



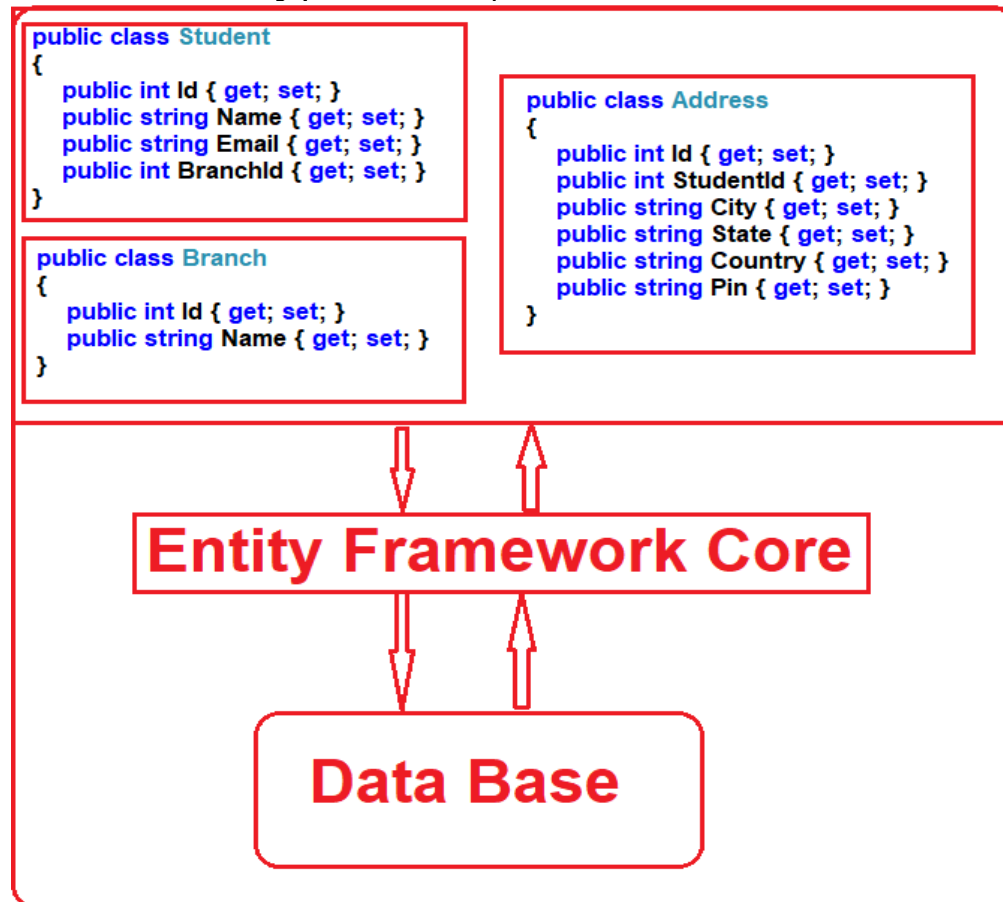
LẬP TRÌNH C# 5

BÀI 1: ENTITY FRAMEWORK CORE – CODE FIRST

- ⦿ Entity Framework Core – Code First
- ⦿ Conventions Rules
- ⦿ Code first và razor view

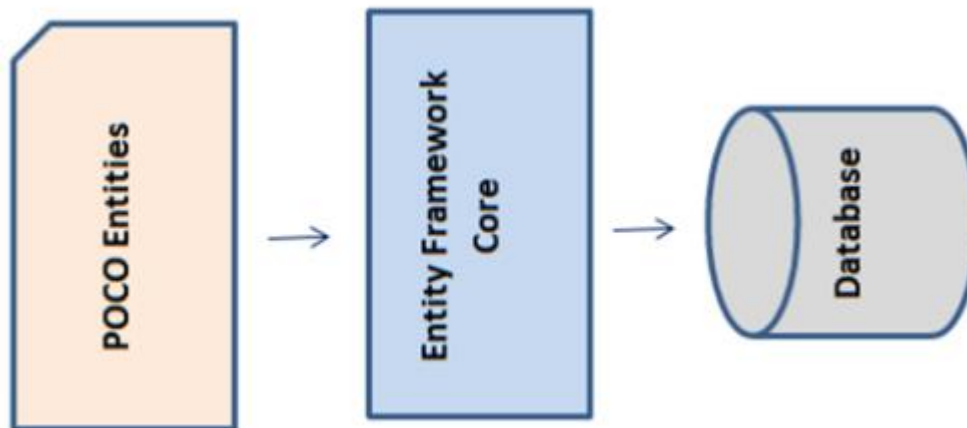


- ❑ Nhắc lại Entity Framework Core: Là ORM (Object-Relational Mapper) Framework đa nền tảng hỗ trợ truy cập data trong .Net (xem bài 7 môn c# 4)

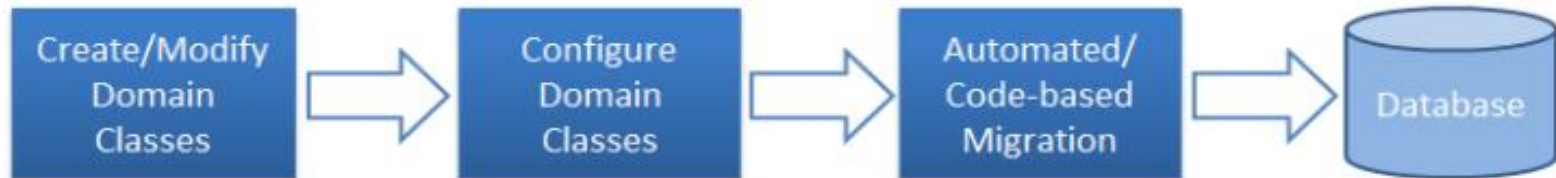


❑ Code First (xem lại bài 6 môn c#3)

