

Chủ đề 4. 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 như sau:

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

The screenshot shows the Microsoft Visual Studio IDE interface. On the left is the code editor window, and on the right is the Solution Explorer window.

Code Editor (sanphamController.cs):

```
HomeController.cs          sanphamController.cs      Index.cshtml      DanhsachSP.cshtml      SP.cs
SampleMVC.Controllers.sanphamController      DanhsachSP()

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

namespace SampleMVC.Controllers
{
    public class sanphamController : Controller
    {
        //
        // GET: /sanpham/
        public ActionResult Chitietsp()
        {
            SP sp1 = new SP
            { maSP="001",
              tenSP="Lennovo",
              giaSP="560USD"
            };
            //truyen doi tuong sp1 vao View
            return View(sp1);
        }
    }
}
```

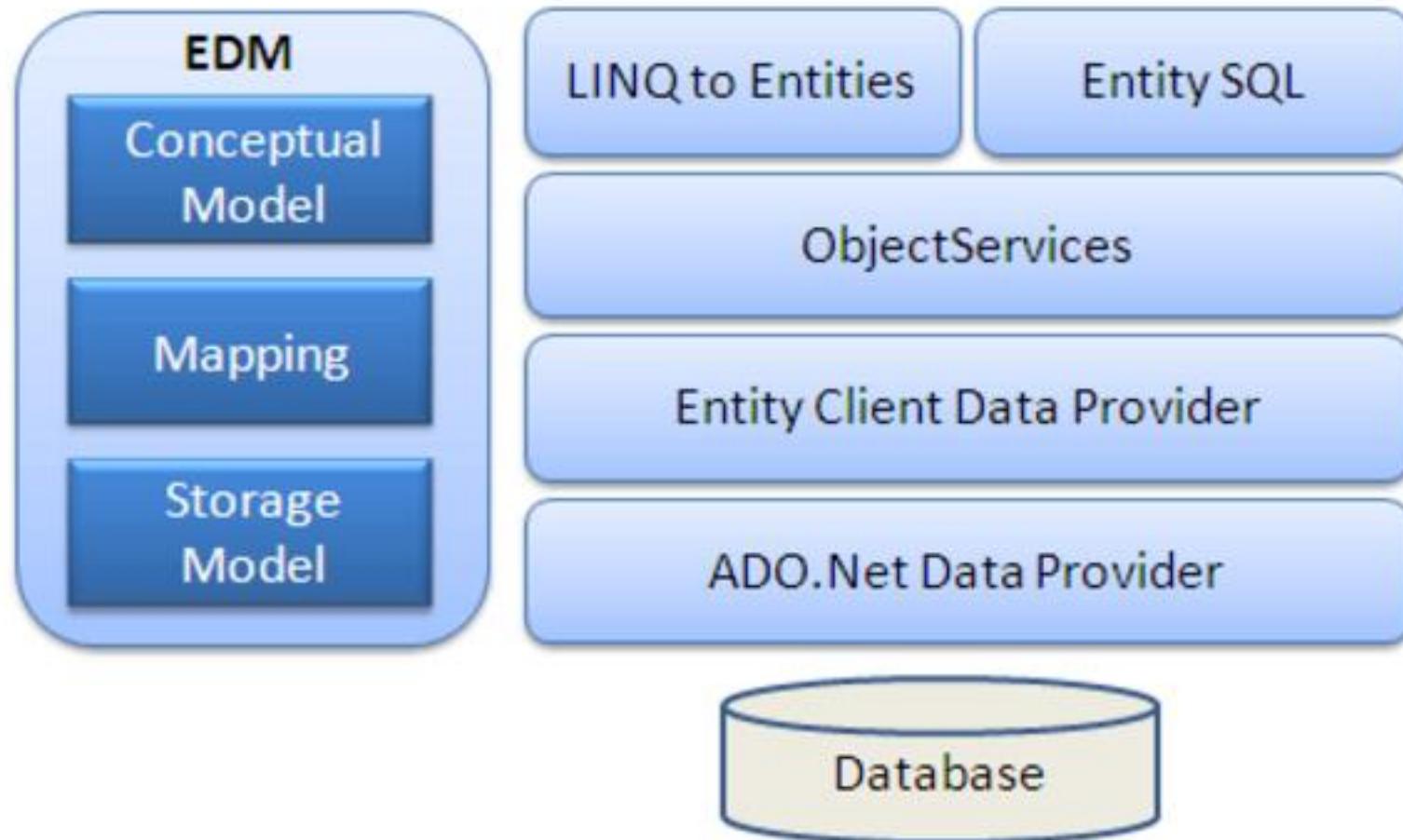
Solution Explorer:

- App_Start
- Content
- Controllers
 - AccountController.cs
 - HelloWorldController.cs
 - HomeController.cs
 - ProductController.cs
 - sanphamController.cs
- fonts
- Models
 - AccountViewModels.cs
 - IdentityModels.cs
 - SP.cs
- Scripts
- Views
 - Account
 - HelloWorld
 - Home
 - About.cshtml
 - Contact.cshtml
 - Index.cshtml
 - Product
 - Byld.cshtml
 - sanpham
 - Chitietsp.cshtml
 - DanhsachSP.cshtml
 - login.cshtml
 - trangchu.cshtml
- Shared

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.

