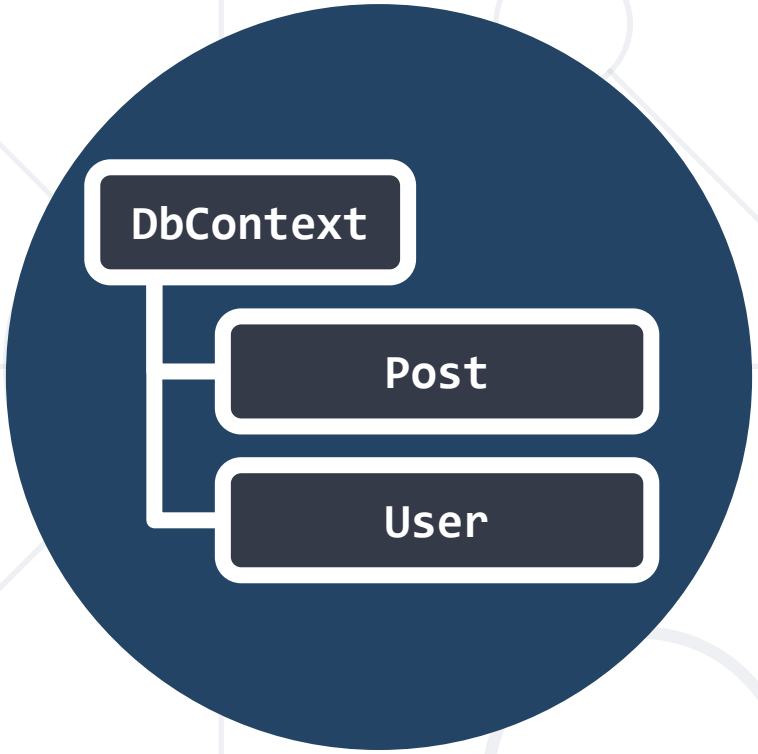
- Delete is done by **Remove()** on the specified entity collection
- **SaveChanges()** method performs the delete action in the DB

```
public static string DeleteFirstProject(SoftUniContext context)
{
    Project project = context.Projects.FirstOrDefault();
    var entitiesWithProject = context.EmployeesProjects
        .Where(x => x.ProjectId == project.ProjectId).ToList();
    context.EmployeesProjects.RemoveRange(entitiesWithProject);
    context.Projects.Remove(project);
    context.SaveChanges(); return project.Name;
}
```

Execute the SQL DELETE command

Mark the entity for deleting at the next save

Delete entities from EmployeesProjects with this ProjectId



EF Core Components

Overview of System Objects

Domain Classes (Models) (1)

- 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**

Domain Classes (Models) (2)

- Another example of a domain class (model)

```
public class Post
{
  public int Id { get; set; }
  public string Content { get; set; }
  public int AuthorId { get; set; }
  public User Author { get; set; }

  public IList<PostAnswer> Answers { get; set; }
}
```

