



# CHỦ ĐỀ

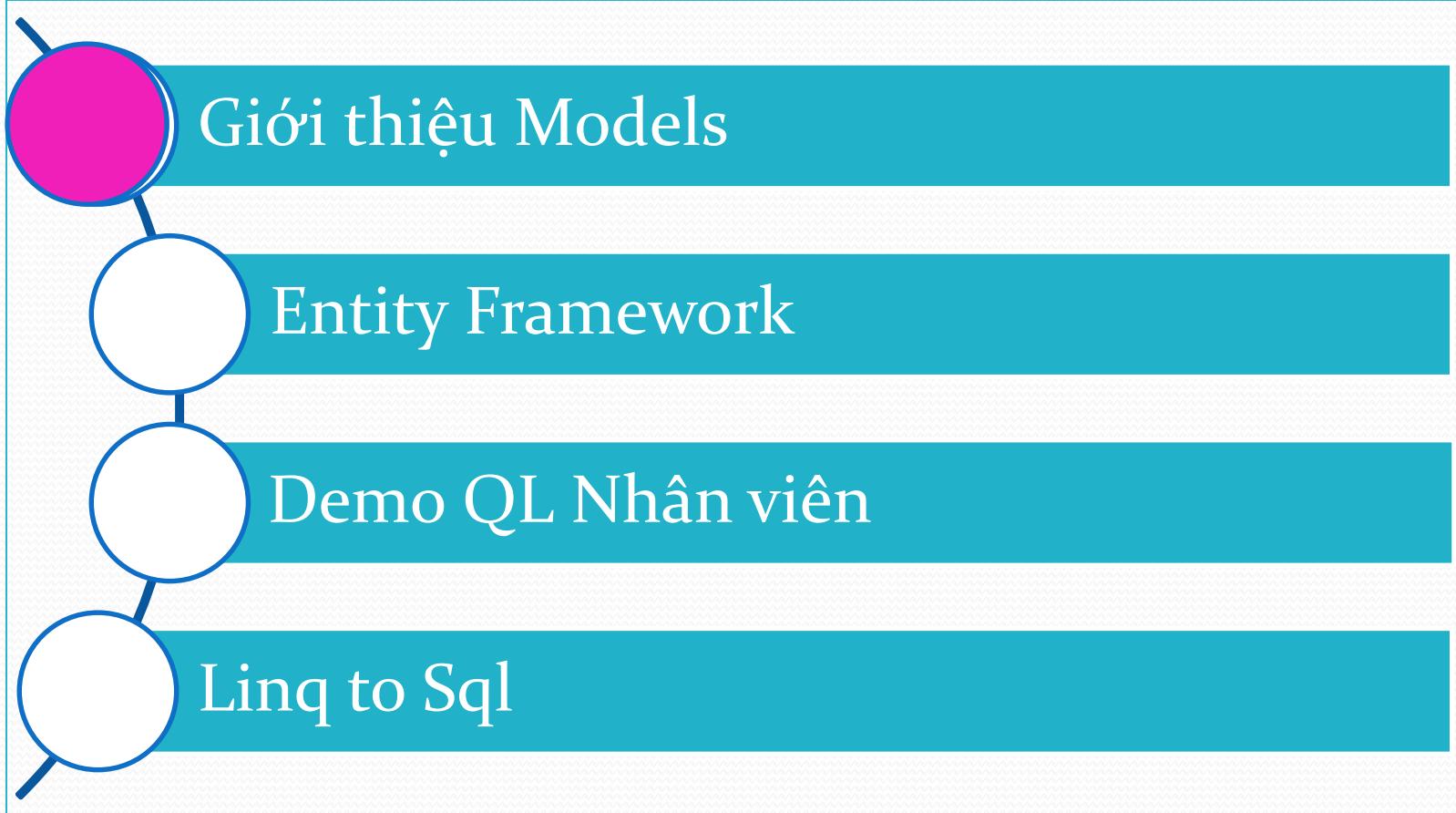


# MODELS

GVGD: PHẠM THỊ KIM NGOAN

Email: ngoanptk@ntu.edu.vn

# Nội dung

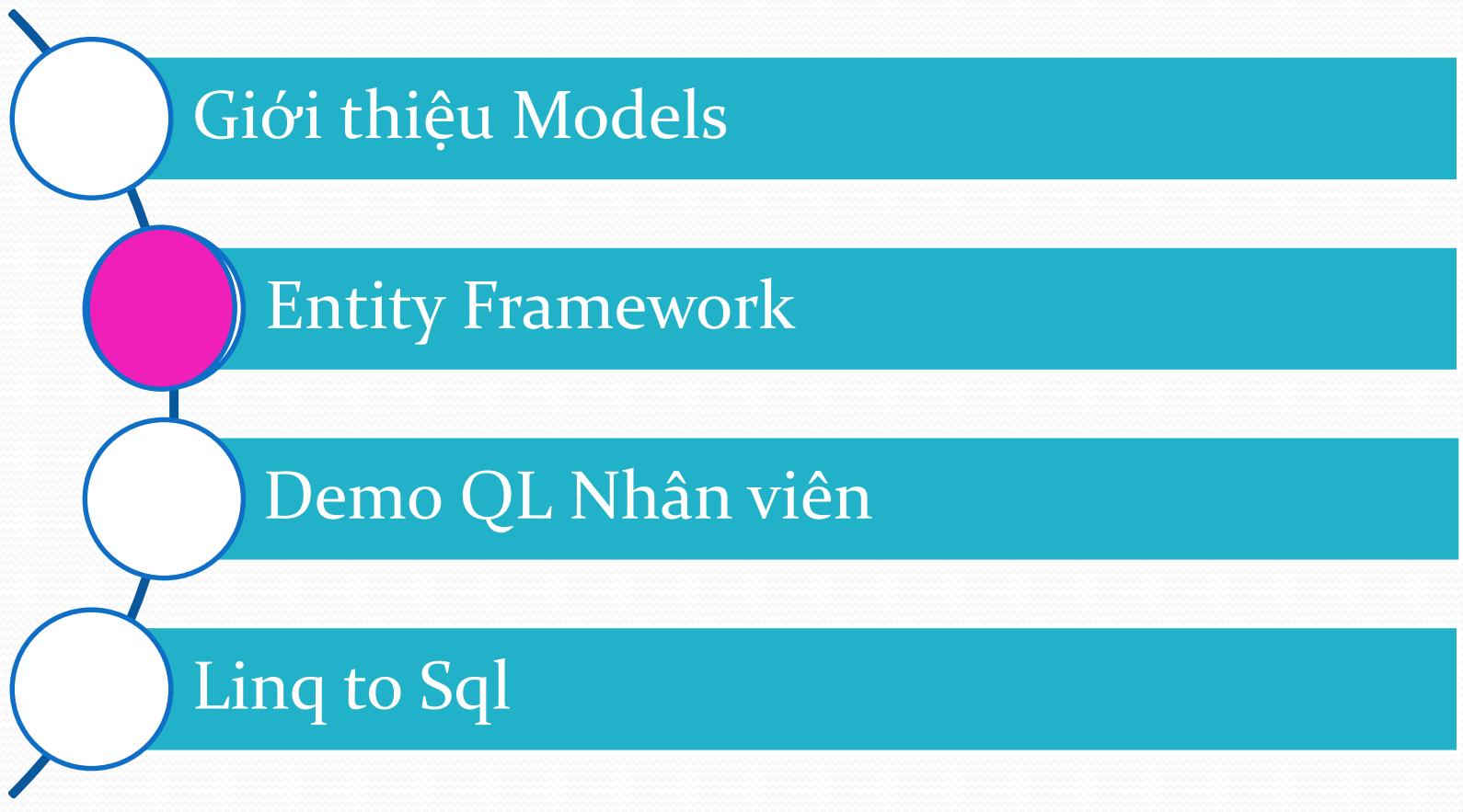


# 1. Giới thiệu Models

- Được dùng định nghĩa dữ liệu được sử dụng cho Controller và View.
- Mỗi Model là một class có phần mở rộng `.cs` trong thư mục Model.
- Cách sử dụng Model trong Controller: Thêm không gian tên trong Controller cần sử dụng Model đã định nghĩa.

```
using System.Web.Mvc; //khong gian ten MVC
using SampleMVC.Models; //them khong gian ten dinh nghia Model
```