Kiến trúc Entity Framework



- **Mô hình khái niệm (Conceptual Model):** Mô hình khái niệm chứa các lớp model và các mối quan hệ.
- **Ánh xạ (Mapping):** Ánh xạ bao gồm các thông tin về cách thức mô hình khái niệm được ánh xạ tới mô hình lưu trữ.
- **Mô hình lưu trữ (Storage Model):** Mô hình lưu trữ là mô hình thiết kế cơ sở dữ liệu bao gồm bảng, view, thủ tục lưu trữ, và các mối quan hệ và khóa chính.

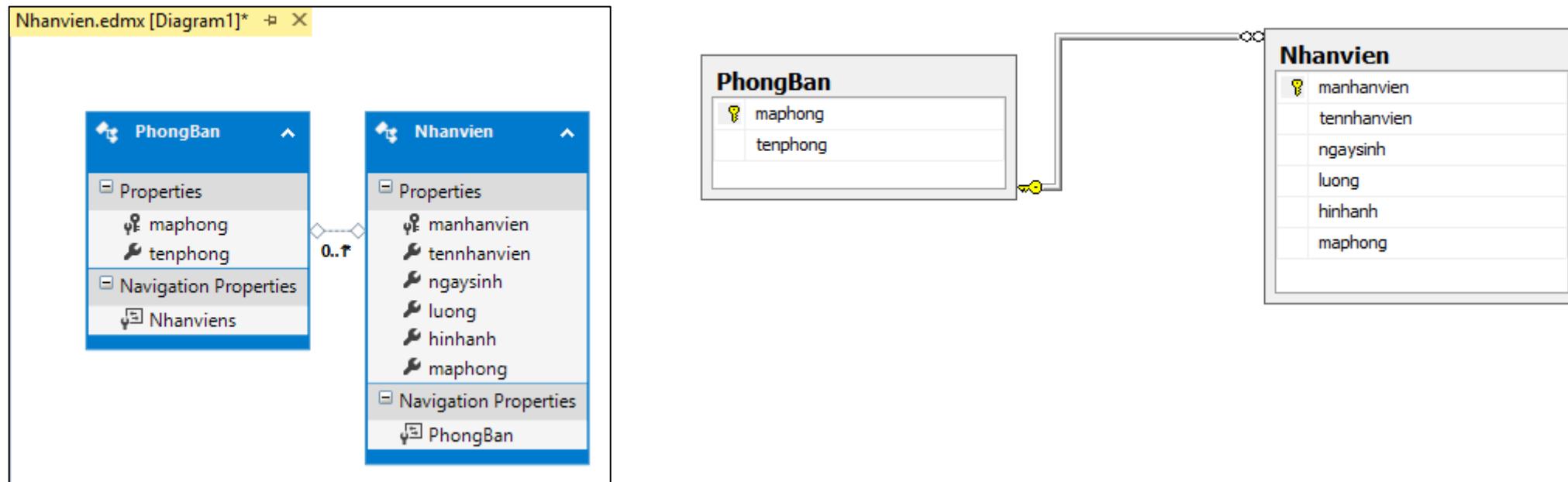
- **LINQ to Entities:** LINQ to Entities là ngôn ngữ truy vấn được sử dụng để viết các truy vấn trên mô hình đối tượng. Nó trả về các thực thể, được định nghĩa trong mô hình khái niệm. Chúng ta có thể sử dụng các câu lệnh LINQ ở đây.
- **Entity SQL:** Entity SQL là ngôn ngữ truy vấn giống như LINQ to Entities. Tuy nhiên, Entity SQL khó hơn LINQ to Entities và các nhà phát triển sẽ phải tìm hiểu nó một cách riêng biệt.
- **Object Service:** Object Service là một điểm vào chính cho việc truy cập dữ liệu từ cơ sở dữ liệu và trả lại Object Service trở lại. Object Service có nhiệm vụ chuyển đổi dữ liệu thực thể được cung cấp bởi client thành thực thể cấu trúc đối tượng.
- **Entity Client Data Provider:** Nhiệm vụ chính của lớp này là chuyển đổi L2E hoặc Entity SQL thành câu lệnh truy vấn SQL mà cơ sở dữ liệu thực thi được.
- **ADO.Net Data Provider:** Lớp này có nhiệm vụ giao tiếp với cơ sở dữ liệu bằng cách sử dụng ADO.Net chuẩn.

Khởi tạo Entity Data Model

- Entity Data Model là mô hình đối tượng thực thể và mối quan hệ giữa các thực thể với nhau.
- **System.Data.Entity** name space chứa EDM
- Bắt đầu tạo Entity Data Model **Nhanvien**
- Trong thư mục Model của cửa sổ Solution xuất hiện **Nhanvien.edmx** hiển thị tất cả các bảng CSDL

Khởi tạo Entity Data Model

Bước 1: Thiết kế CSDL QLNhanvien theo lược đồ sau:

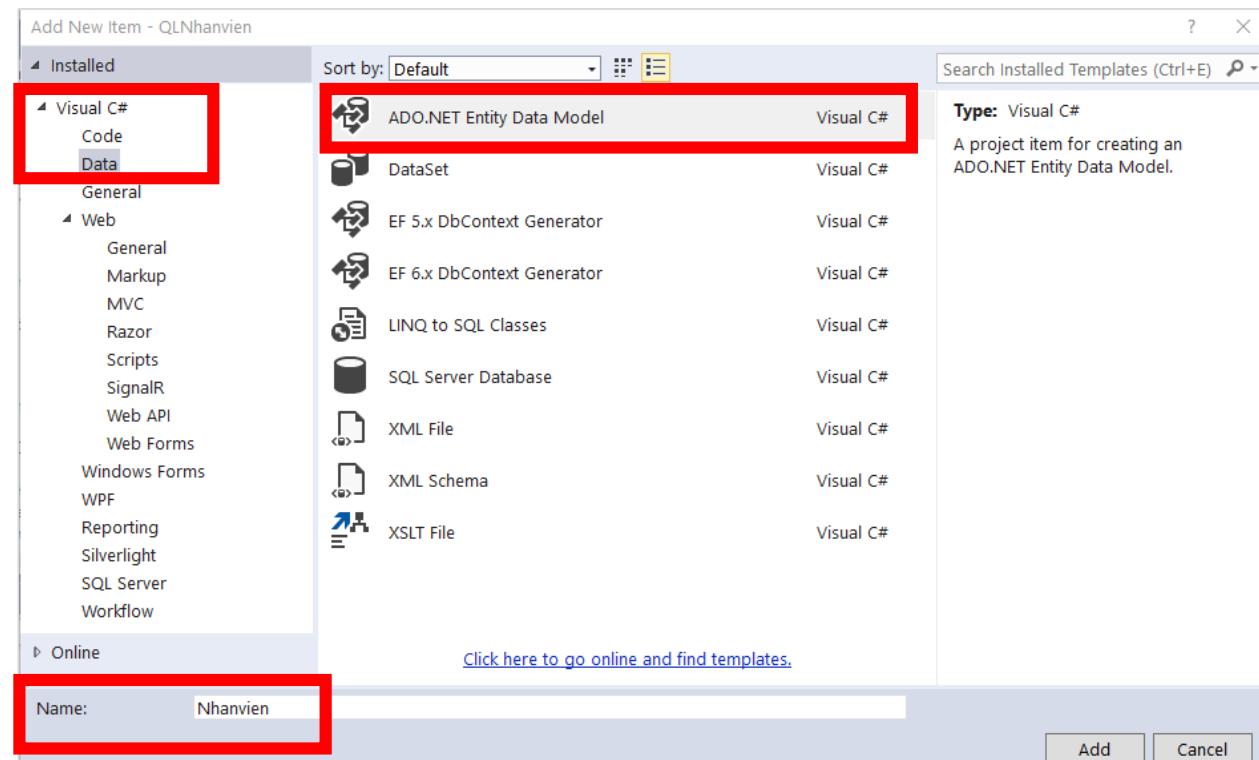


Khởi tạo Entity Data Model

Bước 2: Tạo Project trong Visual Studio

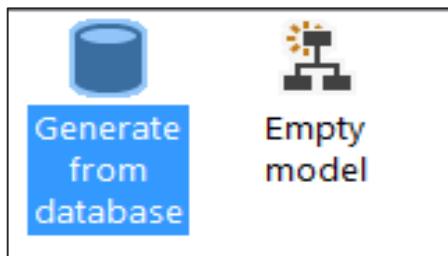
Bước 3: Trong cửa sổ solution explorer,
R_click Model → chọn Add
→ click New Item
→ chọn ADO.NET Entity Data Model.

Hiển thị cửa sổ như sau và
nhập **Nhanvien** vào ô name:

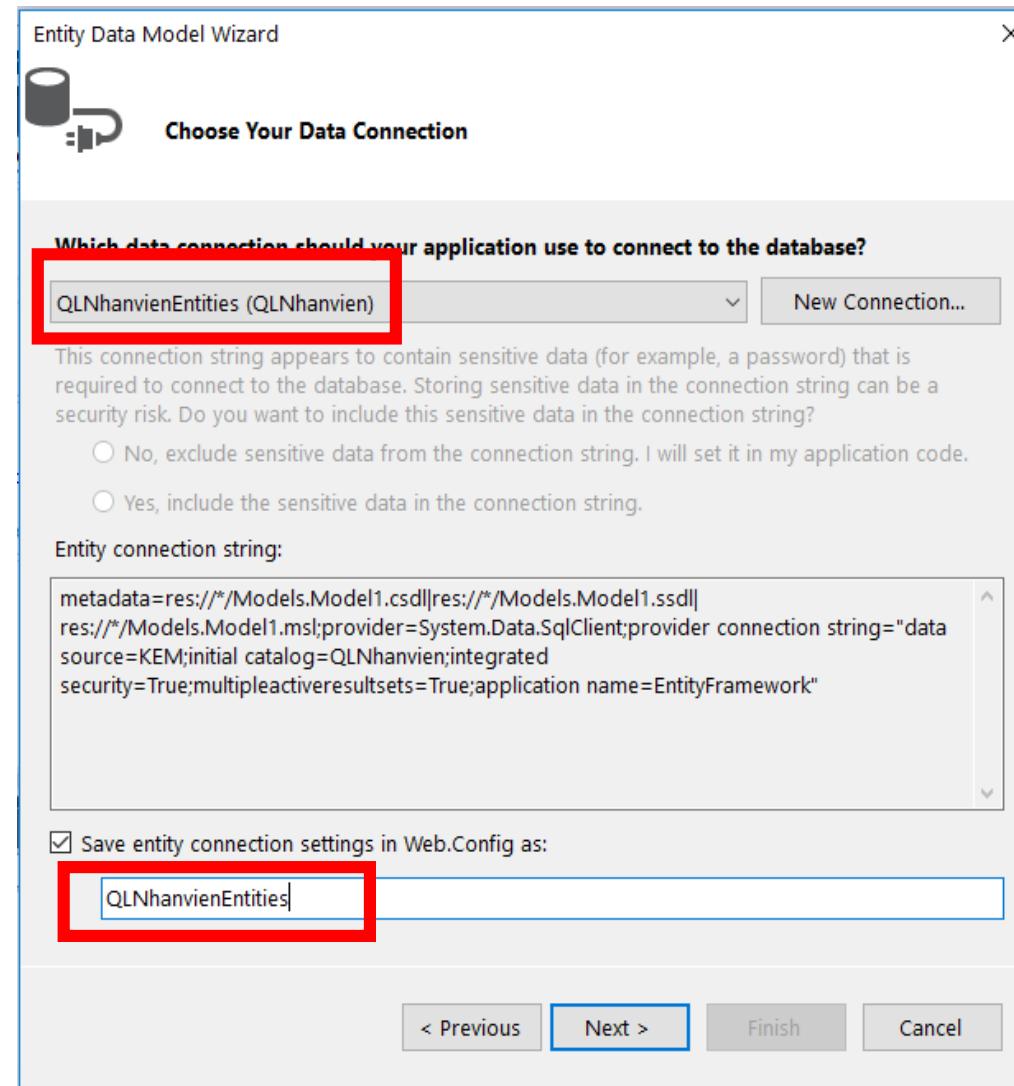


Khởi tạo Entity Data Model

Bước 4: EDM Winzard chọn Generate from Database → Next



Bước 5: Chúng ta có thể chọn chuỗi kết nối đã có hoặc tạo chuỗi kết nối mới đến cơ sở dữ liệu.



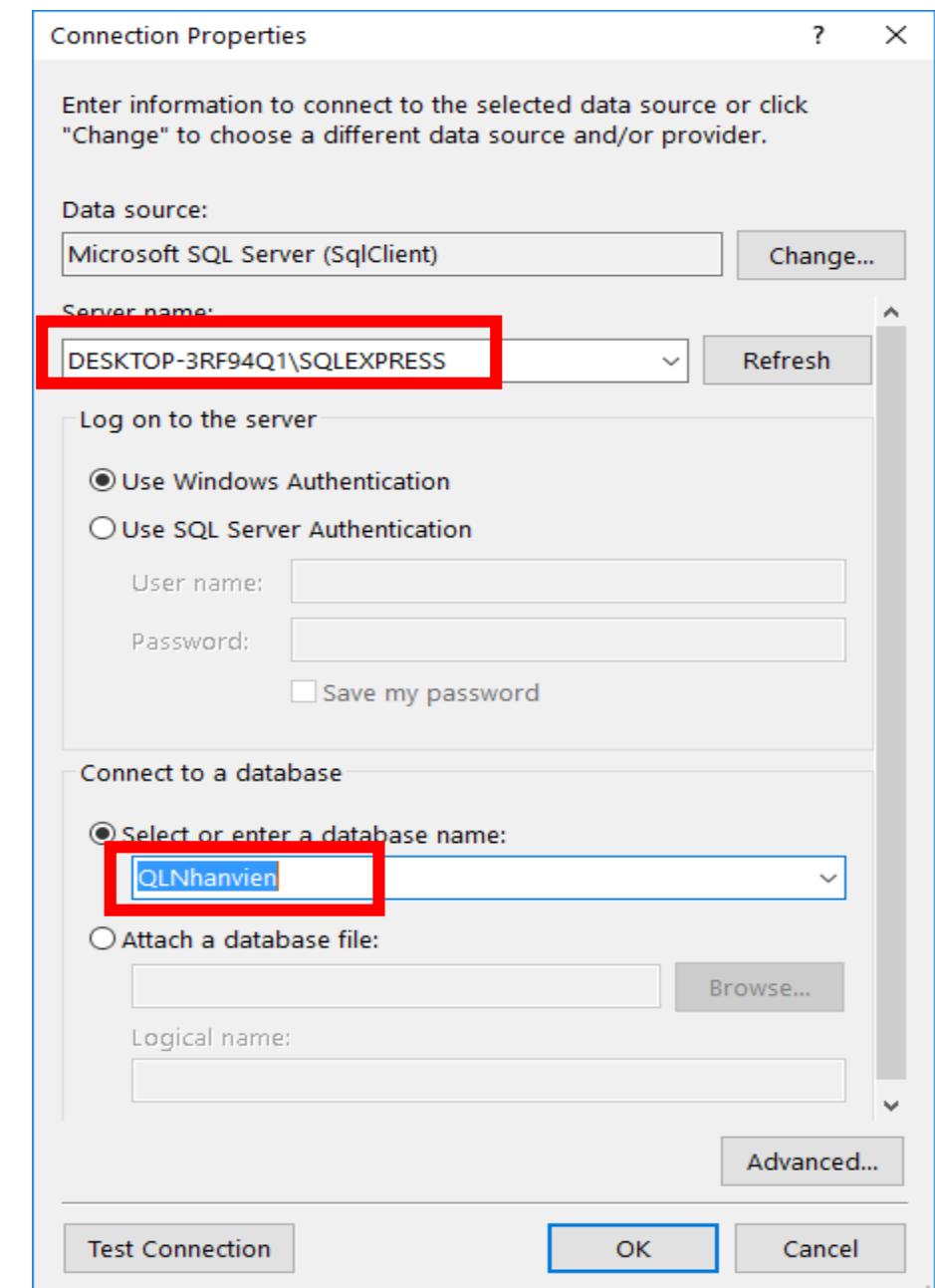
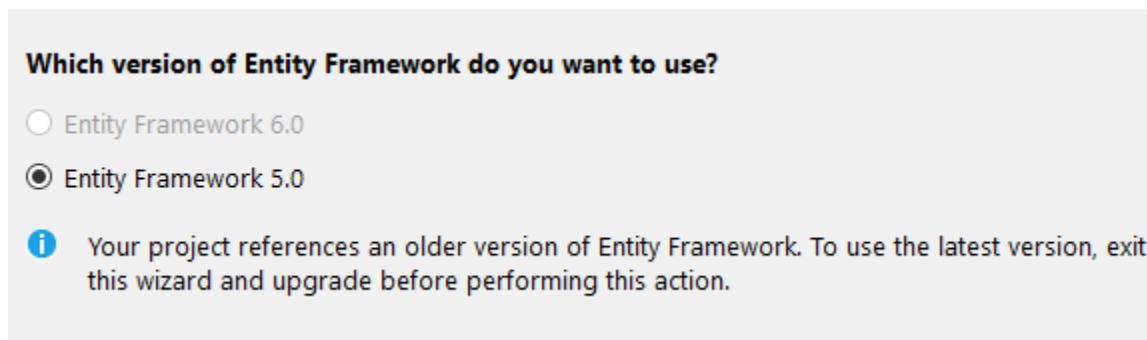
Khởi tạo Entity Data Model