Navigation property

One-to-Many relationship

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

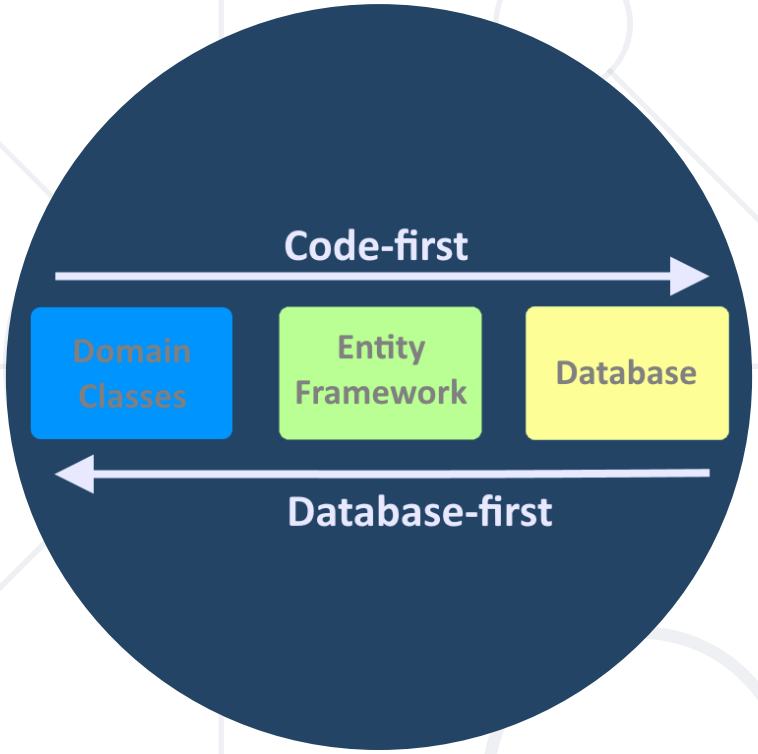
Defining DbContext Class – Example

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

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

EF Reference

Models Namespace

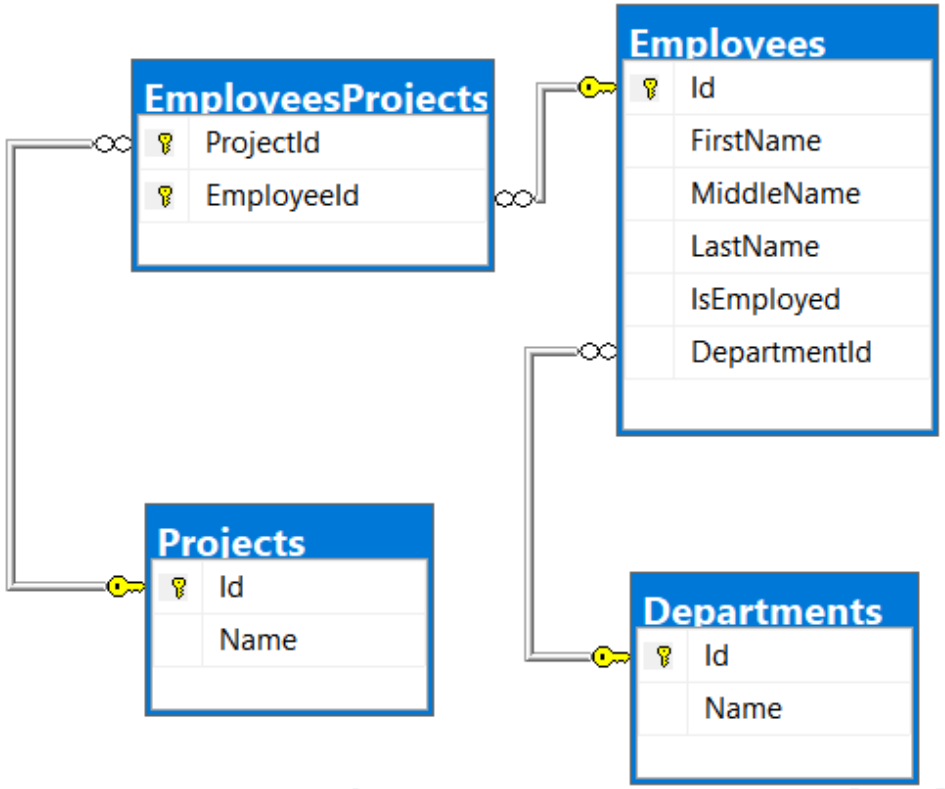


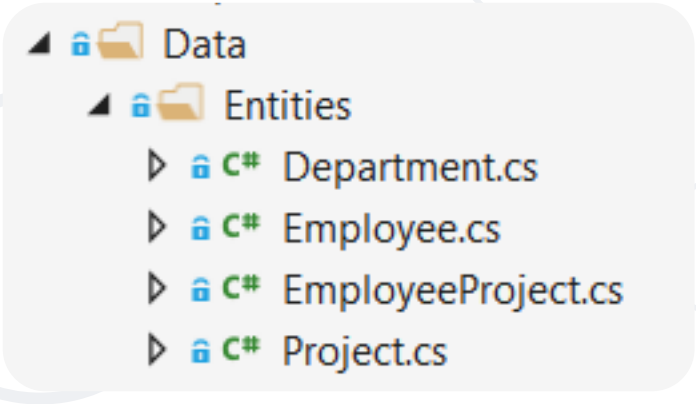
Database First vs Code-first

Approaches to Model the Data Structure

Database First Model

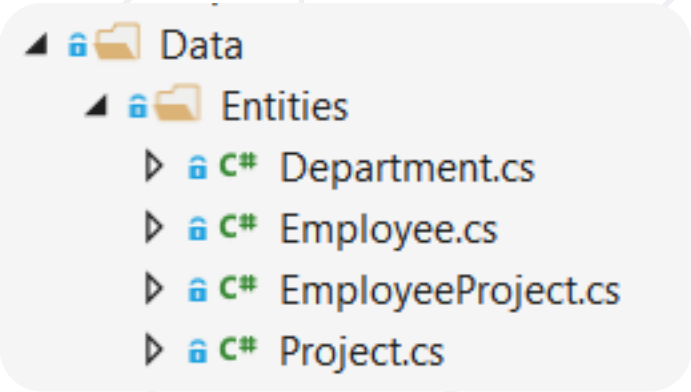
- **Database First model** - models the entity classes after the database