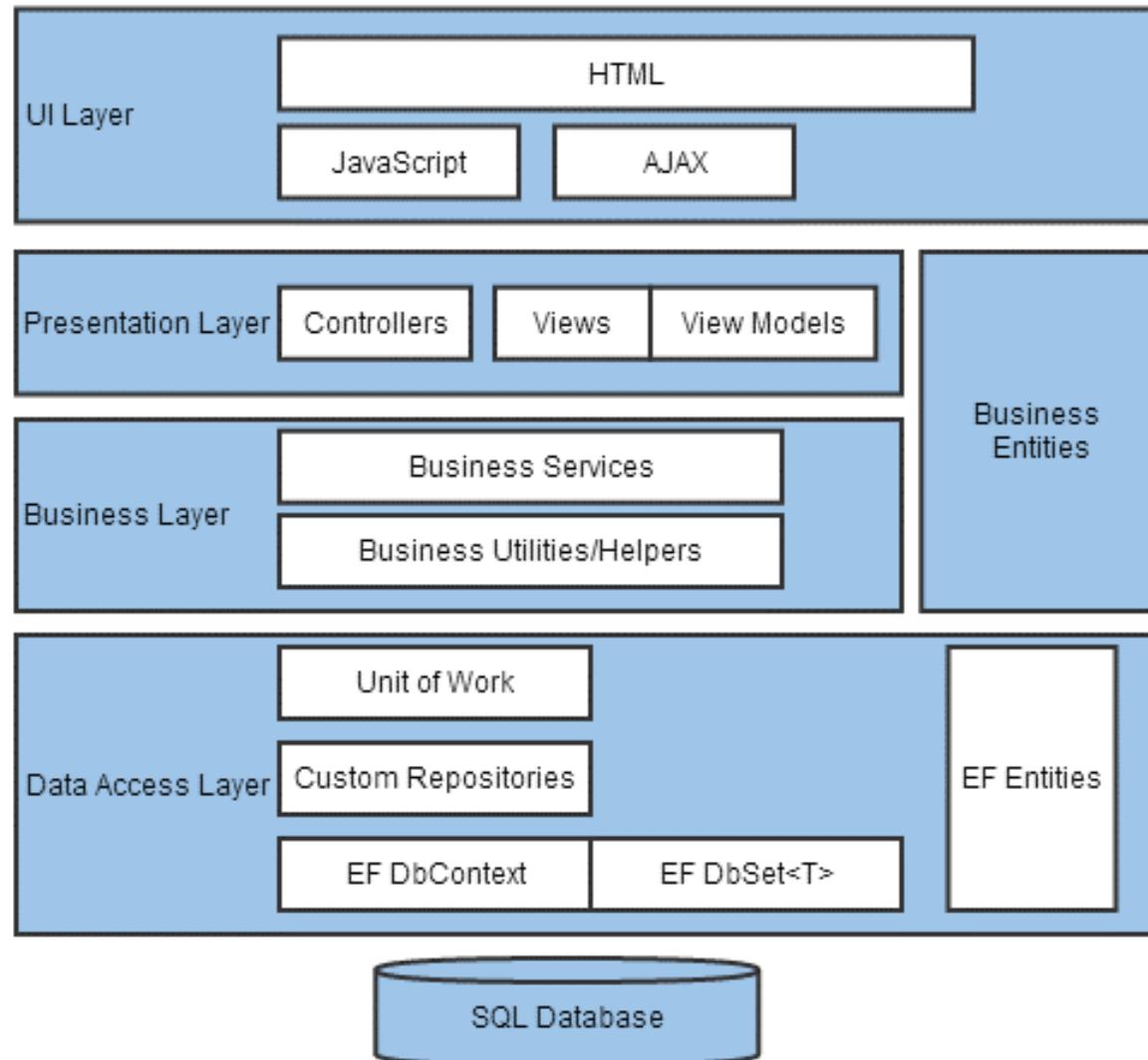
# Nội dung



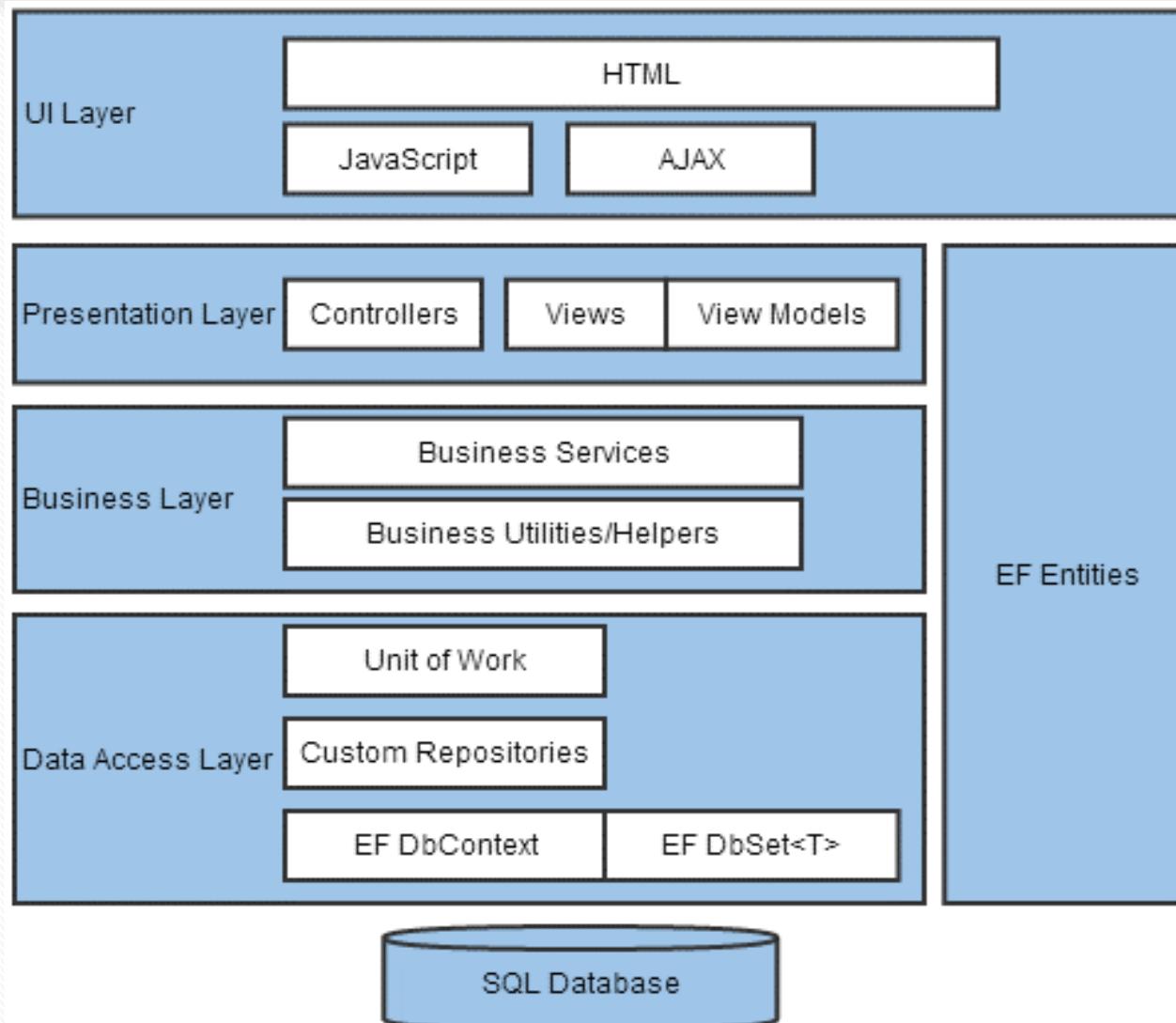
## 2. Giới thiệu Entity Framework

- Microsoft ADO.NET Entity Framework là một nền tảng sử dụng làm việc với cơ sở dữ liệu quan hệ thông qua cơ chế ánh xạ đối tượng Object / Relational Mapping (ORM).
- Entity Framework cho phép các nhà phát triển làm việc với cơ sở dữ liệu quan hệ trên mô hình các đối tượng **không** phải làm việc trực tiếp trên các bảng trong cơ sở dữ liệu, do đó người phát triển **không** phải viết các đoạn mã truy cập cơ sở dữ liệu thường cần phải viết.
- Entity framework là một nền tảng nâng cao của ADO.NET vì vậy Entity framework cung cấp cho các nhà phát triển ứng dụng một **cơ chế tự động** cho việc truy cập và lưu trữ dữ liệu trong cơ sở dữ liệu.

# Entity trong mô hình lập trình web (LT)

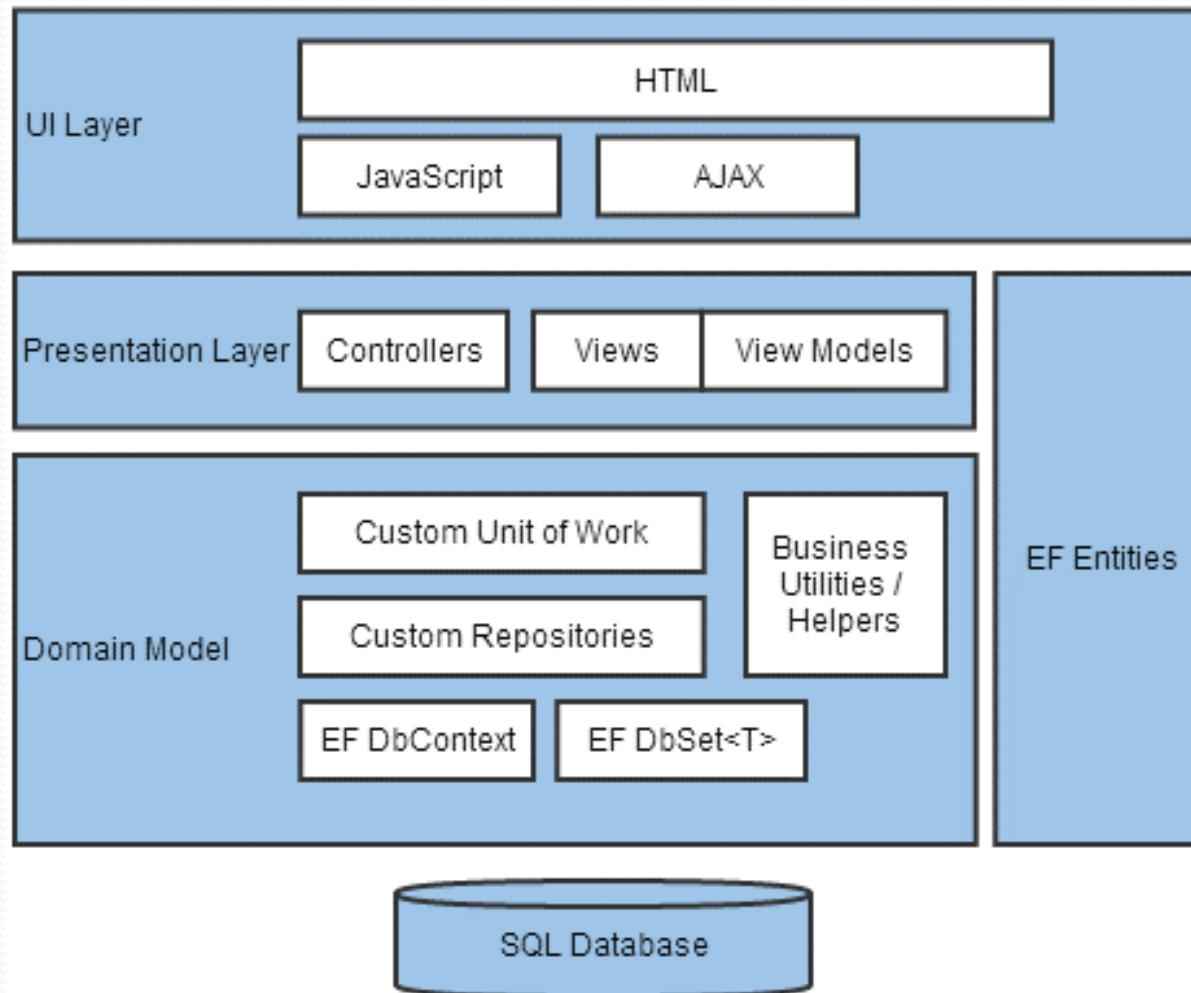


# Entity trong mô hình lập trình web (Thực tế)



# Entity trong mô hình lập trình web (Thực tế)

- Có thể gom tầng Business và Data Access thành Domain Model

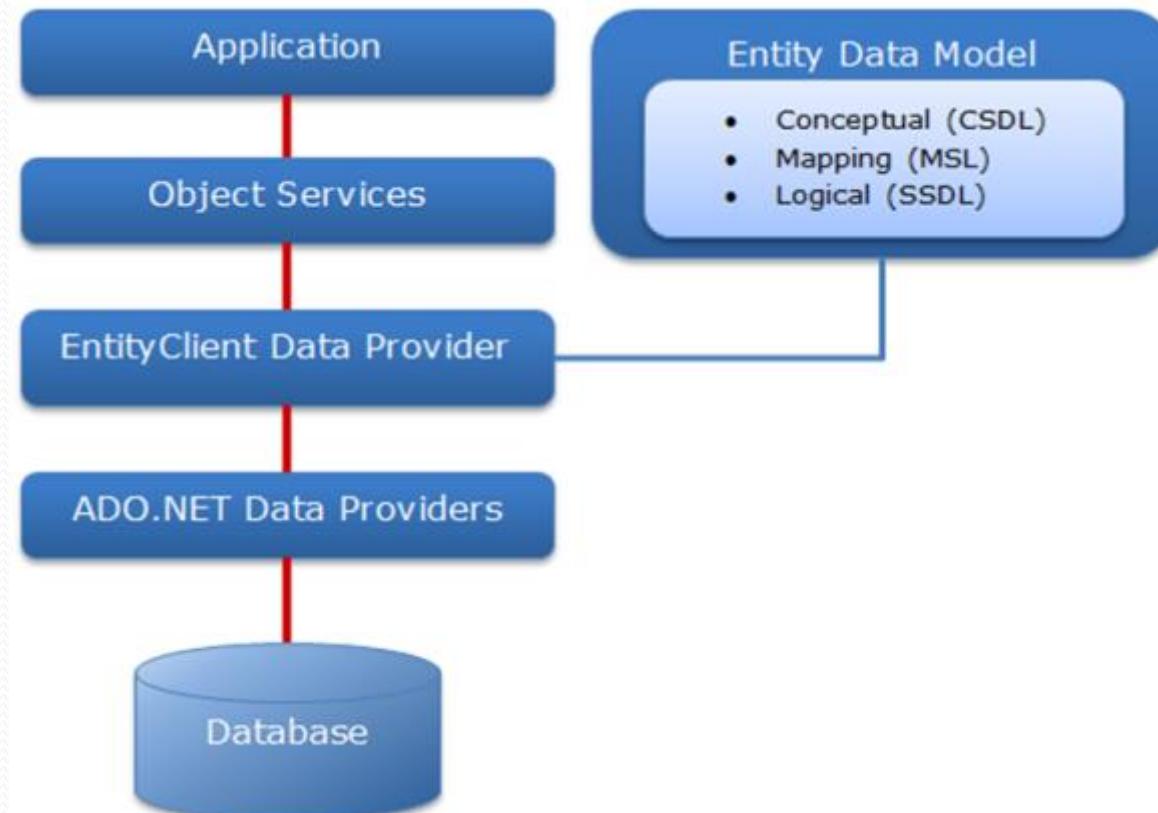


# Giới thiệu Entity Framework

- EF có vị trí trung gian, đóng vai trò kết nối giữa cơ sở dữ liệu và các thành phần khác của 1 dự án Web khi cần đến.
- Cài đặt: EF có sẵn tại gói EntityFramework tại Nuget

<http://nuget.org/packages/EntityFramework/>

# Kiến trúc Entity Framework



# Kiến trúc Entity Framework

- **Application** (ứng dụng) chứa giao diện trang Web (HTML, CSS, Javascript, hình ảnh, ...) và các đoạn mã nguồn (C#, VB) để tương tác dữ liệu với các tầng khác trong mô hình thông qua Object Services.
- **Object Services** (các dịch vụ đối tượng) chứa quá trình tương tác giữa ứng dụng và database.
- **EntityClient Data Provider** cung cấp các kết nối, diễn dịch các truy vấn thực thể thành truy vấn nguồn dữ liệu, trả về data reader để EF dùng chuyển dữ liệu thực thể thành các đối tượng. Phần này kết nối ADO.NET Data Providers để gửi hoặc lấy dữ liệu từ database.