- **Bước 6:** Chọn Server Name

Chọn Database → OK

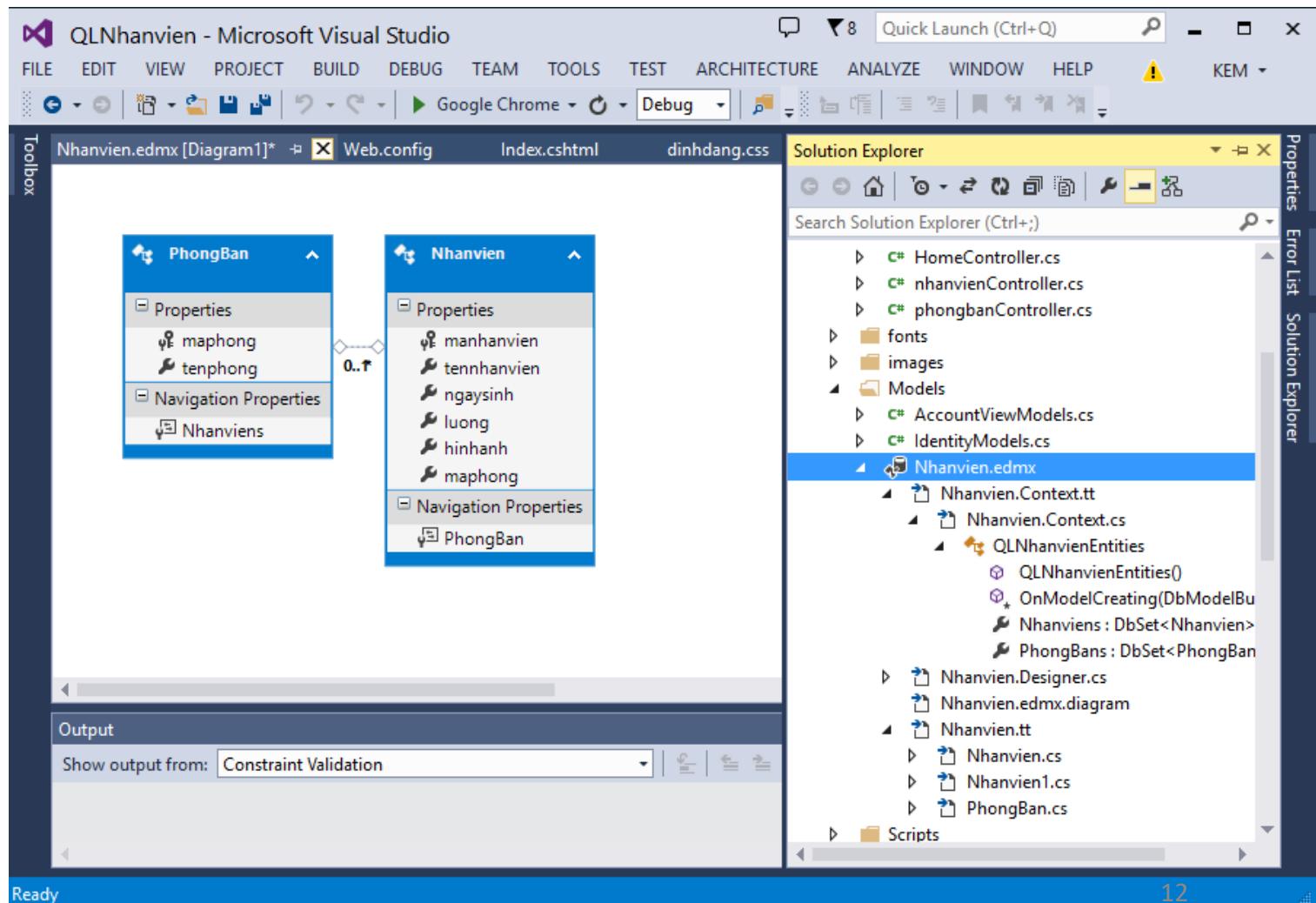
- **Bước 7:** Trong bước này hiển thị tất cả các bảng, view, store procedures trong cơ sở dữ liệu. Chọn các loại và click Finish.

- Chọn phiên bản cho Entity Framework



Khởi tạo Entity Data Model:

Bước 8: Sau khi click nút Finish, file xuất hiện trong cửa sổ solution explorer. **Nhanvien.edmx** hiển thị tất cả thực thể và quan hệ trong cơ sở dữ liệu.