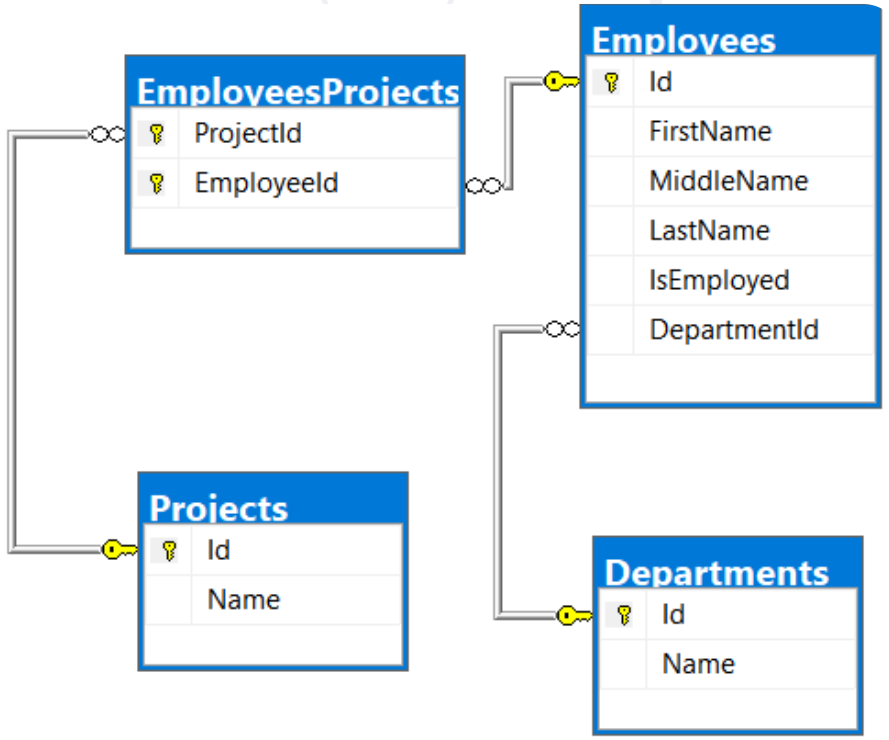


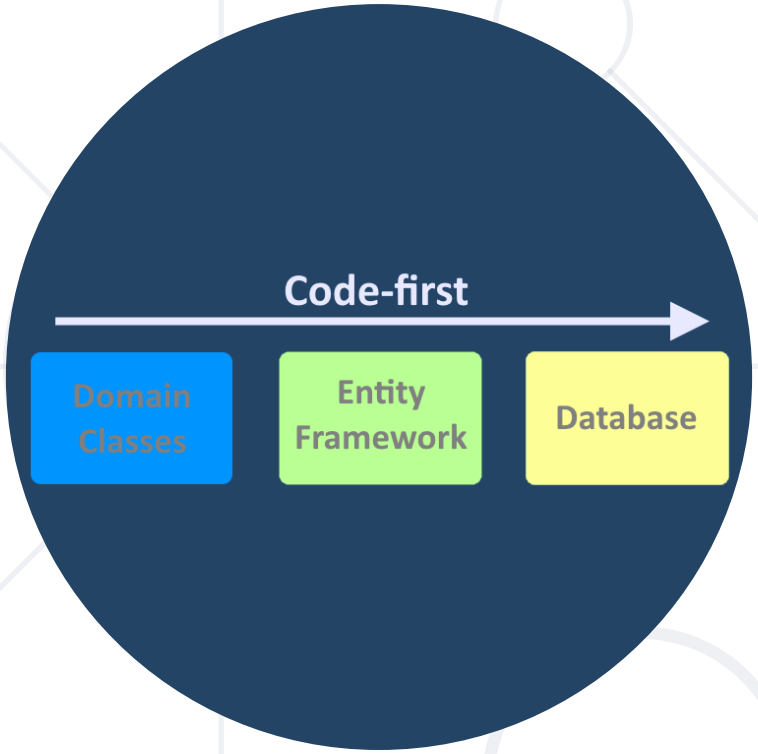


Code-First Model

- **Code-first model** - begins with classes that describe the model and then the ORM generate a database







Code-first

Creates a Database Based on Classes

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?