# Kiến trúc Entity Framework

- **ADO.NET Data Providers** kết nối với database sử dụng ADO.NET.
- **EDM (Entity Data Model)** mô hình dữ liệu thực thể) chứa 3 phần chính: mô hình khái niệm (CSDL – Conceptual schema definition language), mô hình ánh xạ (MSL – mapping specification language) và mô hình lưu trữ (SSDL – store schema definition language). EDM khác với EntityClient Data Provider ở chỗ EDM sử dụng LINQ là ngôn ngữ truy vấn tương tác với database.

# Kiến trúc Entity Framework

- **Mô hình khái niệm** (CSDL – Conceptual schema definition language): Mô hình khái niệm chứa **các lớp mô hình** và **mối quan hệ giữa các lớp** này. Điều này để độc lập với mô hình quan hệ các bảng trong database.
- **Mô hình lưu trữ** (SSDL – store schema definition language): Mô hình lưu trữ là 1 mô hình thiết kế **database** bao gồm các **bảng, view, stored procedure** (thủ tục), và **mối quan hệ** giữa chúng và các khóa. Mô hình này thể hiện gần giống mô hình quan hệ các bảng trong database.
- **Mô hình ánh xạ** (MSL – mapping specification language): Mô hình ánh xạ gồm thông tin về **cách mô hình khái niệm** được **ánh xạ** đến **mô hình lưu trữ**.

# Kiến trúc Entity Framework

- **L2E (LINQ to Entities):** Là 1 ngôn ngữ truy vấn được dùng để viết các truy vấn tới object model và trả về các thực thể được định nghĩa trong Conceptual Model.

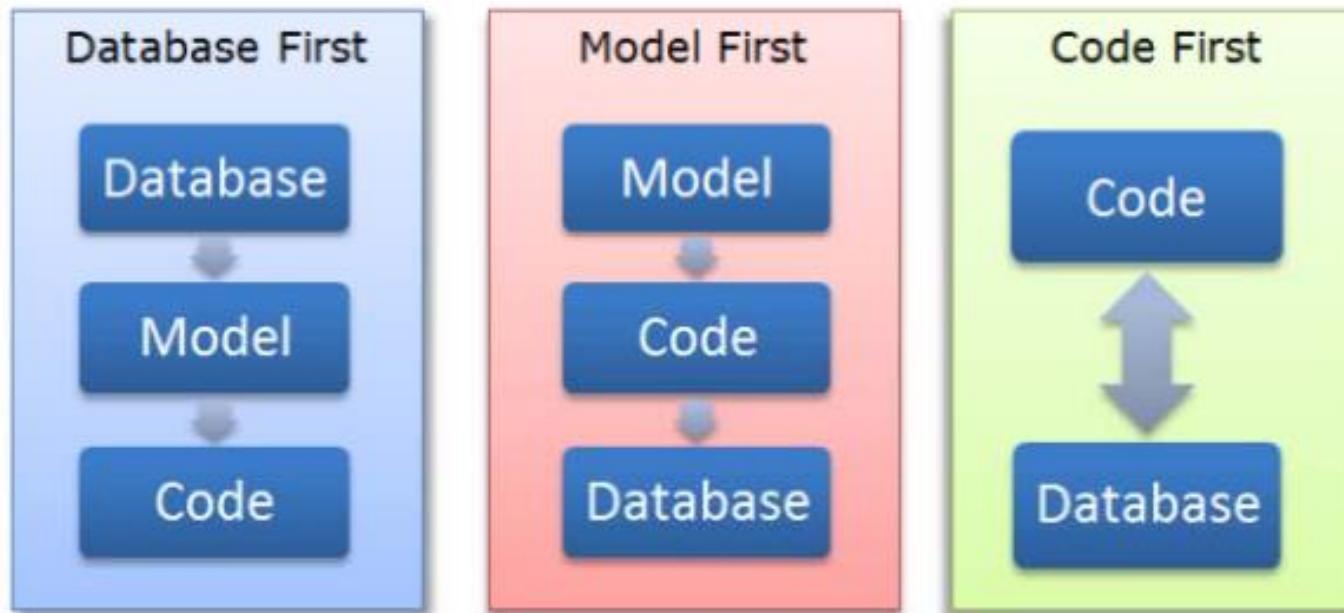
# Entity Framework (EF)

➤ Khi sử dụng EF có 3 lựa chọn:

- **Database First:** thiết kế CSDL (database) trước, sau đó dùng VS tạo lớp entity. Kỹ thuật này giống với kỹ thuật tạo lớp Entity dùng Linq to Sql.
- **Model First:** Dùng VS tạo lớp Entity, sau đó từ Entity sinh ra database.
- **Code first:** Tạo các lớp ứng dụng, lớp Entity như các đối tượng trong C# thông thường. Khi ứng dụng chạy sẽ tự động tạo ra 1 database tương ứng với các lớp.

# Entity Framework (EF)

- Khi sử dụng EF có 3 lựa chọn:



# Database First

---

- Tạo 1 database
- Tạo ứng dụng
- Xây dựng mô hình
- Đọc và ghi dữ liệu
- Thay đổi mô hình

# Database First

## ➤ Tạo 1 database

- Mở SQL Server
- Tạo các Tables
- Hoặc có thể tạo Database trong Visual Studio: *View* → *Server Explorer*. R\_Click vào *Database Connections* → *Add Connections...*

## ➤ Tạo ứng dụng

- Mở Visual Studio
- Tạo ứng dụng Web Asp.net (.Net Framework) dùng MVC.

# Database First

➤ **Xây dựng mô hình:** Tạo mô hình trong thư mục Models

- R\_Click vào thư mục Models → Add → New item → Data → ADO.NET Entity Model.
- Chọn *EF Designer From Database*.
- Kết nối database với project (Sever name của Sql Sever và Database name)
- Chọn các bảng cần generate ra model.
- Các Models tương ứng với các Tables sẽ được tạo ra tự động trong Thư mục Models.

# Database First

---

## ➤ Đọc và ghi dữ liệu:

- Minh họa trong phần Demo.

# Database First

## ➤ Thay đổi mô hình:

- Trong database: thêm 1 bảng mới, ..
- Cập nhật mô hình: mở file *EDMX*, R\_Click chọn *Update Model from Database...* để bật cửa sổ *Update Wizard*.
- Ở tab *Add* của *Update Wizard* chọn *Tables*, chọn bảng mới cần gieo mã nguồn.
- Tab *Refresh* để làm tươi các bảng trong mô hình.
- Tab *Delete* để xóa bảng nào trong mô hình.

# Model First

---

- Tạo ứng dụng
- Tạo mô hình
- Gieo cơ sở dữ liệu
- Đọc và ghi dữ liệu
- Thay đổi mô hình

# Model First

---

## ➤ Tạo ứng dụng

- Mở Visual Studio, chọn *File* → *New* → *Project...*

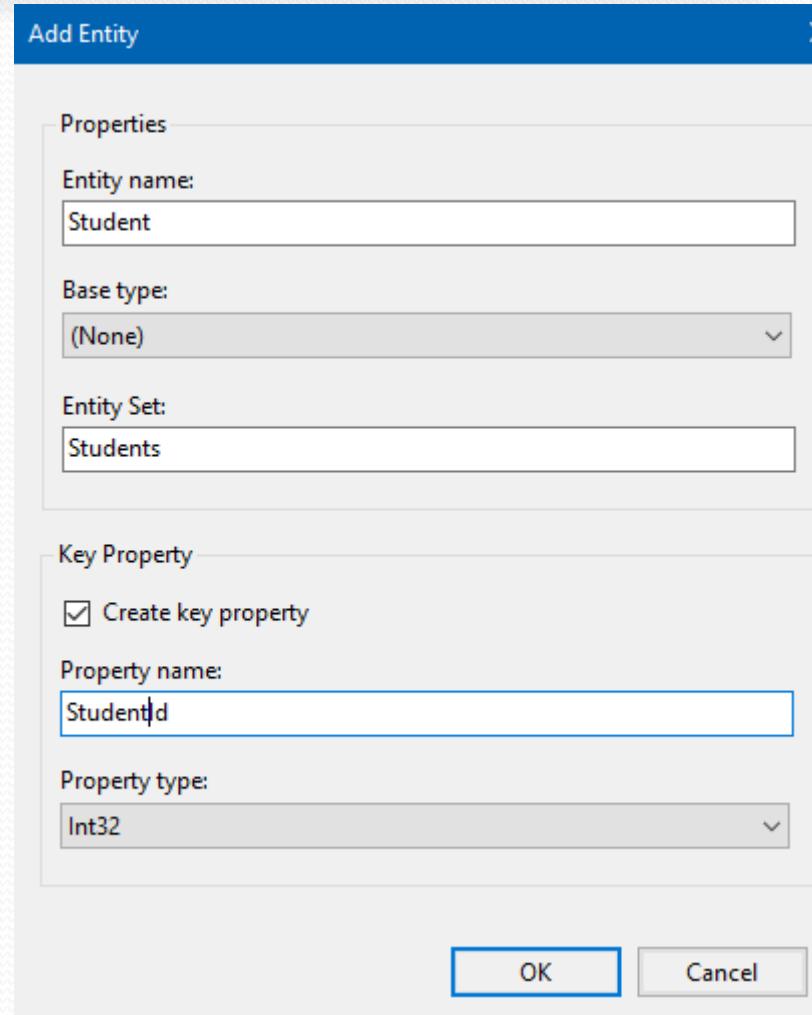
## ➤ Tạo mô hình

- R\_Click vào thư mục *Models* → *Add* → *New item* → *Data* → *ADO.NET Entity Model*.
- Chọn *Empty EF Designer Model*.
- *NameModel.edmx* được tạo ra với 1 mô hình trống.
- R\_Click vào vùng trống trong *NameModel.edmx*, chọn *Properties*

# Model First

## ➤ Tạo mô hình

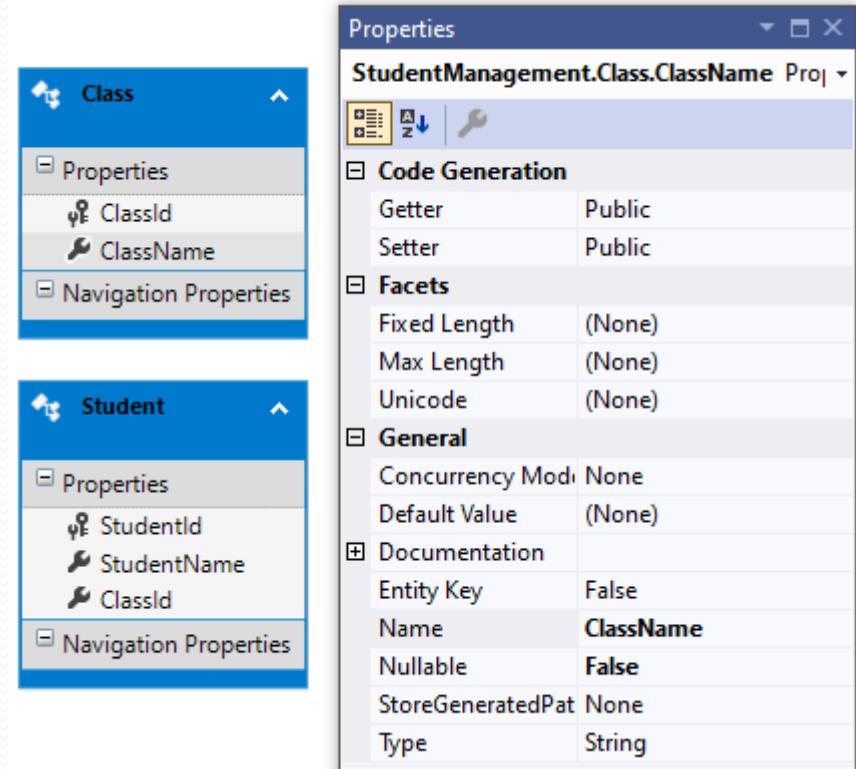
- R\_Click vào vùng trống trong NameModel.edmx, chọn Properties.
- Tạo thực thể: R\_Click lên vùng trống, chọn Add New → Entity...