- 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.

- **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.

```
Nhanvien.Context.cs ✘ X Nhanvien.edmx [Diagram1]* Web.config Index.cshtml dinhdang.css _Layout.cshtml Index.cshtml
QLNhanvien.Models.QLNhanvienEntities OnModelCreating(DbModelBuilder modelBuilder)
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; }
}

100 %
```

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

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

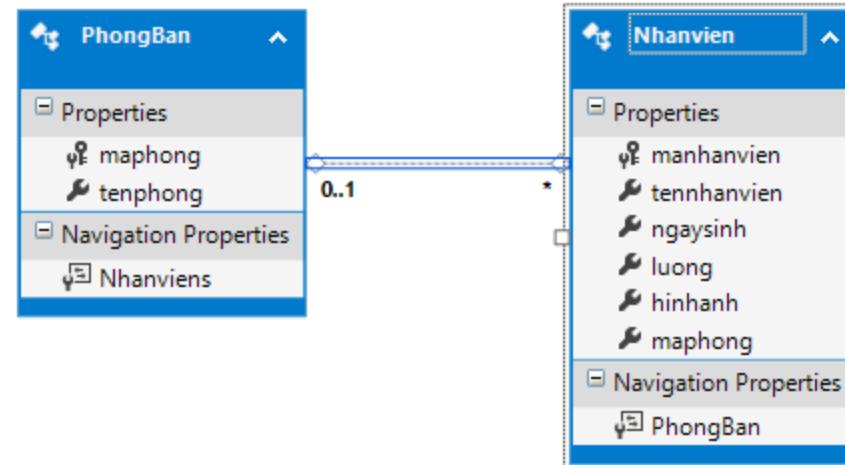
Ví dụ: Lớp context bao gồm tập thực thể kiểu DbSet< TEntity > cho các thực thể.

```
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

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 | <code>db.Nhanviens.ToList()</code> |
| Add | Kiểu thực thể được thêm vào | <code>db.Nhanviens.Add(nv);</code> |
| Find | Tìm kiếm một thực thể theo khóa chính <code>Nhanvien nv = db.Nhanviens.Find(id);</code> | |
| Remove | Xóa thực thể | <code>db.Nhanviens.Remove(nv);</code> |
| SaveChange() | DbContext lưu lại sự thay đổi của các thực thể trong từng Dbset như Remove, Add, ... | <code>db.SaveChanges();</code> |

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

1. 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());
}
```

2. 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);
}
```

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

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

4. 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")]
0 references
public ActionResult DeleteConfirm(int id)
{
    id = Convert.ToInt32(id);
    Nhanvien nv = db.Nhanviens.Find(id);
    db.Nhanviens.Remove(nv);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

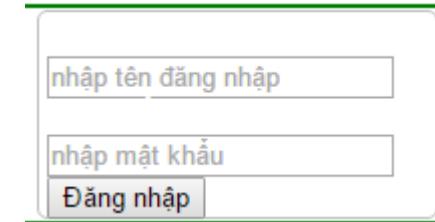
5. Hiển thị chi tiết thông tin của nhân viên: Tìm nhân viên theo id

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

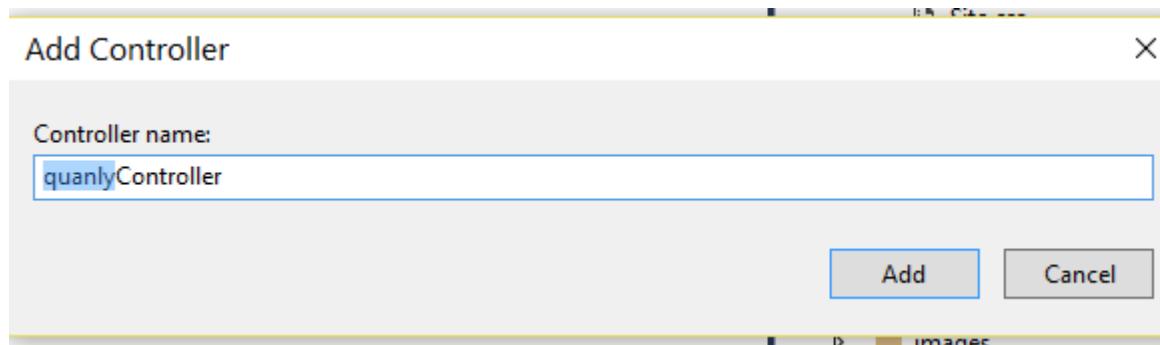
PartialView

- 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>
```