- ❖ Phương pháp tạo các class Model sau đó phát sinh ra database
- ❖ Tập trung vào việc thiết kế **Domain** và bắt đầu tạo ra các class theo yêu cầu của **Domain**
- ❖ Cách tiếp cận Code-First là tạo các class (Entities POCO - Plain Old Class Object) và tạo cơ sở dữ liệu mới từ nó

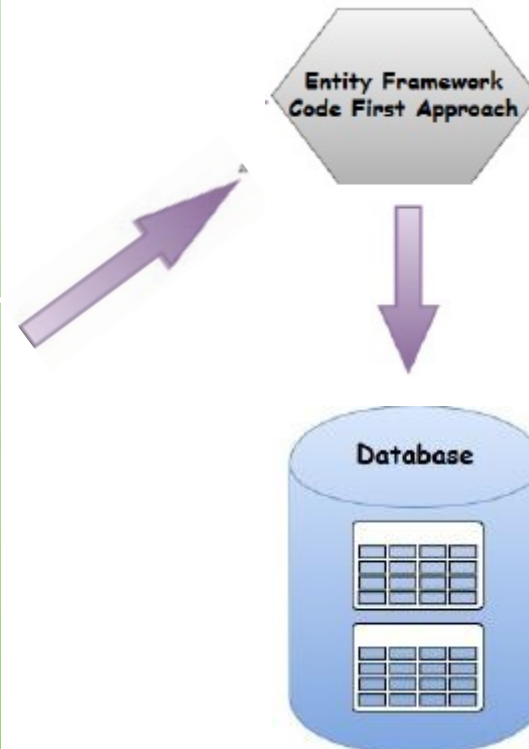
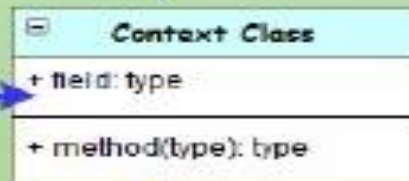
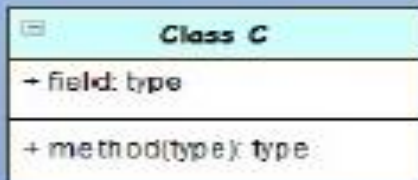
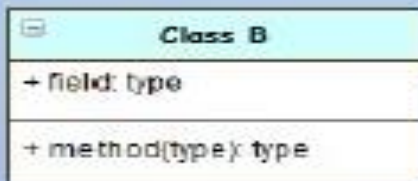
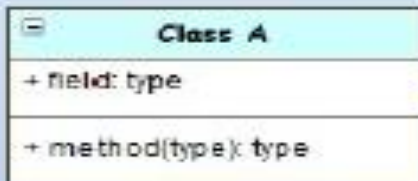


Code-First Workflow




Application

Domain Classes



□ Ưu điểm và nhược điểm

	Code First	Database First
Ưu điểm	<p>Rất phổ biến (vì các lập trình viên thường không thích thiết kế DB, nhưng thích thiết kế class)</p> <p>Kiểm soát hoàn toàn code model, thêm xóa sửa thuộc tính vô cùng dễ dàng</p> <p>Không phải nặng đầu suy nghĩ về DB. Đối với cách tiếp cận này, DB chỉ là cái "cục" data, lòi ra xài thôi</p> <p>Có thể version control Database</p>	<p>Ít phổ biến hơn</p> <p>DB có thể được develop riêng</p> <p>Dùng được DB có sẵn</p> <p>Entity Framework sẽ tạo ra các Entity class cho bạn</p>
Nhược điểm	<p>Các thay đổi cấu trúc trực tiếp trên DB sẽ mất</p> <p>Khó kiểm soát những column sẽ tạo trên Db</p> <p>Hơi khó khi kết hợp với Db có sẵn </p>	<p>Không thể thay đổi code đã được Generate (nó sẽ mất trong lần chỉnh sửa cấu trúc DB tiếp theo)</p> <p>Khó khăn khi muốn thêm các DataAttribute và DisplayAttribute cho các class model</p> <p>Bạn phải nhức đầu suy nghĩ khi muốn biểu diễn các kiểu quan hệ cha con của class</p> <p>Mỗi lần thay đổi cấu trúc DB, bạn sẽ phải update lại EDMX và tạo lại các class Model để phản ánh sự thay đổi đó</p>

□ Tạo mới project NET Core & ASP.NET Core

.NET Core

ASP.NET Core 3.1



Empty

An empty project template for creating an ASP.NET Core application. This template does not have any content in it.



API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.



Web Application

A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.



Web Application (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.



Angular

Authentication

No Authentication

[Change](#)