# Model First

## ➤ Tạo mô hình

- R\_Click vào vùng trống trong *NameModel.edmx*, chọn *Properties*.
- Tạo thực thể: R\_Click lên vùng trống, chọn *Add New → Entity...*
- Tạo thêm thuộc tính cho thực thể: R\_Click thực thể này, chọn *Add New → Scalar Property*, điền *Name* là tên thuộc tính.



# Model First

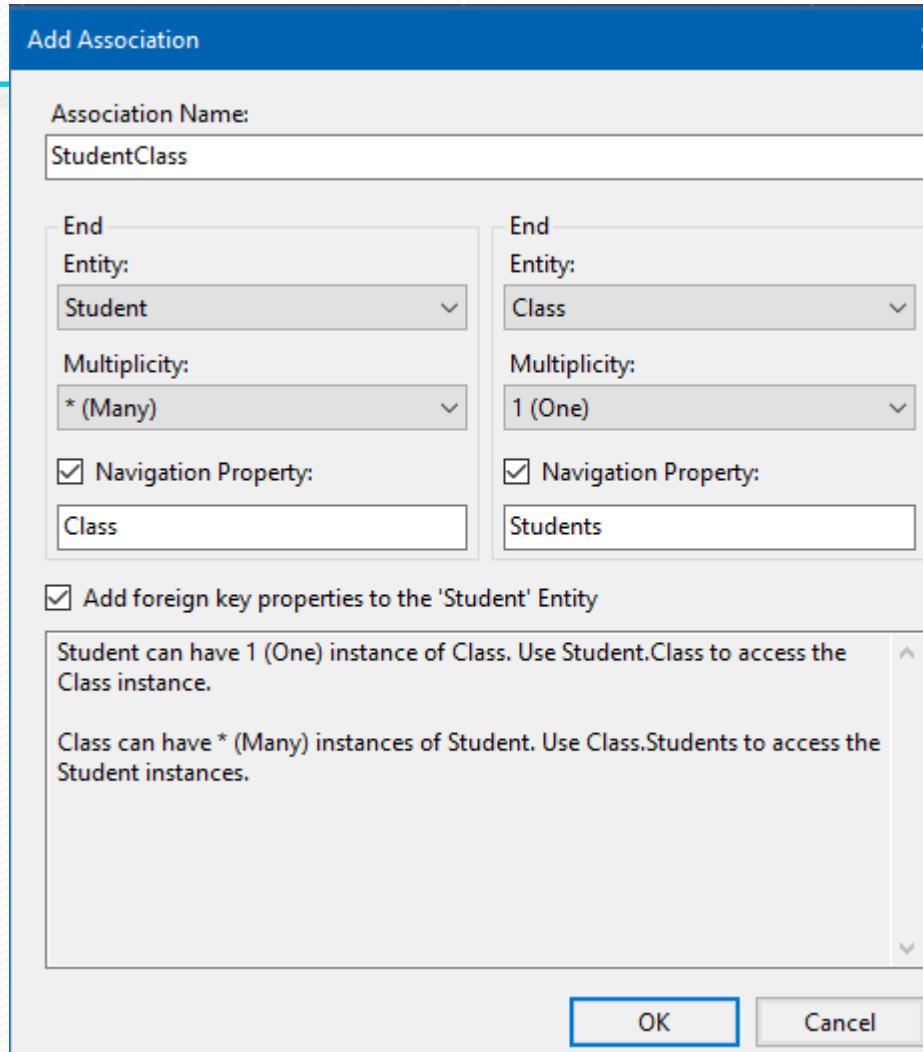
➤ **Tạo mô hình:** Trong cửa sổ Properties có các mục chính sau:

- **Default Value:** là giá trị mặc định của thuộc tính, đặt (None) nghĩa là không có giá trị mặc định
- **Entity Key:** đặt khóa của thực thể (True/False)
- **Fixed Length:** chiều dài cố định của thuộc tính (True/False)
- **Getter/Setter:** phạm vi truy cập của 2 thuộc tính Getter/Setter (Public, Protected, Private, Internal)
- **Max Length:** chiều dài tối đa
- **Name:** tên thuộc tính
- **Nullable:** cho phép thuộc tính có dữ liệu null hay không (None/True/False)
- **Type:** kiểu dữ liệu của thuộc tính

# Model First

## ➤ Tạo mô hình

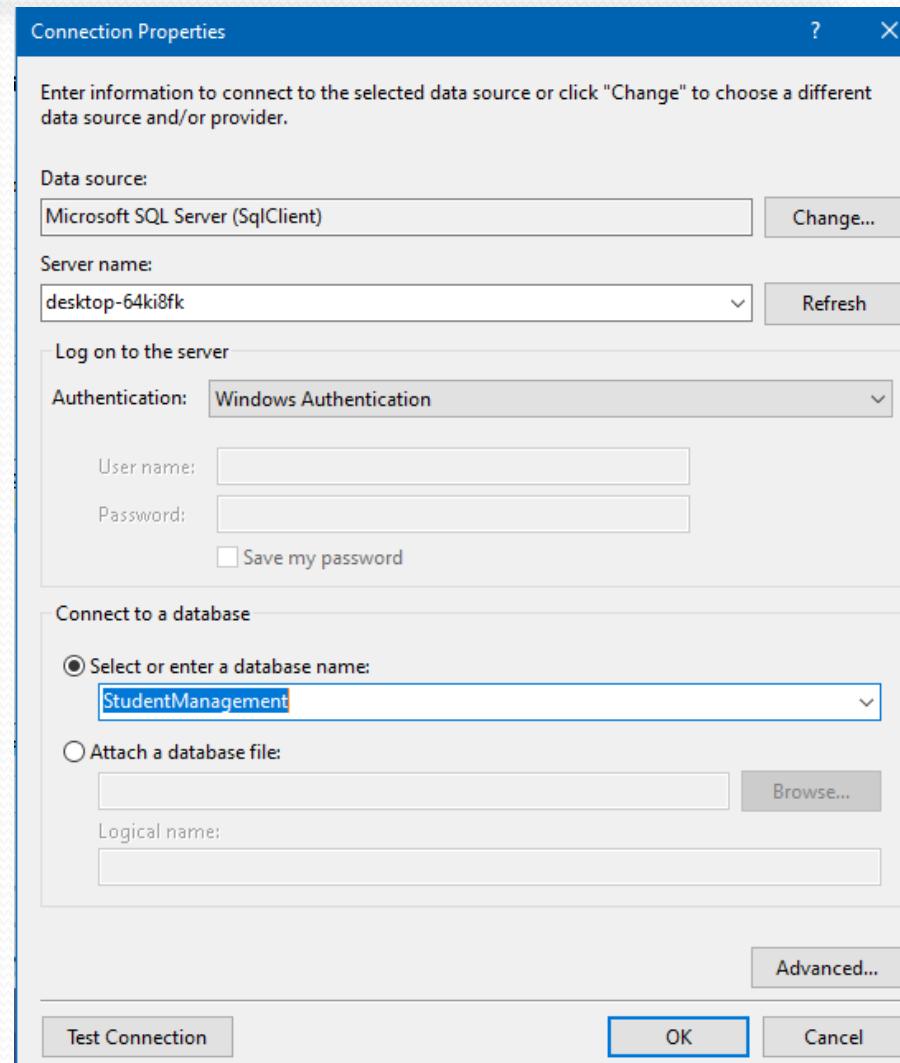
- Tạo mới quan hệ: R\_Click vùng trống giao diện, chọn Add New → Association...



# Model First

## ➤ Gieo cơ sở dữ liệu

- R\_Click lên vùng trống, chọn *Generate Database from Model* → *New Connection...*
- Chọn *Next, ..*



# Model First

---

## ➤ Đọc và ghi dữ liệu:

- Minh họa trong phần Demo.

# Model First

➤ **Thay đổi mô hình:** Khi mô hình thay đổi cần phải cập nhật lược đồ database và cũng như gieo lại các lớp code.

- Có thể thêm mới các Entity, ..
- Tạo lại lược đồ: R\_Click vùng trống ở giao diện tập tin EDMX, chọn *Generate Database from Model...*, Entity Framework sẽ gieo 1 đoạn script để tạo lại lược đồ dựa trên mô hình được cập nhật. Sau đó chọn *Finish*.
- Chấn *Execute (Ctrl + Shift + E)* để gieo mô hình.

# Code First

---

- Với cách tiếp cận Code-first, Entity Framework sẽ tạo các đối tượng bảng cơ sở dữ liệu dựa trên model trong ứng dụng.
- Một vài quy ước của code-first:
  - Quy ước tên bảng: Một class mô tả các thực thể là User sẽ được lưu trữ trong database thì Entity Framework mặc định sẽ tạo ra bảng có tên Users.
  - Quy ước khóa chính: Tạo thuộc tính có tên UserId trong lớp User của model, thuộc tính này được nhận là khóa chính.
  - Quy ước về mối quan hệ: Entity Framework cung cấp các quy ước khác nhau để nhận biết một mối quan hệ giữa 2 models dựa vào tên của thuộc tính và kiểu dữ liệu.