- 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

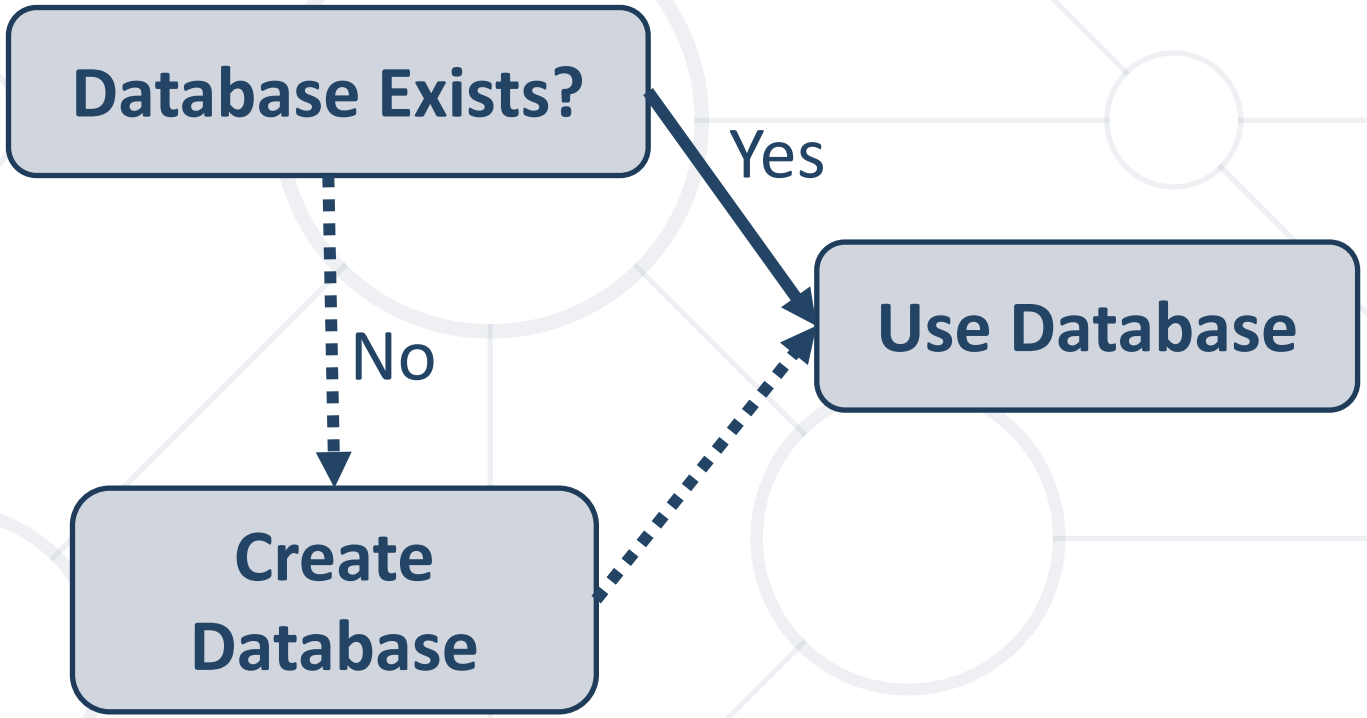
- The **OnModelCreating** Method let us use the Fluent API to describe our **table relations** to **EF Core**

```
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

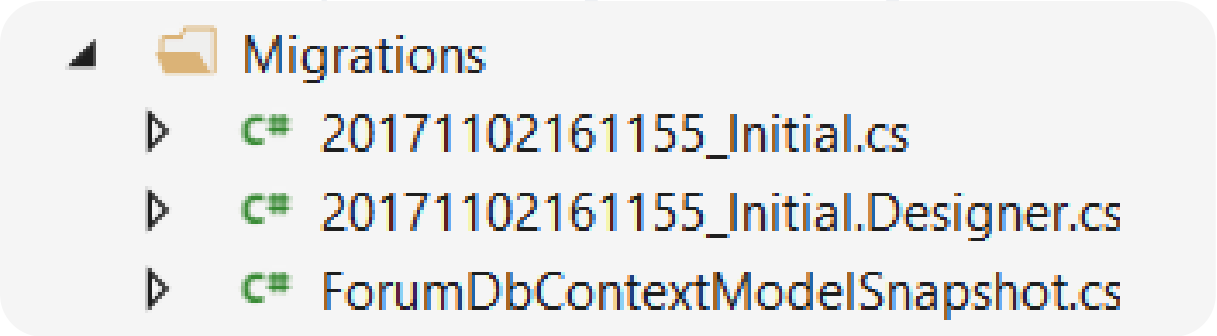


Database Migrations

A Way to Keep the Database Schema in Sync

What Are Database Migrations?

- **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:



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}
```