Advanced

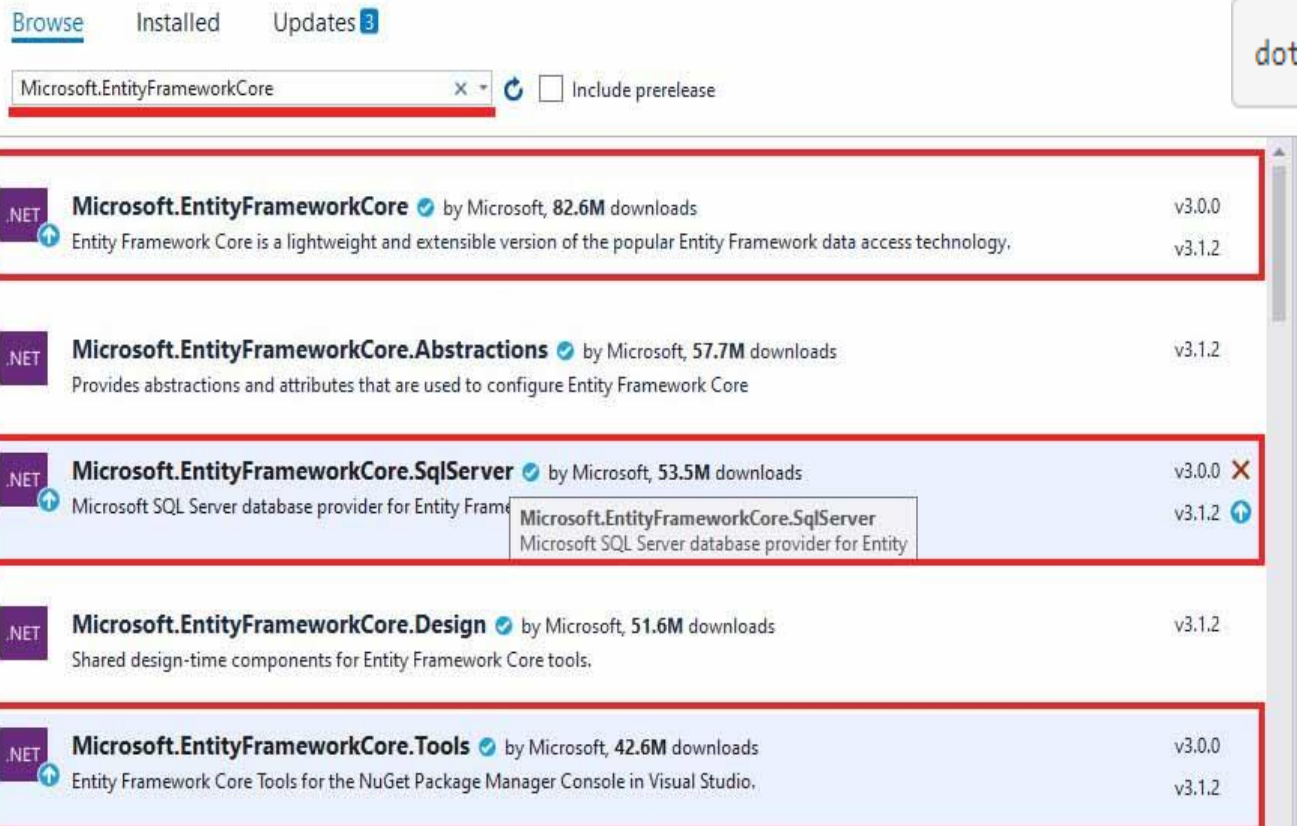
☒ Configure for HTTPS

☐ Enable Docker Support
(Requires [Docker Desktop](#))

Linux

❑ Database được tạo và quản lý bởi Ef-Core code first bằng cách cài các NuGet Packages

- ❖ EF Core SQL Server provider : hỗ trợ làm việc với Sql server
- ❖ EF Core Tools: hỗ trợ thực thi các lệnh dạng command (Migration, scaffoldings...). EF Core hỗ trợ 2 tool là .NET Core command-line interface (CLI) và Package Manager Console (PMC)



The screenshot shows the NuGet package manager interface with the search results for 'Microsoft.EntityFrameworkCore'. The search bar at the top contains 'Microsoft.EntityFrameworkCore'. Below the search bar, there are five packages listed, each with a red box highlighting its name and version information:

- Microsoft.EntityFrameworkCore** (v3.0.0) by Microsoft, 82.6M downloads. Description: Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.
- Microsoft.EntityFrameworkCore.Abstractions** (v3.1.2) by Microsoft, 57.7M downloads. Description: Provides abstractions and attributes that are used to configure Entity Framework Core.
- Microsoft.EntityFrameworkCore.SqlServer** (v3.0.0) by Microsoft, 53.5M downloads. Description: Microsoft SQL Server database provider for Entity Framework Core.
- Microsoft.EntityFrameworkCore.Design** (v3.1.2) by Microsoft, 51.6M downloads. Description: Shared design-time components for Entity Framework Core tools.
- Microsoft.EntityFrameworkCore.Tools** (v3.0.0) by Microsoft, 42.6M downloads. Description: Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

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


❑ Khai báo trực tiếp connection strings trong DbContext

```
public class CompanyContext : DbContext
{
    public DbSet<Information> Information { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            optionsBuilder.UseSqlServer(@"Server=Thepv;Database=Company;Trusted_
        }
    }
}
```

❑ Lưu trữ connection strings: ASP.NET Core có thể đọc các file cấu hình từ nhiều nguồn như appsettings.json, user secrets, environment variables, command line arguments → tùy chọn nơi lưu connection strings (thông thường là appsettings.json)

```
{
  "ConnectionStrings" :
  {
    "MySQLConnection": "server=localhost;uid=root;pwd=12345;database=EFCore",
    "SQLConnection": ""Server=(THEPV)\\MSSQLLocalDB;Database=EFCore;Trusted_Connecti
  }
}
```

- ❑ Truyền Connection string tới DbContext bằng cách tạo lớp kế thừa DbContext, sử dụng DbContextOptions truyền các thông tin như kiểu database, chuỗi kết nối...

```
public class EFContext : DbContext
{
    public EFContext(DbContextOptions options) : base(options)
    {
    }
    public DbSet<Product> Products { get; set; }
}
```

- ❑ Sử dụng phương thức `GetConnectionString` đọc các thông số database từ appsetting

```
var connectionString = Configuration.GetConnectionString("SQLConnection");
```

- ❑ Sử dụng `DbContext` bên trong constructor của Controllers

```
private EFContext db;

public HomeController(EFContext EFContext)
{
    db = EFContext;
}
```

- ❑ Hỗ trợ các lệnh đồng bộ database với domain(entity)
- ❑ Một số lệnh thường dùng migration: add-migration, remove-migration, update-database, Drop-Database, script-migration

PMC Command	CLI Commands	Remarks
Add-Migration	dotnet ef migrations add	Adds a new migration.
Drop-Database	dotnet ef database drop	Drops the database
Get-DbContext	dotnet ef dbcontext info	Gets information about a DbContext type.
Remove-Migration	dotnet ef migrations remove	Removes the last migration
Scaffold-DbContext	dotnet ef dbcontext scaffold	Scaffolds a DbContext and entity types for a database.
Script-Migration	dotnet ef migrations script	Generates a SQL script from migrations.
Update-Database	dotnet ef database update	Updates the database
NA	dotnet ef dbcontext list	Lists available DbContext types.
NA	dotnet ef migrations list	Lists available migrations.