# Code First

---

- Tạo mô hình
- Tạo cơ sở dữ liệu trong SQL Server
- Tạo bối cảnh (context)
- Đọc & Ghi dữ liệu
- Thay đổi mô hình
- Chú thích dữ liệu (Data Annotations)

# Code First

- **Tạo mô hình:** Tạo các class tương ứng với các table trong database.

```
public class Product
{
    public int ProductID { get; set; }

    public string NameOfProduct { get; set; }

    public int CategoryID { get; set; }
}
```

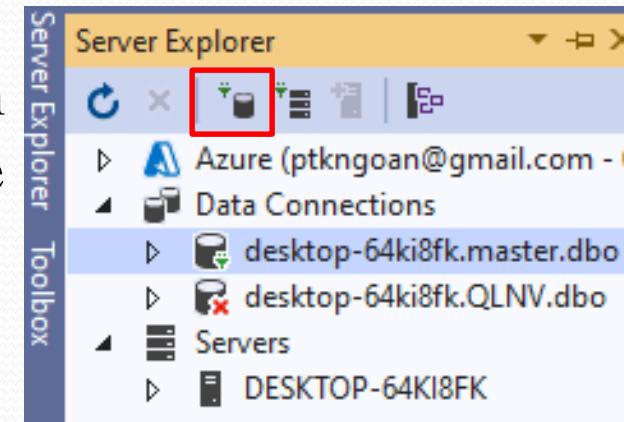
```
public class Category
{
    public int CategoryID { get; set; }

    public string Description { get; set; }
}
```

# Code First

## ➤ Tạo cơ sở dữ liệu

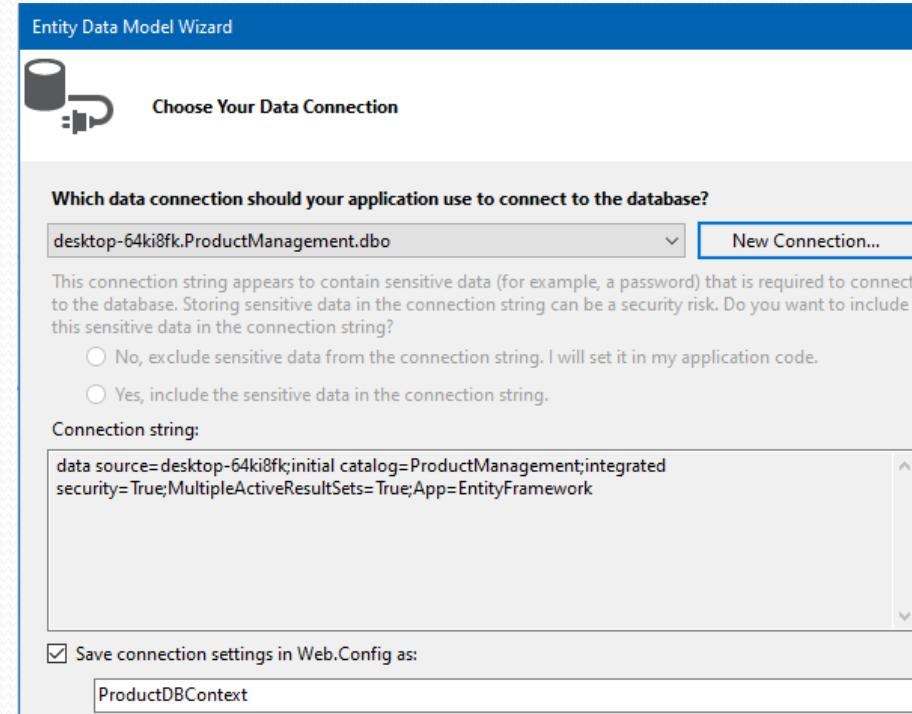
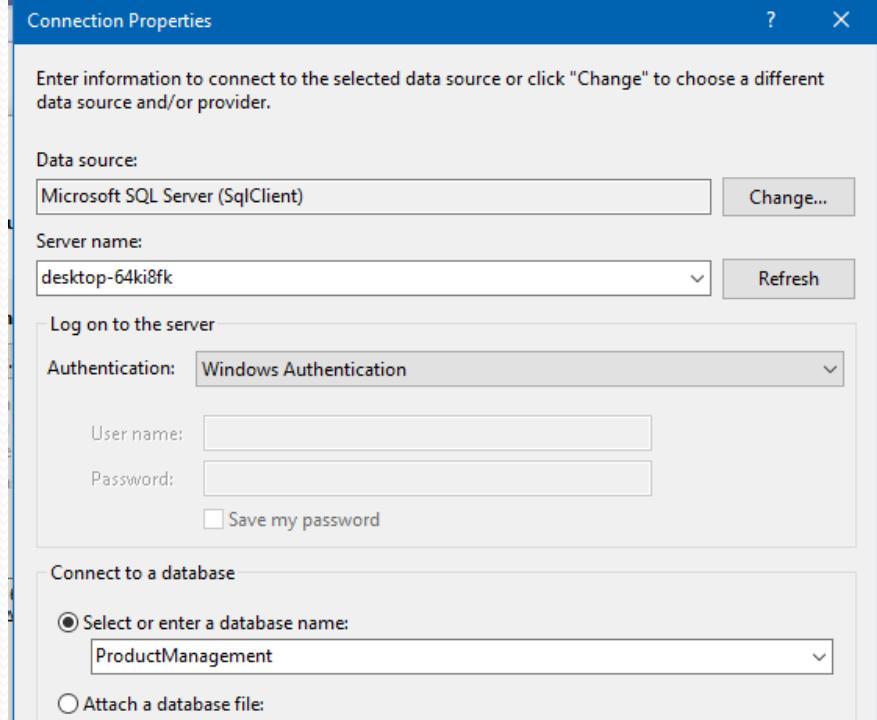
- Kết nối với SQL Server: View → Server Explorer.
- Tạo mới connection tới SQL Server: Chọn Connect to Database, chọn Server Name của SQL Server.
- Tạo mới database trong SQL Server: R\_Click vào Server SQL vừa kết nối chọn New Query.



# Code First

## ➤ Tạo bối cảnh (context)

- Tạo lớp `DbContext` trong Models: Add New Item chọn Installed → Visual C# → Data → ADO.NET Entity Data Model (**Code First from database**)



# Code First

---

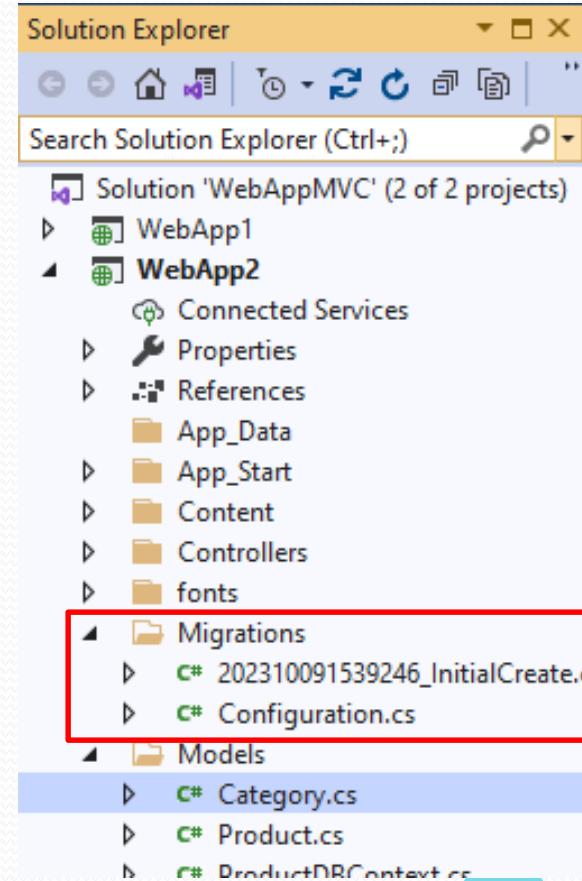
## ➤ Đọc & Ghi dữ liệu:

- Tạo Controller: R\_Click → New Controller → MVC 5 Controller with views, using Entity Framework.
- Chạy project để phát sinh database và nhập dữ liệu.
- Minh họa trong phần Demo.

# Code First

## ➤ Thay đổi mô hình

- Sử dụng tính năng Code First Migrations.
- Kích hoạt chế độ Code First Migration:  
Tool → Nuget package console (cửa sổ *Package manager consoles*) nhập đoạn code *Enable-Migrations* để thực thi enable migration.
- Thư mục Migrations sẽ tự động được tạo ra, trong đó có file *Initial Create* và *Configuration*



# Code First

## ➤ Thay đổi mô hình

- Thêm thuộc tính vào Product

```
public class Product
{
    public int ProductID { get; set; }
    public string NameOfProduct { get; set; }
    public decimal PriceOfProduct { get; set; }
    public int CategoryID { get; set; }
}
```

- Chạy lệnh **Add-Migration AddUrl**

```
2 references
public partial class InitialCreate1 : DbMigration
{
    0 references
    public override void Up()
    {
        AddColumn("dbo.Products", "PriceOfProduct", c => c.Decimal(nullable: false, precision: 18, scale: 2));
    }

    0 references
    public override void Down()
    {
    }
}
```

# Code First

## ➤ Thay đổi mô hình

- Chạy lệnh *Update-Database* trong *Package Manager Console* để cập nhật mô hình mới trong database.

Column Name	Data Type	Allow Nulls
ProductID	int	<input type="checkbox"/>
NameOfProduct	nvarchar(MAX)	<input checked="" type="checkbox"/>
CategoryID	int	<input type="checkbox"/>
PriceOfProduct	decimal(18, 2)	<input type="checkbox"/>

# Data Annotation

## ➤ Chú thích dữ liệu (Data Annotation)

- Trong .NET Framework, Data Annotation dùng để **thêm phần ý nghĩa mở rộng** vào dữ liệu thông qua các thẻ thuộc tính.
- Tính năng Data Annotation được Microsoft giới thiệu lần đầu ở .NET 3.5 tại namespace System.ComponentModel.DataAnnotations. Namespace này chứa các lớp dùng để định nghĩa thuộc tính mở rộng cho dữ liệu.

# Data Annotation

## ➤ Chú thích dữ liệu (Data Annotation)

- Các thuộc tính Data Annotation được phân chia thành 3 thể loại chính:
  - *Thuộc tính xác nhận* (Validation Attribute): dùng để thêm các tập luật xác nhận cho dữ liệu.
  - *Thuộc tính hiển thị* (Display Attribute): dùng để đặc tả cách dữ liệu từ một lớp được hiển thị ở giao diện.
  - *Thuộc tính mô hình* (Modelling Attribute): dùng để đặc tả mục đích sử dụng của lớp thành viên và mối quan hệ giữa các lớp.

# Data Annotation

## ➤ Chú thích dữ liệu (Data Annotation)

- Thêm thư viện DataAnnotations
  - using System.ComponentModel.DataAnnotations;
  - using System.ComponentModel;
- **Định nghĩa thuộc tính hiển thị** (Display attribute): Để định nghĩa thuộc tính hiển thị, đơn giản bạn chỉ cần thêm dòng **[DisplayName("tên cần hiển thị")]** vào trước các thuộc tính cần hiển thị.

# Data Annotation

## ➤ Chú thích dữ liệu (Data Annotation)

- **Định nghĩa thuộc tính xác nhận dữ liệu:** định nghĩa kiểu dữ liệu cho thuộc tính với các loại chú thích sau:
  - *Timestamp:* Thuộc tính này chỉ áp dụng cho kiểu dữ liệu mảng byte, dùng để tạo ra một cột dữ liệu thời gian trong database SQL.
  - *ConcurrencyCheck:* Thuộc tính ConcurrencyCheck áp dụng cho một hay nhiều thuộc tính và các cột tương ứng trong database sẽ được kiểm tra đồng bộ thông qua mệnh đề Where.
  - *Required:* chỉ định thuộc tính phải có dữ liệu nhập vào trước khi submit về server.

# Data Annotation

## ➤ Chú thích dữ liệu (Data Annotation)

- **Định nghĩa thuộc tính xác nhận dữ liệu:** định nghĩa kiểu dữ liệu cho thuộc tính với các loại chú thích sau:
  - *MinLength*: chiều dài tối thiểu của thuộc tính.
  - *MaxLength*: chiều đa tối thiểu của thuộc tính.
  - *StringLength*: định nghĩa chiều dài thuộc tính, cho phép đặc tả cả chiều dài tối thiểu và tối đa.
  - *Range*: định nghĩa giá trị số tối thiểu và tối đa của một thuộc tính.

# Data Annotation

## ➤ Chú thích dữ liệu (Data Annotation)

- **Định nghĩa thuộc tính mô hình:**

- *Key*: định nghĩa thuộc tính nào là khóa chính.
- *Table*: bảng
- *Column*: cột
- *Index*: chỉ mục
- *ForeignKey*: khóa ngoại
- *NotMapped*: không ánh xạ (không kết nối)
- *InverseProperty*: Thuộc tính đảo ngược, dùng để tạo quan hệ giữa các bảng 1-n hoặc n-n.