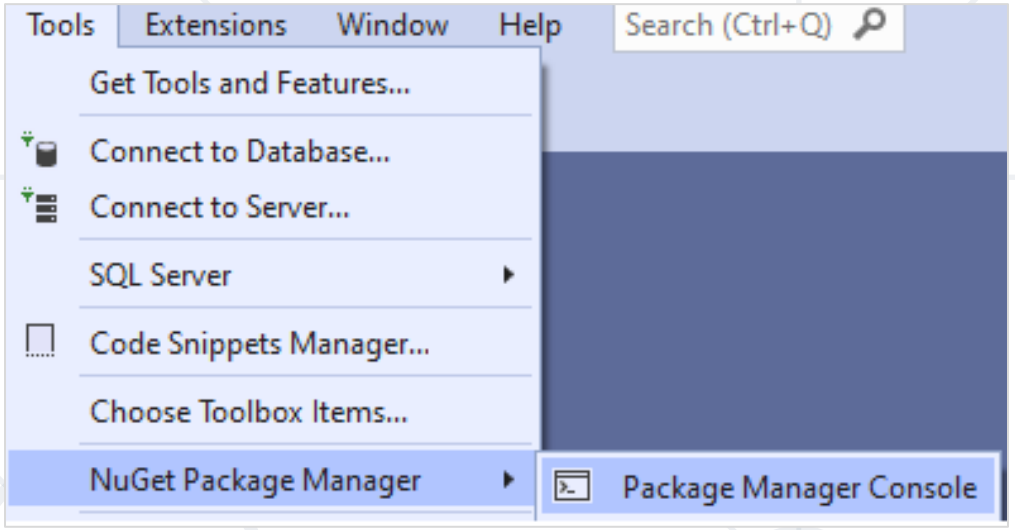
- Commit changes to the database

```
dotnet ef database update
```

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

Package Manager Console Migrations

- **Migration** commands in Entity Framework Core can be executed using the **Package Manager Console**