□ Tạo Entity

```
public class Information
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string License { get; set; }
    public DateTime Established { get; set; }
    public decimal Revenue { get; set; }
}
```

□ Tạo Context classes

```
public DbSet<Information> Information { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=Thepv;Database=Company;Trusted_Connection=
    }
}
```

❑ Tạo và chạy Migration

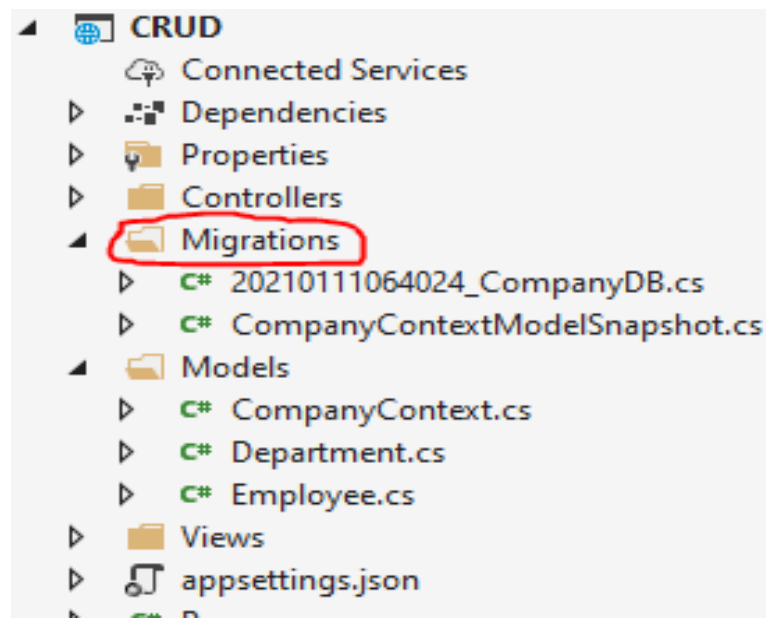
❖ Dùng PMC:

```
PM> add-migration CompanyDB
```

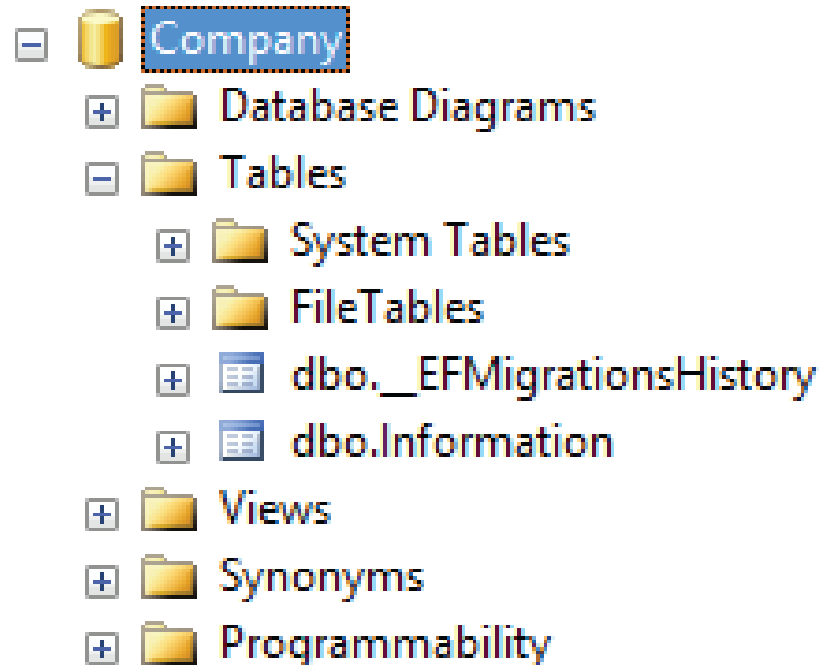
❖ Dùng CLI:

```
PM> dotnet ef migrations add CompanyDB
```

❑ Thư mục migration được tạo




- ❑ Chạy lệnh update-database tạo database: update-database –verbose



❑ Thêm record vào table

```
public string Index()
{
    using (var context = new CompanyContext())
    {
        var info = new Information()
        {
            Name = "Phan Viet The",
            License = "ThePV",
            Revenue = 1000,
            Establishied = Convert.ToDateTime("2021/01/10")
        };
        context.Information.Add(info);
        context.SaveChanges();
    }

    return "Record Inserted";
}
```



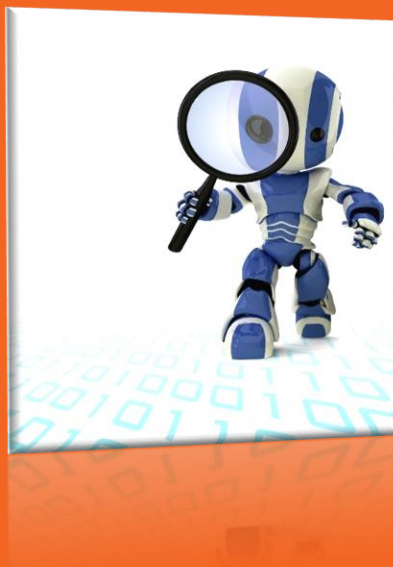
Id	Establishied	License	Name	Revenue
1	2021-01-10	ThePV	Phan Viet The	1000.00
-----				-----