# Validation

- ❑ Dữ liệu không hợp lệ nhập từ người sử dụng sẽ gây các lỗi khó lường.
- ❑ Vì vậy việc kiểm soát dữ liệu vào luôn đóng vai trò quan trọng.
- ❑ Các lỗi thường gặp
  - ☞ Để trống ô nhập...
  - ☞ Không đúng định dạng email, creditcard, url...
  - ☞ Sai kiểu số nguyên, số thực, ngày giờ...
  - ☞ Không hợp lệ - phải có giá trị tối thiểu, tối đa, trong phạm vi...
  - ☞ Không đúng như kết quả tính toán trước

## Kiểm lỗi trong MVC

- ❑ Dữ liệu không hợp lệ nhập từ người sử dụng sẽ gây các lỗi khó lường.
- ❑ Vì vậy việc kiểm soát dữ liệu vào luôn đóng vai trò quan trọng.
- ❑ Các lỗi thường gặp
  - Để trống ô nhập...
  - Không đúng định dạng email, creditcard, url...
  - Sai kiểu số nguyên, số thực, ngày giờ...
  - Không hợp lệ - phải có giá trị tối thiểu, tối đa, trong phạm vi...
  - Không đúng như kết quả tính toán trước

# Mã lệnh tương ứng

## ❑ Mã trên Model

- Đính kèm các Attribute kiểm lỗi cho các Property
  - ✓ [Required], [StringLength]...

## ❑ Mã trên View

### ➤ Hiển thị lỗi

- ✓ @Html.ValidationMessageFor(Property)
- ✓ @Html.ValidationSummary()

### ➤ Kiểm lỗi phía client

- ✓ @Scripts.Render("~/bundles/jquery")

## ❑ Mã trên Controller

### ➤ Kiểm lỗi phía server

- ✓ ModelState.IsValid
- ✓ ModelState.AddModelError()

# Models

```
public class EmployeeInfo
{
    [MinLength(5, ErrorMessage="Tên ít nhất 5 ký tự !")]
    public String FullName { get; set; }
    [Required(ErrorMessage="Không để trống !")]
    [Range(16, 65, ErrorMessage = "Tuổi phải từ 16 đến 65 !")]
    public int Age { get; set; }
}
```

Annotation	Thuộc tính	Mô tả
[MinLength]	FullName	Giới hạn số lượng ký tự tối thiểu là 5. Nếu không nhập vẫn hợp lệ vì không sử dụng Required
[Required]	Age	Không để trống
[Range]	Age	Giới hạn tuổi từ 16 đến 65

# Controllers

```
public class ValidatorController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

Kiểm lỗi phía server

```
public ActionResult Validate(EmployeeInfo model)
{
    if (ModelState.IsValid)
    {
        ModelState.AddModelError("", "Chúc mừng bạn đã nhập đúng !");
    }
    return View("Index");
}
```

Kiểm lỗi

Họ và tên

Tuổi

Kiểm lỗi

Bổ sung thông báo lỗi model

# Views

```
@model Mvc5CodeDemo.Models.EmployeeInfo
<h2>Kiểm lỗi</h2>
@Html.ValidationSummary(true)
@using (Html.BeginForm("Validate", "Validator"))
{
    <div>Họ và tên</div>
    @Html.TextBoxFor(m => m.FullName)
    @Html.ValidationMessageFor(m => m.FullName)
    <div>Tuổi</div>
    @Html.TextBoxFor(m => m.Age)
    @Html.ValidationMessageFor(m => m.Age)
    <hr />
    <input type="submit" value="Kiểm lỗi" />
}

@section scripts{
    @Scripts.Render("~/bundles/jqueryval")
}
```

Thông báo lỗi chung không bao gồm lỗi  
đã thông báo cho từng thuộc tính

Thông báo lỗi cho từng thuộc tính

Thực hiện kiểm lỗi phía client

# Thuộc tính kiểm lỗi

Annotation	Mô tả	Ví dụ
[Required]	Bắt buộc	[Required] public String Name{get;set;}
[Range(Min, Max)]	Giới hạn số trong khoảng	[Range(16, 65)] public String Age{get;set;}
[StringLength(Max)]	Giới hạn độ dài chuỗi	[StringLength (20, MinimumLength=5)] public String Password{get;set;}
[EmailAddress]	Định dạng email	[EmailAddress] public String Email{get;set;}
[CreditCard]	Định dạng số thẻ tín dụng	[CreditCard] public String CardNumber{get;set;}
[Url]	Định dạng url	[Url] public String Website{get;set;}
[Compare(Property)]	So sánh giá trị	[Compare("Password")] public String ConfirmPassword{get;set;}
[RegularExpression(Regex)]	So khớp chuỗi	[RegularExpression("\d{9}")] public String IdCard{get;set;}
[MinLength(Min)]	Giới hạn tối thiểu chuỗi, mảng	[MinLength(1)] public String[] Hobbies{get;set;}
[MaxLength (Max)]	Giới hạn tối đa chuỗi, mảng	[MaxLength (255)] public String Description{get;set;}

# Thẻ HTML

□ [DataType(DataType.Password, ErrorMessage = "")]

- DataType.CreditCard
- DataType.Currency
- DataType.Date
- DataType.DateTime
- DataType.Duration
- DataType.EmailAddress
- DataType.Html
- DataType.ImageUrl
- DataType.MultilineText
- DataType.Password
- DataType.PhoneNumber
- DataType.PostalCode
- DataType.Text
- DataType.Time
- DataType.Upload
- DataType.Url

# Kiểm lỗi bằng mã lệnh

```
public ActionResult Validate(String FullName, int Age)
{
    if (String.IsNullOrEmpty(FullName))
    {
        ModelState.AddModelError("FullName", "Không để trống họ và tên");
    }
    else if (FullName.Length < 5)
    {
        ModelState.AddModelError("FullName", "Ít nhất 5 ký tự !");
    }

    if (Age < 16 && Age > 65)
    {
        ModelState.AddModelError("Age", "Tuổi phải từ 16 đến 65 !");
    }

    if (ModelState.Count == 0) // không có lỗi nào
    {
        ModelState.AddModelError("", "Chúc mừng bạn đã nhập đúng !");
    }
    return View("Index");
}
```

# Kiểm lỗi tùy biến

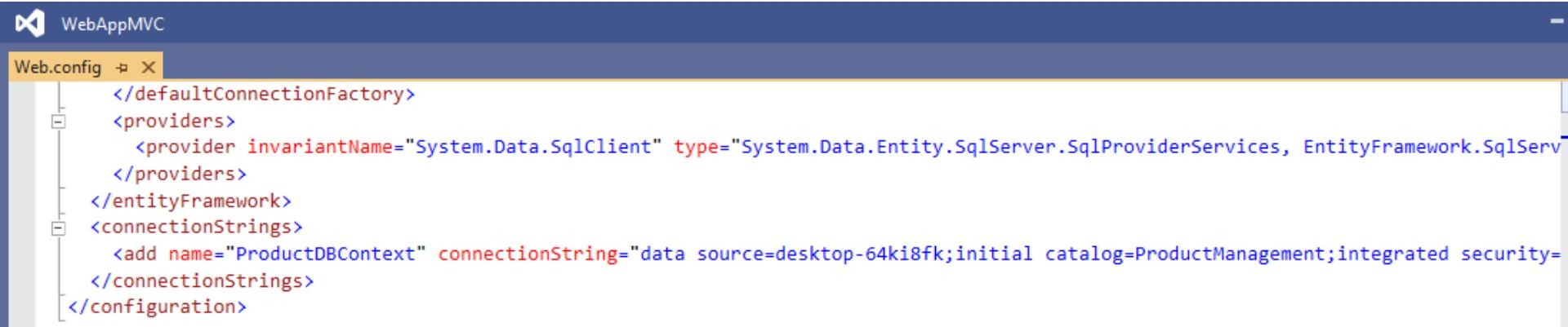
```
public sealed class EvenNumberAttribute : ValidationAttribute
{
    public EvenNumberAttribute() : base("Vui lòng nhập số chẵn !") { }