- **Add** command

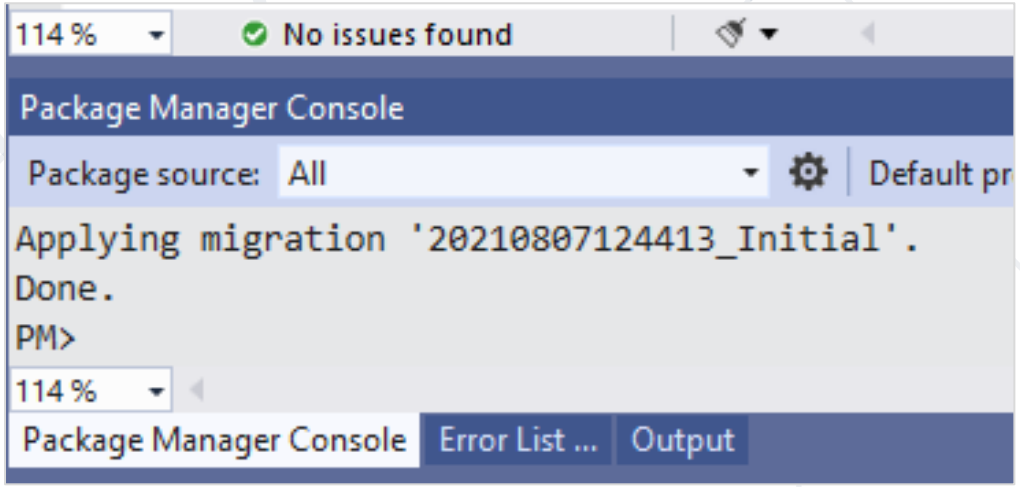
```
Add-Migration {MigrationName}
```

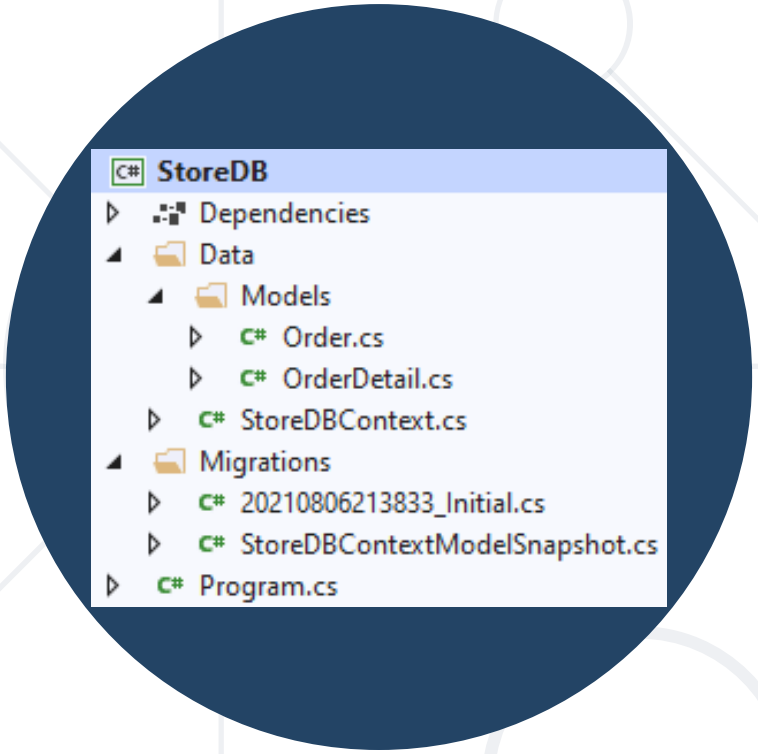
- **Remove** command

```
Remove-Migration
```

- **Commit** changes to the database

```
Update-Database
```

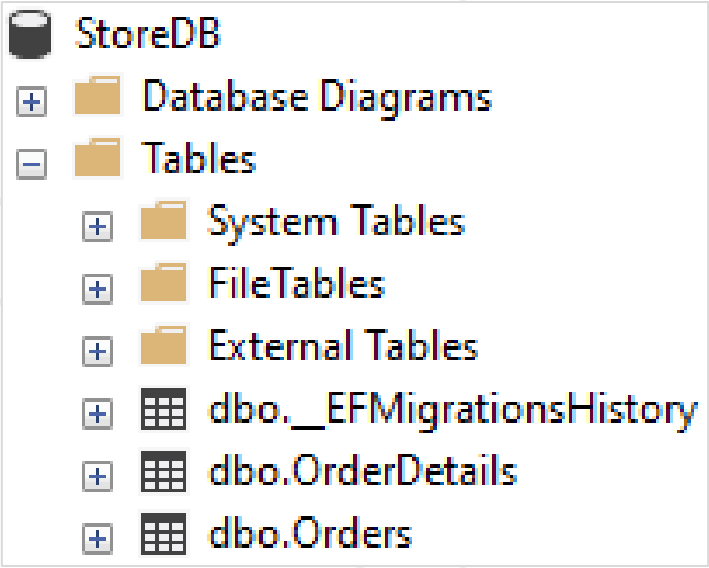




Code First and Migrations

Problem: Store Database

- Use the **Code-First** model to create a **Store database** with two **tables**: **Orders** and **OrderDetails**
- Use **Migrations** to create that database in Microsoft SQL Server



Orders

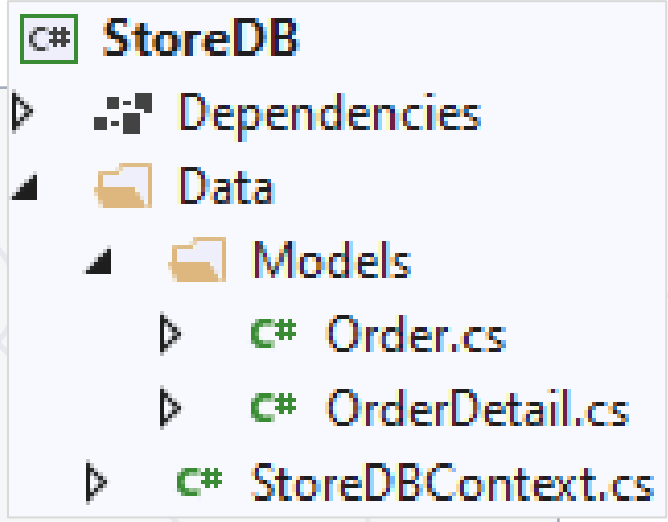
Column Name	Condensed Type	Nullable
OrderID	int	No
Customer	int	No
Employee	int	No
OrderDate	datetime2(7)	No