DEMO

- Hiện thực các ví dụ

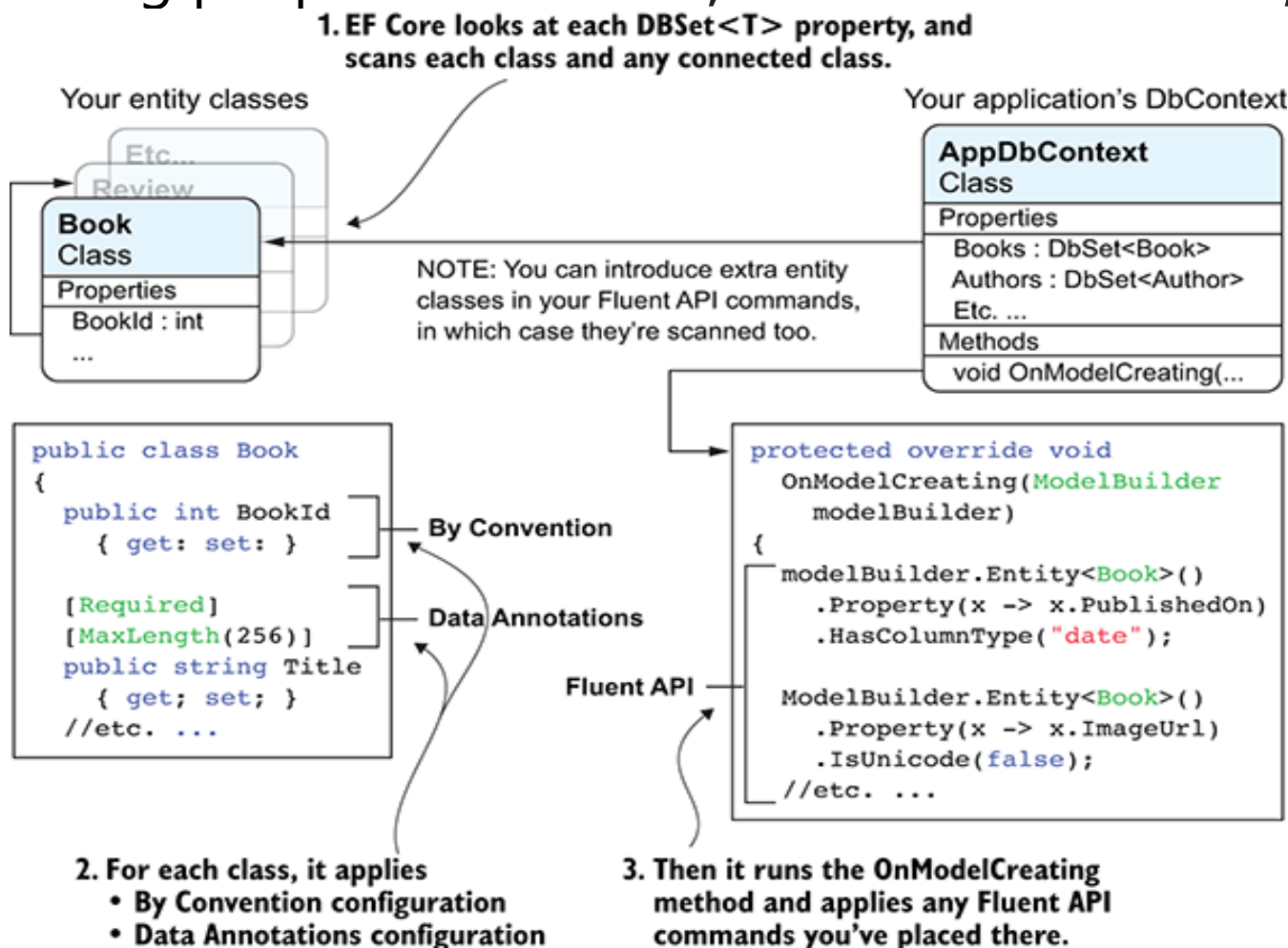




LẬP TRÌNH C# 5

BÀI 1: ENTITY FRAMEWORK CORE – CODE FIRST (P2)

- ❑ Entity Framework code-first có khả năng tự động tạo cơ sở dữ liệu và bảng dữ liệu từ class.
- ❑ Các phương pháp: Convention, Data Annotations, Fluent



1. Các bộ quy tắc mặc định tự động cấu hình, ánh xạ giữa các lớp .NET và cơ sở dữ liệu (Table Name, Schema, Column, Data types, Nullability...) với cách tiếp cận Code First.
2. Entity Framework sẽ ánh xạ mỗi domain class sang một bảng CSDL nếu trong lớp context có property DbSet<> tương ứng.
3. Mặc định luôn tạo bảng dữ liệu trong schema dbo
4. Tên property của class được sử dụng làm tên cột của bảng CSDL
5. Thứ tự của cột trong bảng CSDL giống như thứ tự của property đó trong class.
6. Entity Framework quy ước tên khóa chính của bảng có thể ở hai dạng: Id, và <Tên class>Id
7. Nếu property thuộc kiểu reference hoặc nullable, cột tương ứng của nó được đánh dấu NULL (cho phép để trống không chứa giá trị). Nếu property thuộc kiểu value, nó sẽ ánh xạ sang cột NotNull (bắt buộc phải có giá trị).