    public override bool IsValid(object value)
    {
        if (value == null)
        {
            return true;
        }
        return Convert.ToInt64(value) % 2 == 0;
    }
}
```

[EvenNumber]  
public String Age{get;set}

# Connection strings

- Chuỗi kết nối Cơ sở dữ liệu (connection strings) trong tập tin Web.config

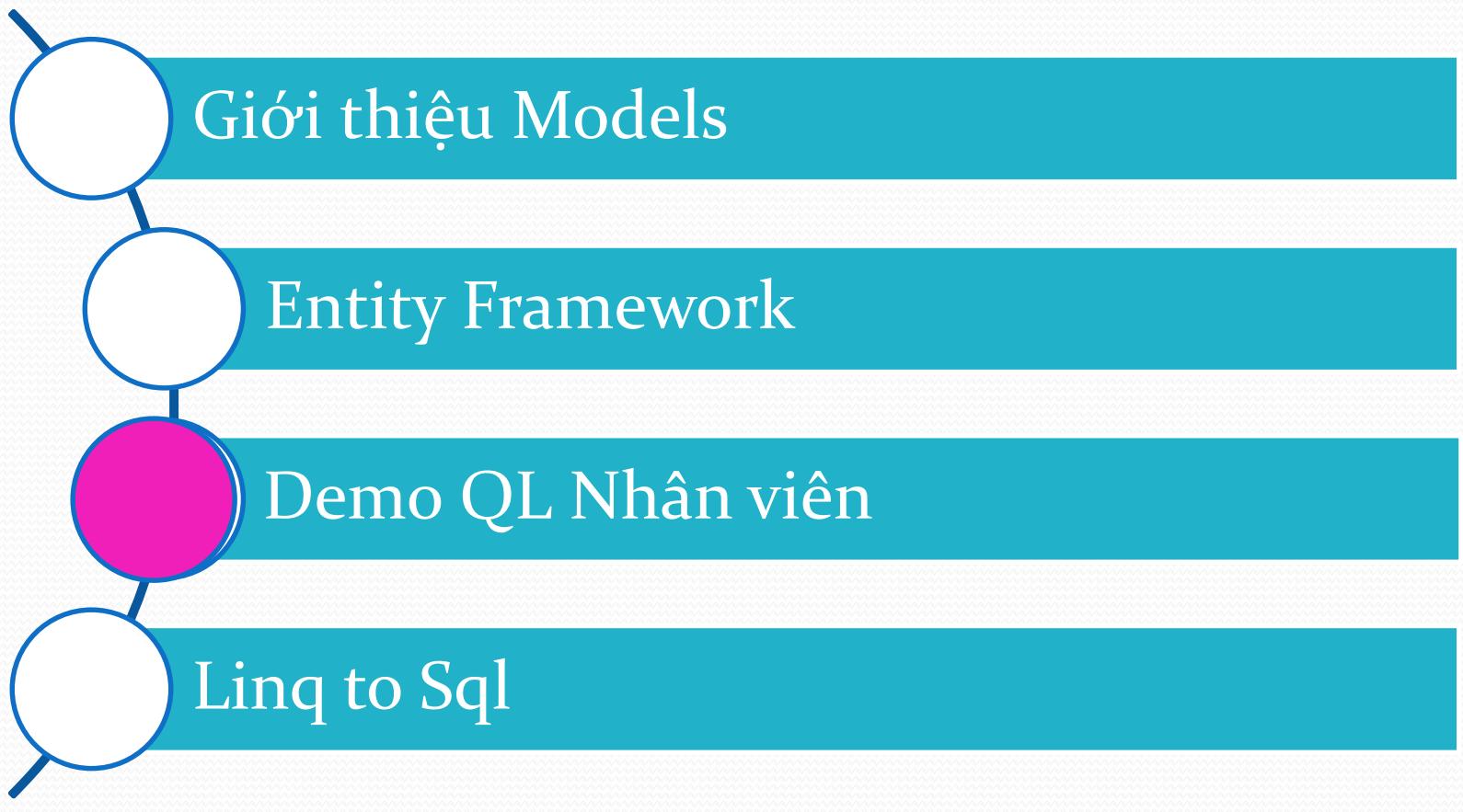


WebAppMVC

Web.config

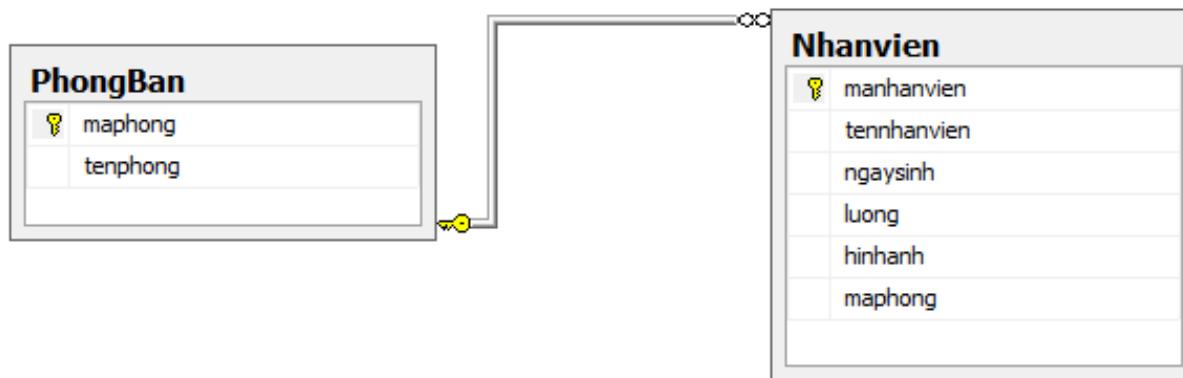
```
</defaultConnectionFactory>
<providers>
  <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer"></provider>
</providers>
</entityFramework>
<connectionStrings>
  <add name="ProductDBContext" connectionString="data source=desktop-64ki8fk;initial catalog=ProductManagement;integrated security=true;multipleactiveresultsets=true;maxpoolsize=100;minpoolsize=10;packet size=4096;pooling=true;timeout=30;"/>
</connectionStrings>
</configuration>
```

# Nội dung



# Entity Framework

➤ Ví dụ: CSDL QLNhanvien theo lược đồ



# Context và Entity

## ➤ Ánh xạ Entity-Table

- Mỗi thực thể trong EDM được ánh xạ với các bảng cơ sở dữ liệu. Chúng ta có thể kiểm tra ánh xạ thực thể-bảng bằng cách kích chuột phải vào bất kỳ thực thể trong thiết kế *EDM* → chọn *TableMapping*.

## ➤ Lớp Context và Entity

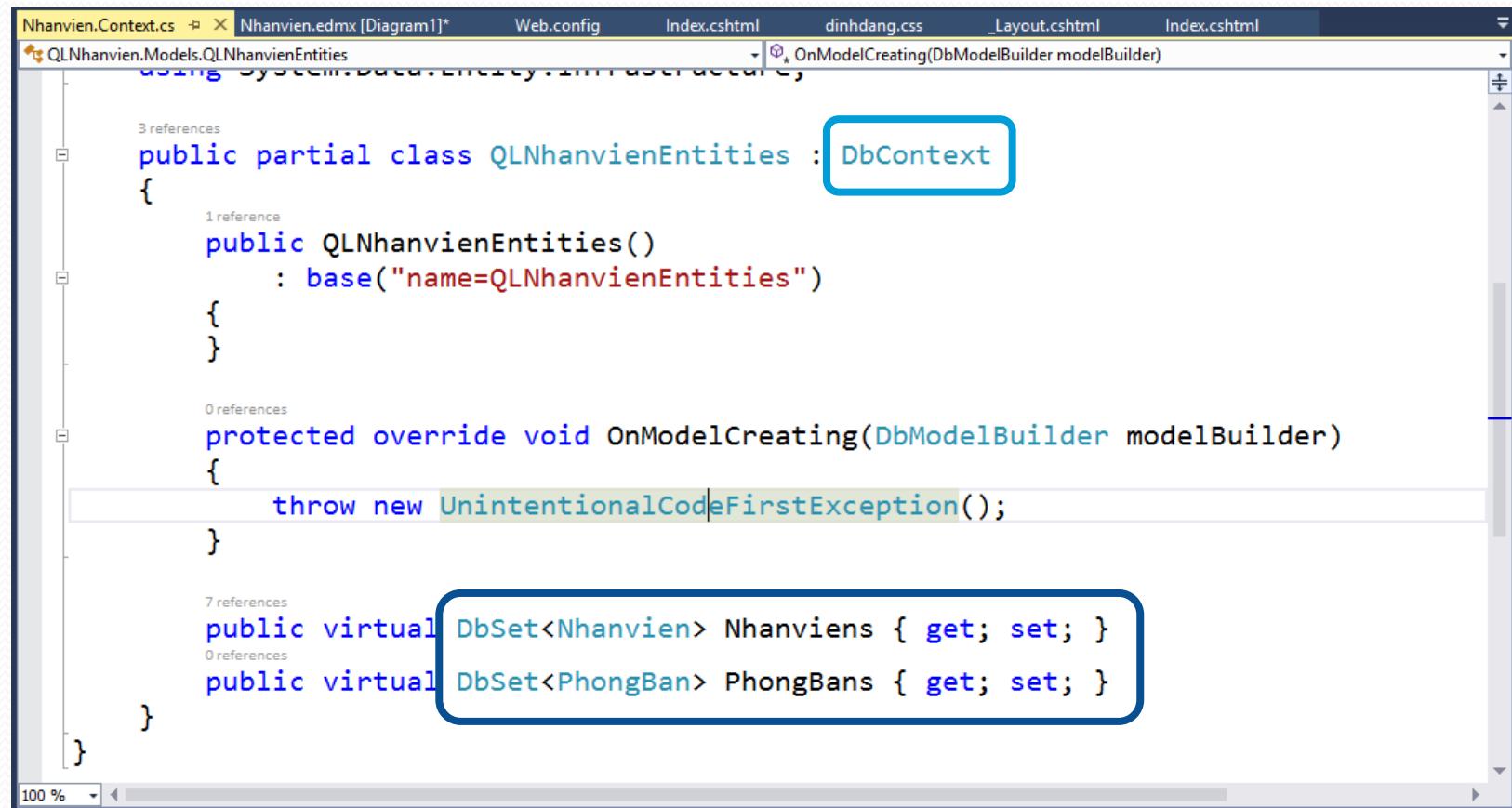
- Entity Data Model tạo một lớp *context* và các lớp *entity* cho các bảng trong cơ sở dữ liệu.

# Context và Entity

## ➤ **Nhanvien.Context.tt:**

- **Entity Data Model** (file .edmx) thì file template này tạo lớp context. các tập tin lớp context bằng cách mở rộng *Nhanvien.Context.tt*. Lớp context này được kế thừa từ lớp *DbContext* trong Entity Framework.

# Context và Entity



The screenshot shows a code editor window with the following code:

```
Nhanvien.Context.cs X Nhanvien.edmx [Diagram1]* Web.config Index.cshtml dinhdang.css _Layout.cshtml Index.cshtml
QLNhanvien.Models.QLNhanvienEntities
using System.Data.Entity;
public partial class QLNhanvienEntities : DbContext
{
    public QLNhanvienEntities()
        : base("name=QLNhanvienEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<Nhanvien> Nhanviens { get; set; }
    public virtual DbSet<PhongBan> PhongBans { get; set; }
}
```

Annotations in the code:

- A blue rounded rectangle highlights the line `public partial class QLNhanvienEntities : DbContext`.
- A blue rounded rectangle highlights the declaration `public virtual DbSet<Nhanvien> Nhanviens { get; set; }` and the declaration `public virtual DbSet<PhongBan> PhongBans { get; set; }`.

# Lớp DBContext Entity Framework

- EDM tạo lớp SchoolDBEntities kế thừa từ lớp ***System.Data.Entity.DbContext***.
- DbContext là thành phần quan trọng của Entity Framework.
- DbContext tương ứng với cơ sở dữ liệu (một tập hợp gồm **nhiều table**), có thể nói DbContext tương ứng với database.
- DbContext là một tập hợp gồm nhiều DbSet.
- Phương thức DbContext: SaveChange()
- Một số hoạt động DBContext

# Lớp DBContext Entity Framework

➤ **EntitySet:** DbContext chứa tập thực thể (DbSet < TEntity >) cho tất cả các thực thể được ánh xạ đến các bảng cơ sở dữ liệu.

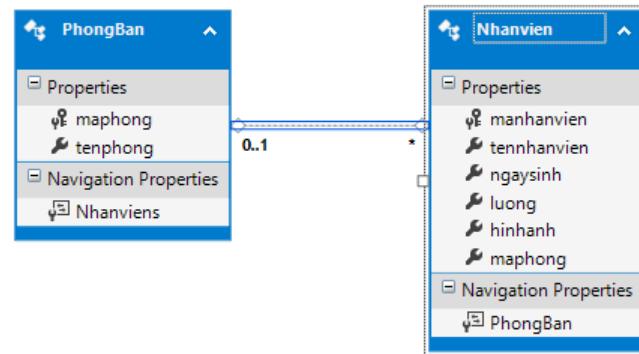
Ví dụ: Lớp context bao gồm tập thực thể kiểu DbSet< TEntity > cho các thực thể.

```
DbSet< Nhanvien > Nhanviens { get; set; }  
DbSet< PhongBan > PhongBans { get; set; }
```

➤ **Truy vấn:** DbContext chuyển đổi truy vấn LINQ-to-Entities thành truy vấn SQL và gửi truy vấn đến cơ sở dữ liệu.

# Quan hệ giữa các thực thể trong DBContext

- Một – Một
- Một – nhiều
- Nhiều – nhiều



# Lớp DbSet Entity Framework

- Lớp DbSet Entity framework thể hiện một tập thực thể được sử dụng thực hiện các hoạt động: **tạo, truy vấn, cập nhập, và xóa** thực thể, DbSet Entity Framework là một phiên bản Generic của lớp DBDet< TEntity > sử dụng cho một kiểu thực thể xác định.

```
DbSet<Nhanvien> Nhanviens { get; set; }  
DbSet<PhongBan> PhongBans { get; set; }
```

- DbSet tương ứng với mỗi bảng trong CSSDL.
- Lớp DbSet chứa trong DbContext như ví dụ trên

# Lớp DBSet Entity Framework

Một số phương thức thường dùng:

Phương thức	Kiểu trả về	Mô tả
ToList	Hiển thị tất cả thực thể về 1 danh sách → liệt kê	db.Nhanviens.ToList()
Add	Kiểu thực thể được thêm vào	db.Nhanviens.Add(nv);
Find	Tìm kiếm một thực thể theo khóa chính <code>Nhanvien</code> nv = db.Nhanviens.Find(id);	
Remove	Xóa thực thể	db.Nhanviens.Remove(nv);
SaveChange()	DbContext lưu lại sự thay đổi của các thực thể trong từng Dbset như Remove, Add, ...	db.SaveChanges();

# Một số quy ước DbContext và DBSet

- Dạng số nhiều của tên lớp thực thể sẽ được dùng như tên bảng  
Ví dụ: Nhanviens, Phongbans
- Tên các thuộc tính sẽ được dùng như tên cột.
- Ví dụ: nv.masonv, nv.hoten, ...
- Các thuộc tính ID hay maso được coi là khóa chính
- Entity Framework kết nối đến CSDL bằng cách tìm chuỗi kết nối có cùng tên với lớp Dbcontext. (NhanvienContext)

```
<connectionStrings>
  <add name="QLNhanvienEntities" connectionString="metadata=res://*/Mod
</connectionStrings>
```

# Demo QLNhanvien

- Thực hiện hiển thị toàn bộ danh sách các thông tin của nhân viên trong một bảng Nhanvien.

```
public ActionResult Index()
{
    return View(db.Nhanviens.ToList());
}
```

# Demo QLNhanvien

## ➤ Thêm mới nhân viên:

```
public ActionResult them()
{
    return View();
}
[HttpPost]
public ActionResult them([Bind(Include = "manhanvien, tennhanvien, ngaysinh, luong,hinhanh")] Nhanvien nv)
{
    if (ModelState.IsValid)
    {
        db.Nhanviens.Add(nv);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(nv);
}
```

# Demo QLNhanvien

- Chỉnh sửa thông tin của nhân viên: Để chỉnh sửa phải tìm ra nhân viên cần chỉnh sửa theo id truyền vào.