OrderDetails

Column Name	Condensed Type	Nullable
OrderDetailID	int	No
OrderID	int	No
Product	int	No
Quantity	int	No

Solution: Store Database

```
public class Order
{
  public int OrderID { get; set; }
  public int Customer { get; set; }
  public int Employee { get; set; }
  public DateTime OrderDate { get; set; }
  public List<OrderDetail> OrderDetails { get; set; }
}
```



Solution: Store Database (2)

```
public class OrderDetail
{
  public int OrderDetailID { get; set; }
  public int OrderID { get; set; }
  public Order Order { get; set; }
  public int Product { get; set; }
  public int Quantity { get; set; }
}
```

Solution: Store Database (3)

- **Install** the following **Packages**:

```
Microsoft.EntityFrameworkCore.SqlServer
```

```
Microsoft.EntityFrameworkCore
```

```
public class StoreDBContext : DbContext {
  public DbSet<OrderDetail> OrderDetails { get; set; }
  public DbSet<OrderDetailet<Order> Orders { get; set; }
  protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
  {
    optionsBuilder.UseSqlServer(
      @"Server=(localdb)\mssqllocaldb;Database=StoreDB");
  }
}
```

Solution: Store Database (4)

- To create a database using migrations from your model, **install** the following **packages**:

```
Microsoft.EntityFrameworkCore.Tools
Microsoft.EntityFrameworkCore.Design
```

- Once these packages are installed, run the following command in **Package Manager Console**

```
Add-Migration Initial
```