8. Mặc định Entity Framework sẽ thực hiện ánh xạ kiểu của C# sang kiểu của Sql Server theo bảng dưới đây

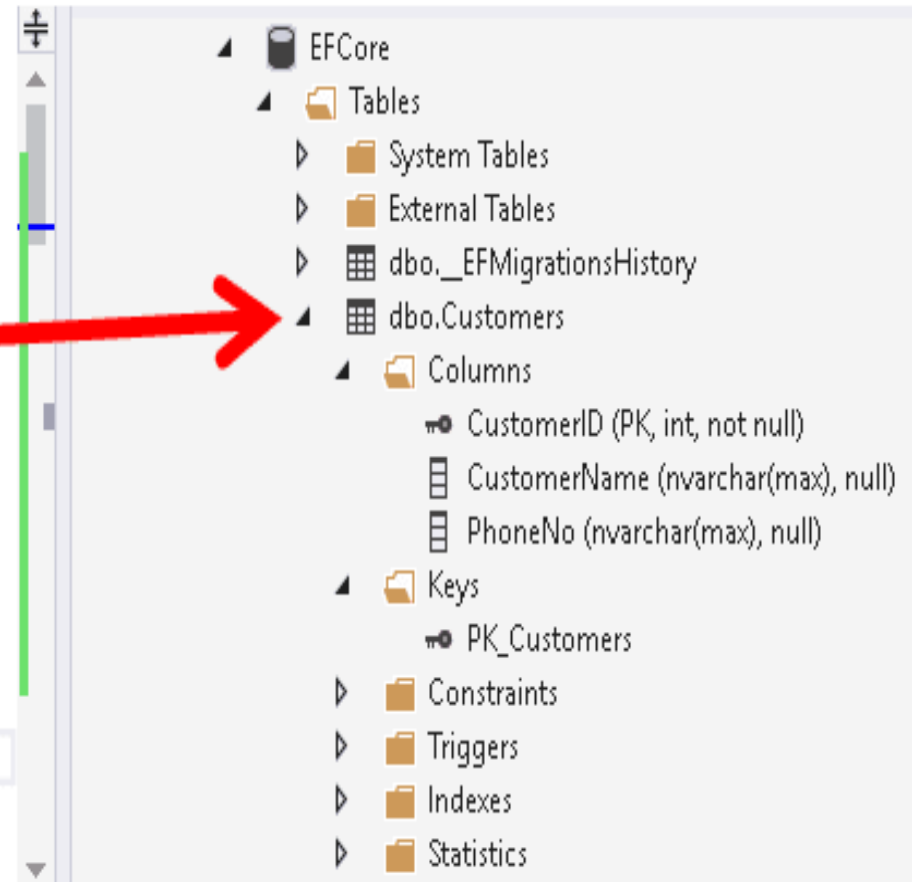
Data type	Mapped	Null ?
□ string	nvarchar(max)	Null
decimal	decimal(18, 2)	Not Null
decimal?	decimal(18, 2)	Null
double	float	Not Null
double?	float	Null
int	int	Not Null
int?	int	Null
bool	bit	Not Null
bool?	bit	Null
DateTime	datetime	Not Null
DateTime?	datetime	Null
byte[]	varbinary(max)	Null

❑ Dùng code first demo giải thích 8 rules của slide trước

```
using System;  
using System.Collections.Generic;  
using System.Text;
```

```
namespace EFCoreConventions.Models
```

```
{  
    public class Customer  
    {  
        public int CustomerID { get; set; }  
        public string CustomerName { get; set; }  
        public string PhoneNo { get; set; }  
    }  
}
```



```
public DbSet<Customer> Customers { get; set; } #5
```

9. Để tạo ra các bảng có quan hệ Foreign Key, khi xây dựng các entity class cần chỉ ra các mối quan hệ này (một – nhiều, nhiều – nhiều, một – một)
10. Quan hệ một – nhiều

The screenshot displays the Visual Studio IDE. On the left, the 'Model.cs' file is open, showing two entity classes: `Employee` and `Department`. The `Employee` class has properties `EmployeeID` (uint) and `EmployeeName` (string). The `Department` class has properties `DepartmentID` (int) and `DepartmentName` (string). A red arrow points from the `Department` property in the `Employee` class to the `DepartmentID` (FK, int, null) column in the `dbo.Employees` table in the SQL Server Object Explorer on the right. The SQL Server Object Explorer shows the database structure, including tables `dbo.Customers`, `dbo.Departments`, and `dbo.Employees`. The `dbo.Employees` table has columns `EmployeeID` (PK, bigint, not null) and `DepartmentID` (FK, int, null).

```
public class Employee
{
    public uint EmployeeID { get; set; }
    public string EmployeeName { get; set; }
    //Reference Navigation Property
    public virtual Department Department { get; set; }
}

public class Department
{
    public int DepartmentID { get; set; }
    public string DepartmentName { get; set; }
}
```