```
public ActionResult chinh sua(int id)
{
    id = Convert.ToInt32(id);
    Nhanvien nvien = db.Nhanviens.Find(id);
    return View(nvien);
}
[HttpPost]
public ActionResult chinh sua([Bind(Include = "manhanvien, tennhanvien, ngaysinh, luong,hinhanh")] Nhanvien nv)
{
    db.Entry(nv).State = EntityState.Modified;
    db.SaveChanges();
    return View(nv);
}
```

# Demo QLNhanvien

- Xóa nhân viên: Tìm nhân viên theo id để xóa

```
public ActionResult xoa(int id)
{
    id = Convert.ToInt32(id);
    var nv = db.Nhanviens.Find(id);
    return View(nv);
}

[HttpPost, ActionName("xoa")]
public ActionResult DeleteConfirm(int id)
{
    id = Convert.ToInt32(id);
    Nhanvien nv = db.Nhanviens.Find(id);
    db.Nhanviens.Remove(nv);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

# Demo QLNhanvien

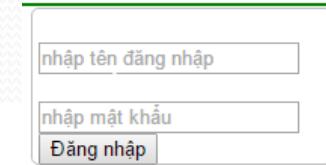
- Hiển thị chi tiết thông tin của nhân viên: Tìm nhân viên theo id

```
public ActionResult chitiet(int id)
{
    id = Convert.ToInt32(id);
    var nv = db.Nhanviens.Find(id);
    return View(nv);
}
```

# Demo QLNhanvien

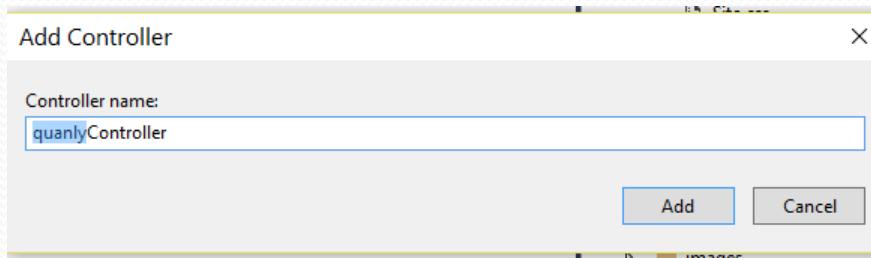
- Hướng dẫn làm form đăng nhập trên trang layout
- Mở tập tin layout.cshtml tạo form như trên:

```
<div id="links">
    <div id="LoginForm">
        @using (Html.BeginForm("DangNhap", "quanly", new
            { @strURL = Request.Url.ToString() }))
        {
            @Html.Label("Tên đăng nhập")
            @Html.TextBox("txtUser", "", htmlAttributes:
                new{@class = "textboxDangNhap", @placeholder = "nhập tên đăng nhập"})
            @Html.Label("Mật khẩu")
            @Html.Password("txtPass", "", htmlAttributes: new
            {
                @class = "textboxDangNhap",
                @placeholder = "nhập mật khẩu"
            })
            <input type="submit" name="btnDangNhap" class="buttonDangNhap" value="Đăng nhập">
        }
    </div>
```



# Demo QLNhanvien

- Trong thư mục Controller → tạo **quanlyController.cs**



```
public ActionResult Index()
{
    return View();
}
```

# Demo QLNhanvien

Thêm **ActionResult DangNhap()** truyền giá trị đăng nhập từ form bên layout:

```
[HttpPost]
public ActionResult DangNhap(string strURL, FormCollection f)
{
    string username = f["txtUser"].ToString();
    string password = f["txtPass"].ToString();

    if (CheckUser(username, password))
    {
        //tạo session
        Session["maTaiKhoan"] = username;
        HttpCookie ck = new HttpCookie("myCookies");
        ck["name"] = username;
        //tạo cookies
        Response.Cookies.Add(ck);
        ck.Expires = DateTime.Now.AddDays(3); //giới hạn thời gian 3 ngày
        return Redirect(strURL);
    }
    ViewBag.ThongBao = "Tên tài khoản hoặc mật khẩu không đúng!";
    return Redirect(strURL);
}
```

# Demo QLNhanvien

- Tạo hàm **CheckUser(string username, string password)** kiểm tra username và password từ bảng Account của EDM:

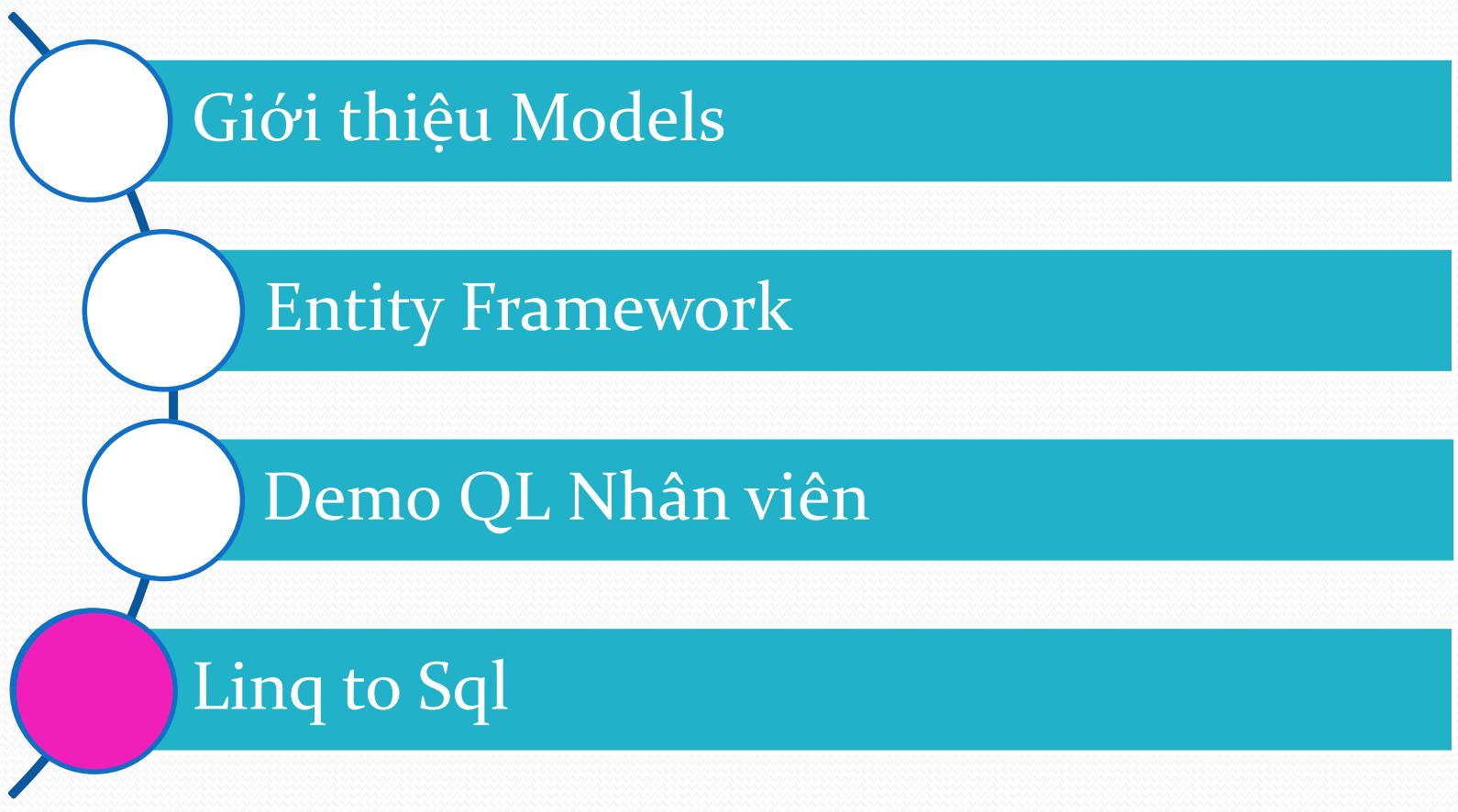
```
public bool CheckUser(string username, string password)
{
    using (var db = new SinhvienEntities ())
    {
        var kq = db.Accounts.Where(x => x.Ten == username &&
            x.Matkhau == password).ToList<Account>();
        if (kq.Count() > 0)
            return true;
        return false;
    }
}
```

# Demo QLNhanvien

- Tạo ActionResult DangXuat()

```
public ActionResult DangXuat(string strURL)
{
    //Xóa Session
    Session.Abandon();
    //xóa cookies
    if (Request.Cookies["myCookies"] != null)
    {
        HttpCookie myCookie = new HttpCookie("myCookies");
        myCookie.Expires = DateTime.Now.AddDays(-1d);
        Response.Cookies.Add(myCookie);
    }
    return RedirectToAction(strURL);
}
```

# Nội dung



# Linq to Sql

---

- **LINQ** - Language Integrated Query (tạm dịch là ngôn ngữ truy vấn tích hợp) - là cách thức truy vấn dữ liệu từ một tập hợp dữ liệu.
- **Thành phần cấu thành LINQ**
  - Nguồn dữ liệu.
  - Tạo câu lệnh truy vấn.
  - Thực thi truy vấn để lấy kết quả.

# Nguồn dữ liệu

Tên provider	Mô tả
LINQ to Objects	Sử dụng LINQ đối với các đối tượng collection mà implement từ IEnumerable hoặc IEnumerable<T> (dữ liệu được lưu trong bộ nhớ). Được sử dụng rộng rãi đặc biệt đối với những bài toán cần hiệu năng cao.
LINQ to SQL	<p>Thực hiện map các tables, views, store procedures thành các đối tượng. LINQ sẽ thực hiện truy vấn trên các đối tượng đó bằng cách chuyển đổi qua lại giữa đối tượng và câu lệnh sql.</p> <p>Ngoài truy vấn ta cũng có thể thực hiện thêm/sửa/xóa dữ liệu dựa trên các đối tượng trên.</p> <p>Hỗ trợ transaction.</p> <p>Ưu điểm: được sử dụng khá nhiều trong thực tế dưới cái tên Entity Framework hoặc Entity Framework Core.</p> <p>Nhược điểm: chỉ làm việc với cơ sở dữ liệu là SQL Server.</p>
LINQ to Entities	Tương tự như LINQ to SQL nhưng hỗ trợ nhiều loại cơ sở dữ liệu.
LINQ to DataSets	Nhược điểm: sử dụng phức tạp. Nhiều cơ sở dữ liệu không thích hợp sử dụng chung với .Net.
LINQ to XML	Truy vấn thông tin trong file XML.

# Tạo câu lệnh truy vấn

- Có 2 cách để tạo ra câu truy vấn:
  - Cú pháp truy vấn (query syntax)
  - Cú pháp phương thức (method syntax)
- Để viết được câu truy vấn cần using thư viện System.Linq.
- Cú pháp truy vấn

```
var lists = from <Biến lưu thông tin từng phần tử> in <Nguồn dữ liệu>
[<Phép toán truy vấn: where, join ... in, order by...> Biểu thức lambda]
select <Biến lưu thông tin từng phần tử >
```

- Cú pháp phương thức

Là những phương thức mở rộng của I Enumerable hoặc I Enumerable<T>.

# Ví dụ

```
var result = from s in students
             where s.Marks > 9
             orderby s.Marks descending
             select new { s.Name, s.Marks };
```

**Biểu thức truy vấn**

**Kiểu nội bộ tự suy**

**Kiểu nặc danh**

**Khởi tạo đối tượng**

```
var result2 = students
             .Where(s => s.Marks > 9)
             .OrderByDescending(s => s.Marks)
             .Select(s => new { s.Name, s.Marks });
```

**Biểu thức lambda**

**Phương thức mở rộng**

# Kết thúc chủ đề 4