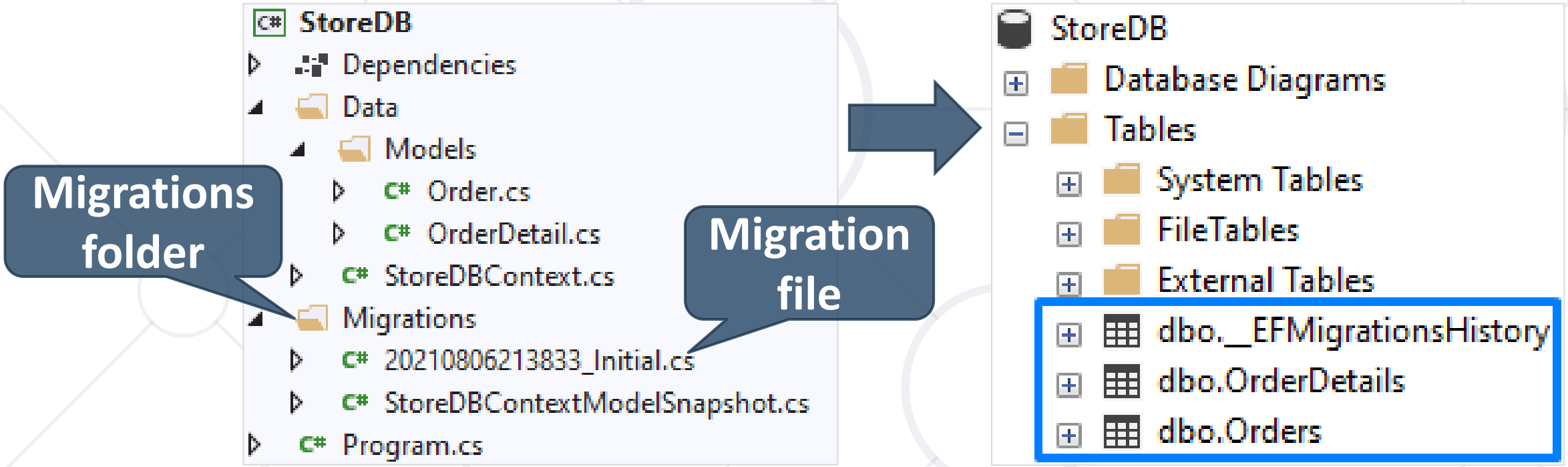
Create the set of tables for your model

- then run the following command

```
Update-Database
```

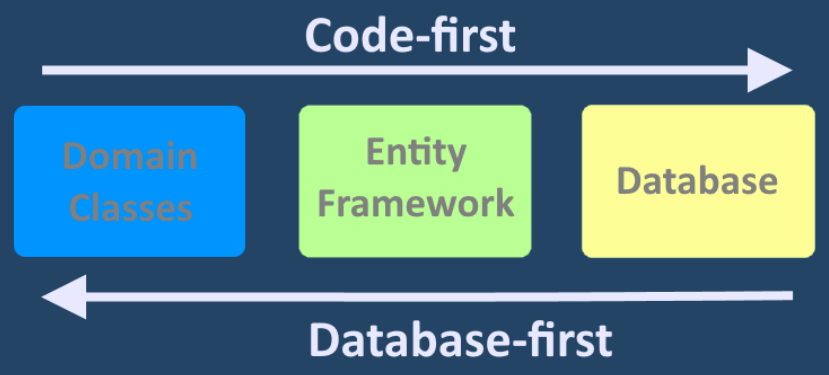
Apply the new migration to the database

Solution: Store Database (5)



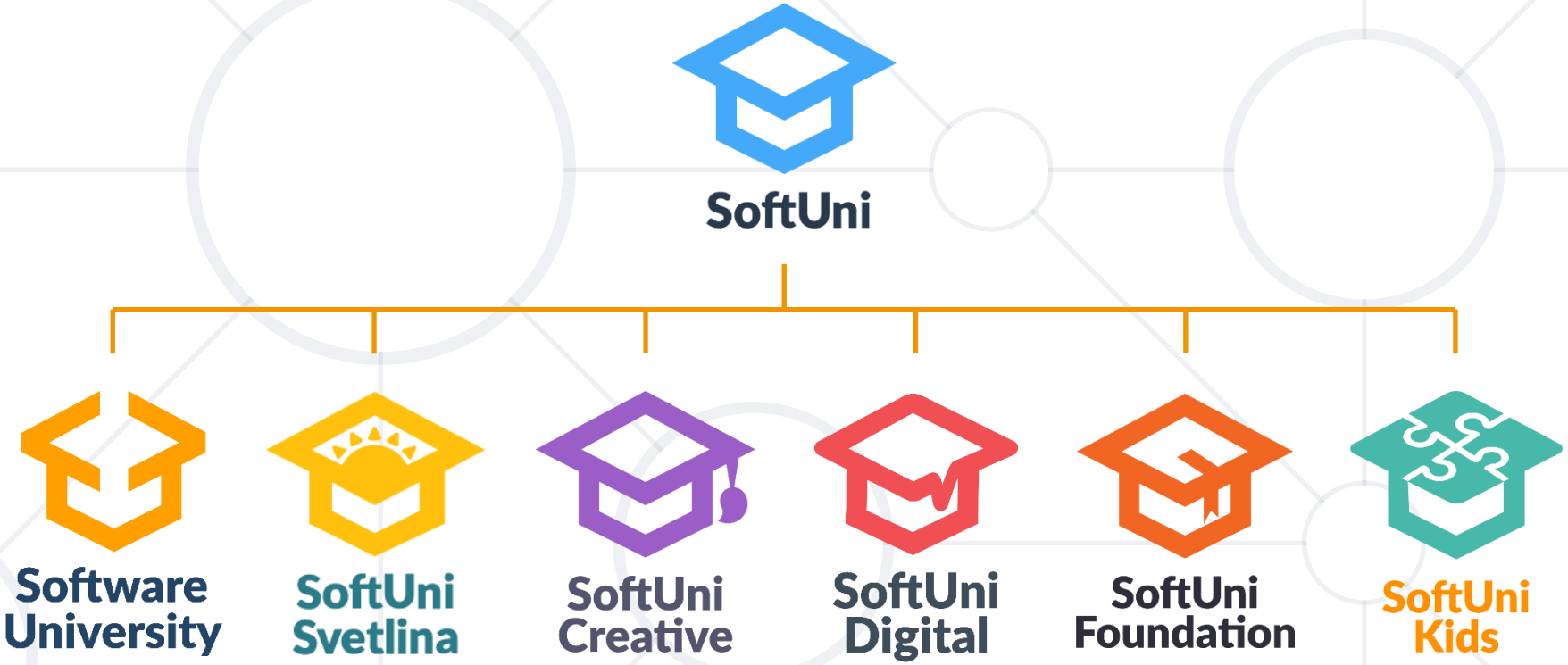
Summary

- **ORM frameworks** maps database schema to objects in a programming language
- **Entity Framework Core** is the standard .NET ORM



- We can use **Database Migrations** to update our database without losing our data

Questions?



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- 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://softuni.org
- © Software University – https://softuni.bg