10. Quan hệ một – nhiều có thể thực hiện thông qua Collection Navigation Property

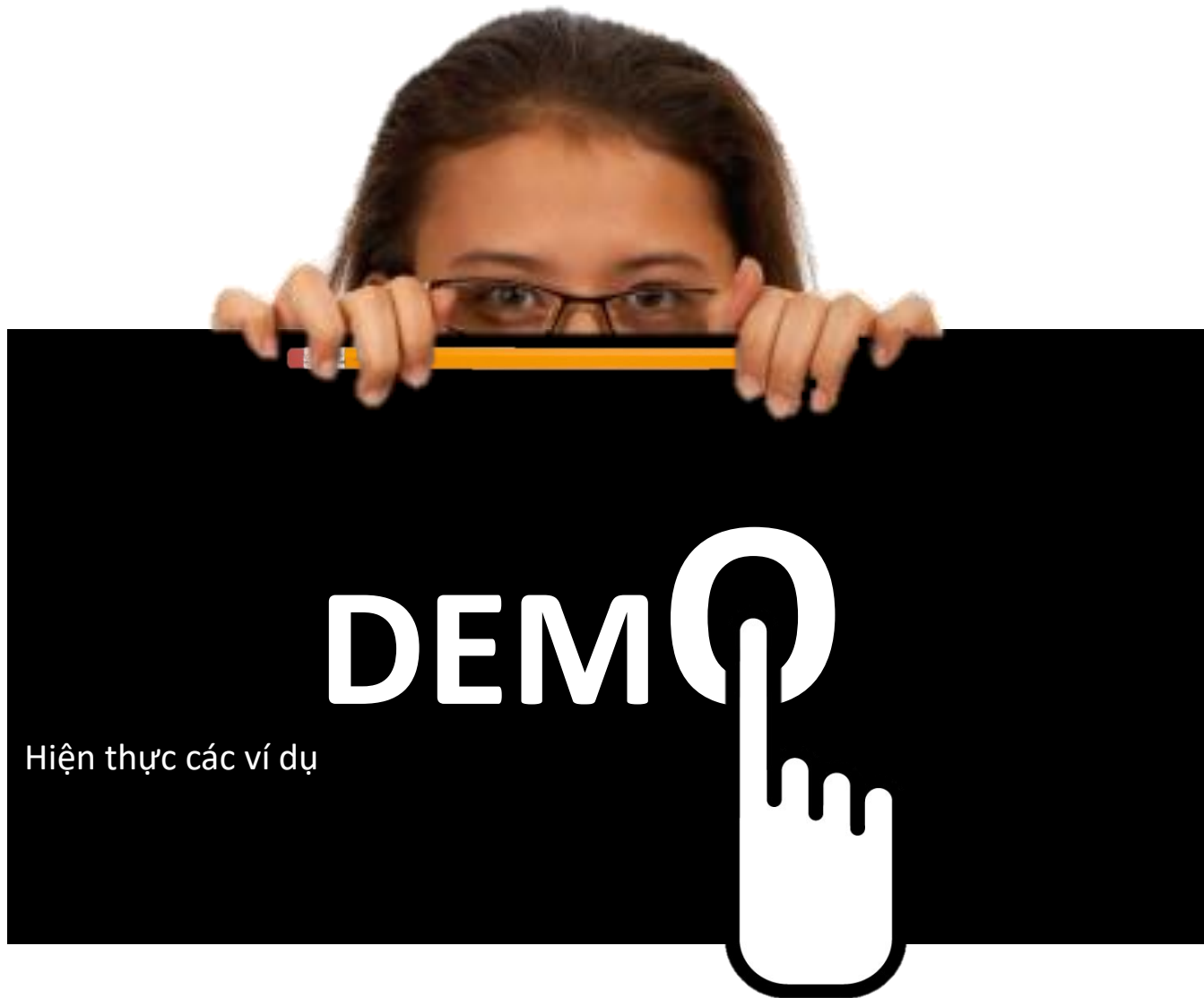
```
public class Employee
{
    0 references
    public uint EmployeeID { get; set; }
    0 references
    public string EmployeeName { get; set; }
}
0 references
public class Department
{
    0 references
    public int DepartmentID { get; set; }
    0 references
    public string DepartmentName { get; set; }
    //Collection Navigation Property
    0 references
    public ICollection<Employee> Employees { get; set; }
}
```


11. Quan hệ một – một

```

public class Employee
{
    0 references
    public uint EmployeeID { get; set; }
    0 references
    public string EmployeeName { get; set; }
    //Reference Navigation Property
    0 references
    public Department Department { get; set; }
}

1 reference
public class Department
{
    0 references
    public int DepartmentID { get; set; }
    0 references
    public string DepartmentName { get; set; }
    //Reference Navigation Property
    0 references
    public Employee Employee { get; set; }
}
    
```



- ❑ Code first kết hợp Razor View Engine tạo ra giao diện tương tác giúp các chức năng crud trực quan
- ❑ Ví dụ code first và razor view các chức năng crud thông tin Product

```
public class Product
{
    0 references
    public int Id { get; set; }
    0 references
    public string Name { get; set; }
    0 references
    public decimal Price { get; set; }
    0 references
    public int Quantity { get; set; }
    0 references
    public bool Status { get; set; }
}
```

```
public class CompanyContext : DbContext
{
    0 references
    public DbSet<Product> Product { get; set; }
}
```

Structure of Product Table


	Column Name	Data Type	Allow Nulls
🔑	Id	int	<input type="checkbox"/>
	Name	varchar(250)	<input checked="" type="checkbox"/>
	Price	money	<input checked="" type="checkbox"/>
	Quantity	int	<input checked="" type="checkbox"/>
	Status	bit	<input checked="" type="checkbox"/>

- ❑ Ví dụ code first và razor view các chức năng crud thông tin Product, một số controller và view tương ứng

Đọc dữ liệu

```
private DataContext db = new DataContext();

[Route("")]
[Route("index")]
[Route("~/")]
public IActionResult Index()
{
    ViewBag.products = db.Product.ToList();
    return View();
}
```



```
<table border="1">
  <tr>
    <th>Id</th>
    <th>Name</th>
    <th>Price</th>
    <th>Quantity</th>
    <th>Status</th>
  </tr>
  @foreach (var product in ViewBag.products)
  {
    <tr>
      <td>@product.Id</td>
      <td>@product.Name</td>
      <td>@product.Price</td>
      <td>@product.Quantity</td>
      <td>@product.Status</td>
    </tr>
  }
</table>
```

Thêm dữ liệu

```
[Route("add")]
[HttpGet]
public IActionResult Add()
{
    return View();
}

[Route("add")]
[HttpPost]
public IActionResult Add(Product product)
{
    db.Product.Add(product);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```



```
<form method="post" asp-controller="product" asp-action="add">
    <table cellpadding="2" cellspacing="2">
        <tr>
            <td>Name</td>
            <td>
                <input type="text" asp-for="Name" />
            </td>
        </tr>
        <tr>
            <td>Price</td>
            <td>
                <input type="text" asp-for="Price" />
            </td>
        </tr>
        <tr>
            <td>Quantity</td>
            <td>
                <input type="text" asp-for="Quantity" />
            </td>
        </tr>
        <tr>
            <td>Status</td>
            <td>
                <input type="checkbox" asp-for="Status" />
            </td>
        </tr>
        <tr>
            <td>&nbsp;</td>
            <td>
                <input type="submit" value="Save" />
            </td>
        </tr>
    </table>
</form>
```

Cập nhật dữ liệu

```
[Route("edit")]
[HttpGet]
public IActionResult Edit()
{
    return View("Edit");
}


[Route("edit")]
[HttpPost]
public IActionResult Edit(Product product)
{
    db.Entry(product).State = EntityState.Modified;
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

```
<form method="post" asp-controller="product" asp-action="edit">
  <table cellpadding="2" cellspacing="2">
    <tr>
      <td>Id</td>
      <td>
        <input type="text" asp-for="Id" />
      </td>
    </tr>
    <tr>
      <td>Name</td>
      <td>
        <input type="text" asp-for="Name" />
      </td>
    </tr>
    <tr>
      <td>Price</td>
      <td>
        <input type="text" asp-for="Price" />
      </td>
    </tr>
    <tr>
      <td>Quantity</td>
      <td>
        <input type="text" asp-for="Quantity" />
      </td>
    </tr>
    <tr>
      <td>Status</td>
      <td>
        <input type="checkbox" asp-for="Status" />
      </td>
    </tr>
    <tr>
      <td>&nbsp;</td>
      <td>
        <input type="submit" value="Save" />
      </td>
    </tr>
  </table>
</form>
```

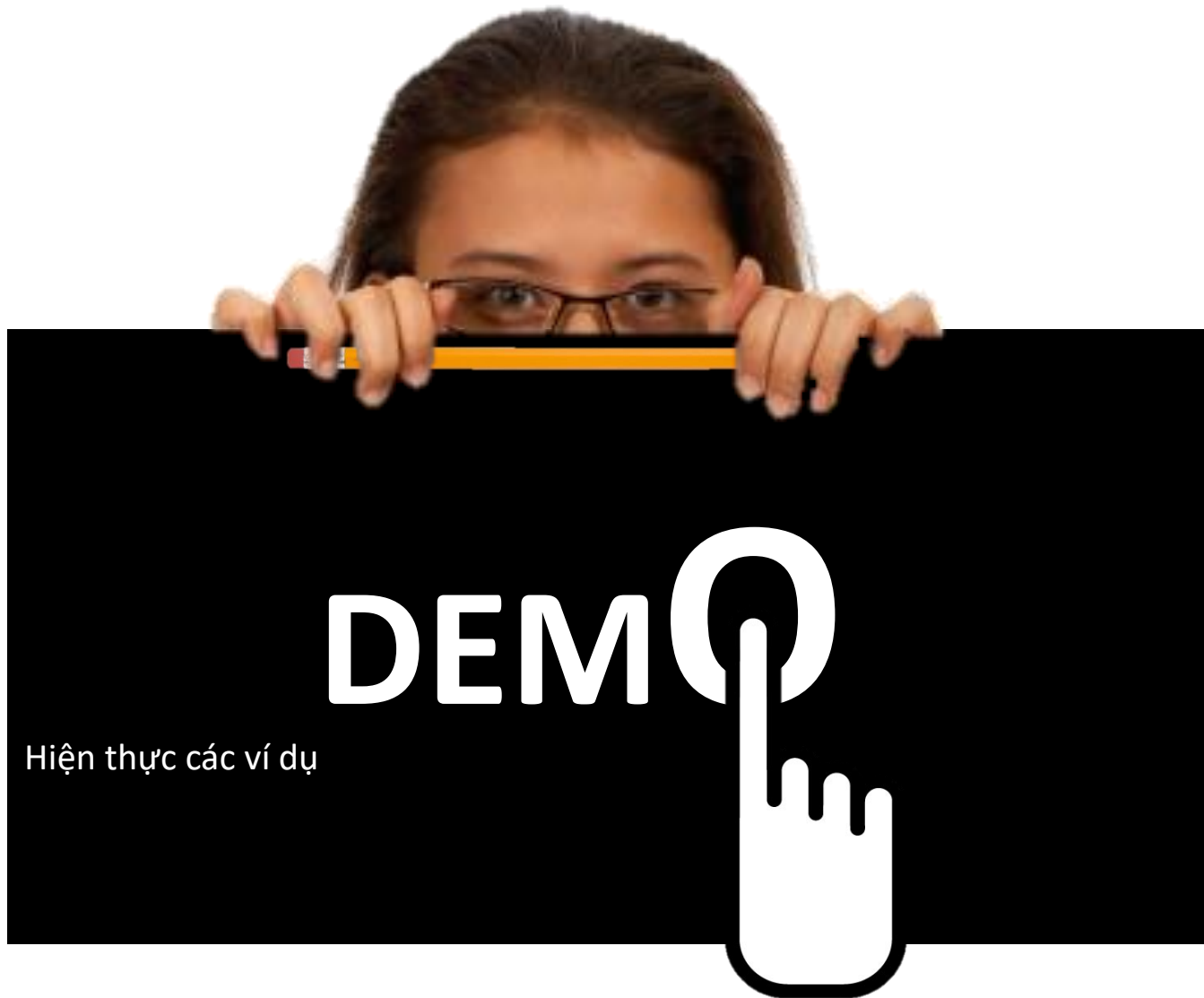
Xóa dữ liệu

```
[Route("delete")]
[HttpGet]
public IActionResult Delete()
{
    return View("Delete");
}

[Route("delete")]
[HttpPost]
public IActionResult Delete(int id)
{
    db.Product.Remove(db.Product.Find(id));
    db.SaveChanges();
    return RedirectToAction("Index");
}
```



```
<form method="post" asp-controller="product" asp-action="delete">
    <table cellpadding="2" cellspacing="2">
        <tr>
            <td>Id</td>
            <td>
                <input type="text" asp-for="Id" />
            </td>
        </tr>
        <tr>
            <td>&nbsp;</td>
            <td>
                <input type="submit" value="Delete" />
            </td>
        </tr>
    </table>
</form>
```



Tổng kết bài học

- ◎Entity Framework Core – Code First
- ◎Conventions Rules
- ◎Code first và razor view





KẾT THÚC