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



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

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);
}
```

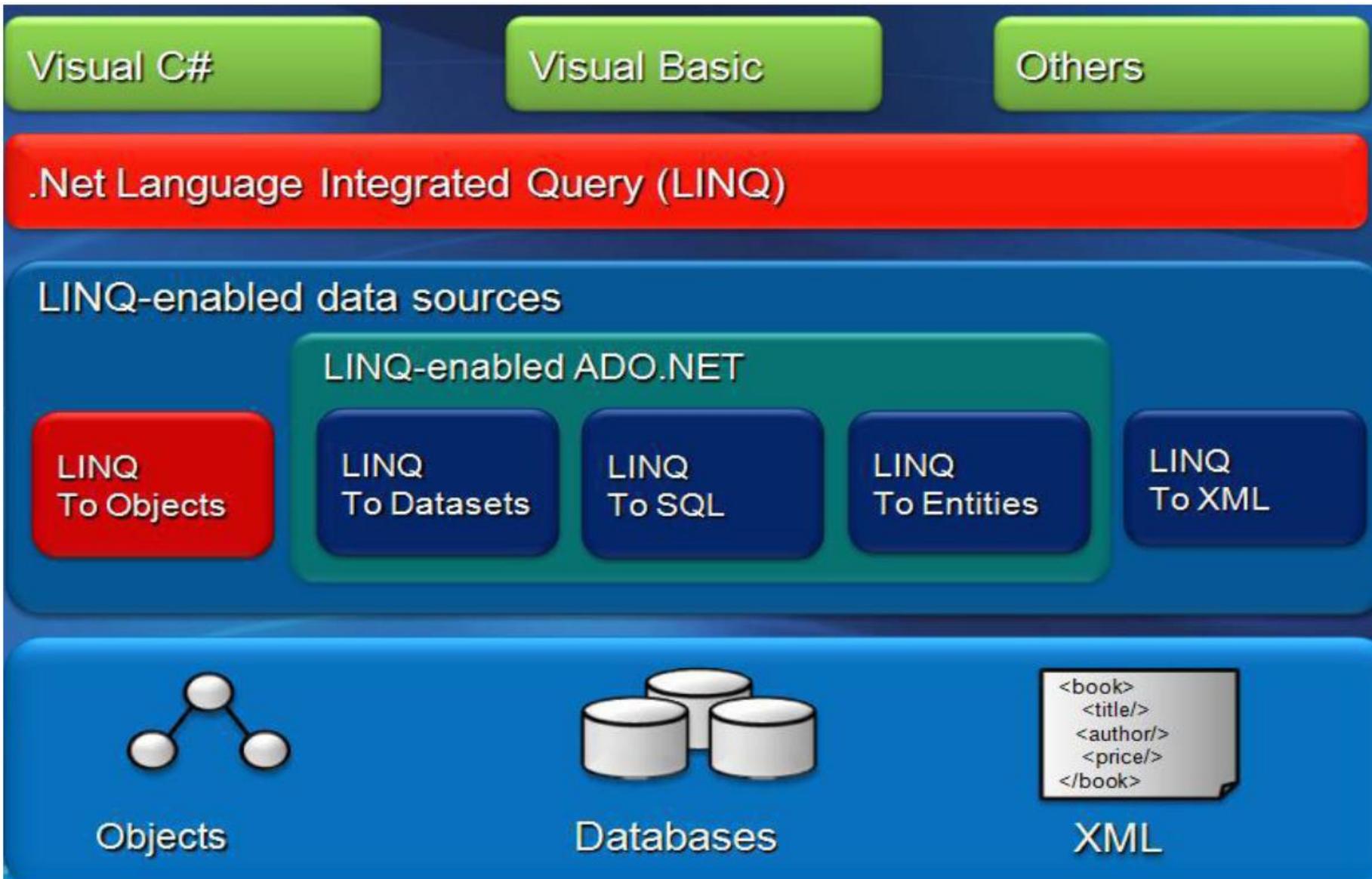
- Tạo hàm **CheckUser(string username, string password)** kiểm tra username và password từ bảng Account của EDM:

```
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;
    }
}
```

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 Redirect(strURL);
}
```

Kiến trúc LINQ



Truy vấn dữ liệu:

```
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 naked 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

Ví dụ 1: Truy vấn số chẵn:

```
int[] numbers = { 19, 23, 6, 56, 45, 87, 5, 8, 13 };
```

```
var evens = from n in numbers  
            where n % 2 == 0  
            select n;
```

```
foreach(int n in numbers){  
    if(n % 2 == 0){  
        tích lũy số chẵn  
    }  
}
```

- ❑ **from**: chỉ ra phần tử được lấy từ tập hợp cần truy vấn
- ❑ **where**: chỉ ra điều kiện lọc
- ❑ **select**: chỉ ra đối tượng nhận được

Ví dụ 2: Truy vấn số chẵn:

```
int[] numbers = { 19, 23, 6, 56, 45, 87, 5, 8, 13 };

var evens = from n in numbers
    where n % 2 == 0
    let rate = n / numbers.Sum()
    orderby n descending
    select new { number = n, rate = rate };

foreach (var e in evens)
{
    int n = e.N
}
```

Đối tượng

The screenshot shows an IDE interface with code in C#. An orange box highlights the 'select' statement. A callout box labeled 'Đối tượng' (Object) points to the 'new { ... }' part of the 'select' statement. Below it, another orange box highlights the 'N' in 'e.N'. An Intellisense dropdown is open over the 'N', listing properties: N, Equals, GetHashCode, GetType, number, rate, and ToString. The 'number' item is selected and highlighted in blue. To its right, a tooltip displays the type information: 'int 'a.number'' and 'Anonymous Types: 'a is new { int number, int rate }''. The background shows the rest of the code with some syntax highlighting.

Tổng hợp, thống kê:

```
int[] numbers = { 19, 23, 6, 56, 45, 87, 5, 8, 13 };

var evens = from n in numbers
            group n by n % 3 into g
            select new
            {
                Nhóm = g.Key,
                Tổng = g.Sum(),
                SốLượng = g.Count(),
                SốNN = g.Min(),
                SốLN = g.Max(),
                SốTB = g.Average()
            };

```

- ❑ Nhóm chia 3 dư 0: gồm 6, 45, 87
- ❑ Nhóm chia 3 dư 1: gồm 19, 13
- ❑ Nhóm chia 3 dư 2: gồm 23, 56, 5, 8

Sử dụng phương thức mở rộng:

```
var evens = numbers
    .Where(n => n % 2 == 0)
    .Select(n => n);
```

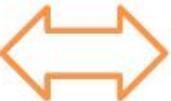
```
var evens = numbers
    .Where(n => n % 2 == 0)
    .OrderByDescending(n => n)
    .Select(n => new
    {
        number = n,
        rate = n / numbers.Sum()
    });

```

```
var evens = numbers.GroupBy(n => n % 3)
    .Select(g => new
    {
        Nhóm = g.Key,
        Tổng=g.Sum(),
        SốLượng=g.Count(),
        SốNN=g.Min(),
        SốLN=g.Max(),
        SốTB=g.Average()
    });

```

```
var evens = from n in numbers  
            where n % 2 == 0  
            select n;
```



```
var evens = numbers  
            .Where(n => n % 2 == 0)  
            .Select(n => n);
```

```
var evens = from n in numbers  
            where n % 2 == 0  
            orderby n descending  
            select new  
            {  
                number = n,  
                rate = n / numbers.Sum()  
            };
```



```
var evens = numbers  
            .Where(n => n % 2 == 0)  
            .OrderByDescending(n => n)  
            .Select(n => new  
            {  
                number = n,  
                rate = n / numbers.Sum()  
            });
```

```
var evens = from n in numbers  
            group n by n % 3 into g  
            select new  
            {  
                Nhom = g.Key,  
                Tong = g.Sum(),  
                SoLuong = g.Count(),  
                SoNN = g.Min(),  
                SoLN = g.Max(),  
                SoTB = g.Average()  
            };
```



```
var evens = numbers.GroupBy(n => n % 3)  
            .Select(g => new  
            {  
                Nhom = g.Key,  
                Tong=g.Sum(),  
                SoLuong=g.Count(),  
                SoNN=g.Min(),  
                SoLN=g.Max(),  
                SoTB=g.Average()  
            });
```

Truy vấn cơ bản:

| Phương thức | Mô tả | Ví dụ |
|----------------------------------|------------------------------------|--|
| .Where(e=>điều kiện) | Lọc | Students.Where(s=>s.Marks > 9) |
| .GroupBy(e=>biểu thức) | Nhóm | Students.GroupBy(s=>s.Clazz) |
| .OrderBy(e=>biểu thức) | Sắp xếp | Students.OrderBy(s=>s.Name) |
| .OrderByDescending(e=>biểu thức) | | |
| .Select(e=>đối tượng) | Chọn | Students.Select(s=>new{s.Name, s.Marks}) |
| .Distinct() | Giữ 1 của các đối tượng giống nhau | Numbers.Distinct() |

var studs = Students

 .Where(s=>s.Marks > 9)

 .OrderBy(s=>s.Marks)

 .Select(s=>s);

Truy vấn phân trang:

| Phương thức | Mô tả | Ví dụ |
|-------------------------------------|---------------------------------------|---|
| .Take(số lượng) | Lấy các phần tử đầu | Students.Take(5) |
| .Skip(số lượng) | Bỏ qua các phần tử đầu | Students.Skip(3).Take(6) |
| .TakeWhile($e \Rightarrow$ đ.kiện) | Lấy các phần tử đầu thỏa điều kiện | Students.TakeWhile($s \Rightarrow s.Marks < 4$) |
| .SkipWhile($e \Rightarrow$ đ.kiện) | Bỏ qua các phần tử đầu thỏa điều kiện | Students.SkipWhile($s \Rightarrow s.Marks < 0$) |

```
var result = db.Products  
    .Skip(10).Take(20)
```

Truy vấn một thực thể:

```
var result = db.Customers  
    .Single(c=>c.Id=="A" && c.Password=="B")
```

| Phương thức | Mô tả | Ví dụ |
|--------------------|---|---------------------------------|
| .Single(e=>đ.kiện) | Lấy 1 phần tử thỏa điều kiện. Ngoại lệ nếu không tìm thấy hoặc nhiều hơn một. | Students.Single(s=>s.Id=="Hoa") |
| .First() | Lấy phần tử đầu | Students.First() |
| .Last() | Lấy phần tử cuối | Students.Last() |

Tổng hợp số liệu:

| Phương thức | Mô tả | Ví dụ |
|-------------------------------|--------------------|------------------------------|
| .Sum(e=>biểu thức số học) | Tính tổng | Students.Sum(s=>s.Marks) |
| .Count(e=>biểu thức số học) | Đếm số lượng | Students.Count(s=>s.Id) |
| .Min(e=>biểu thức số học) | Giá trị nhỏ nhất | Students.Min(s=>s.Marks) |
| .Max(e=>biểu thức số học) | Giá trị lớn nhất | Students.Max(s=>s.Marks) |
| .Average(e=>biểu thức số học) | Giá trị trung bình | Students.Average(s=>s.Marks) |

❑ Var result = db.Products

 ↳ .GroupBy(p=>p.Category)

 ↳ .Select(g=>new{g.Key.Name, g.Count})

Ví dụ, Thống kê doanh số:

```
var items7 = db.Products.GroupBy(p => p.Category)
    .Select(g => new ReportInfo
    {
        Group = g.Key.Name, //--tên loại
        Sum = g.Sum(p=>p.UnitPrice), //--tổng đơn giá hàng hóa của loại
        Count = g.Count(), //--số hàng hóa của loại
        Min = g.Min(p => p.UnitPrice), //--giá hàng hóa thấp nhất
        Max = g.Max(p => p.UnitPrice), //--giá hàng hóa cao nhất
        Avg = g.Average(p => p.UnitPrice) //--giá trung bình
    });

```

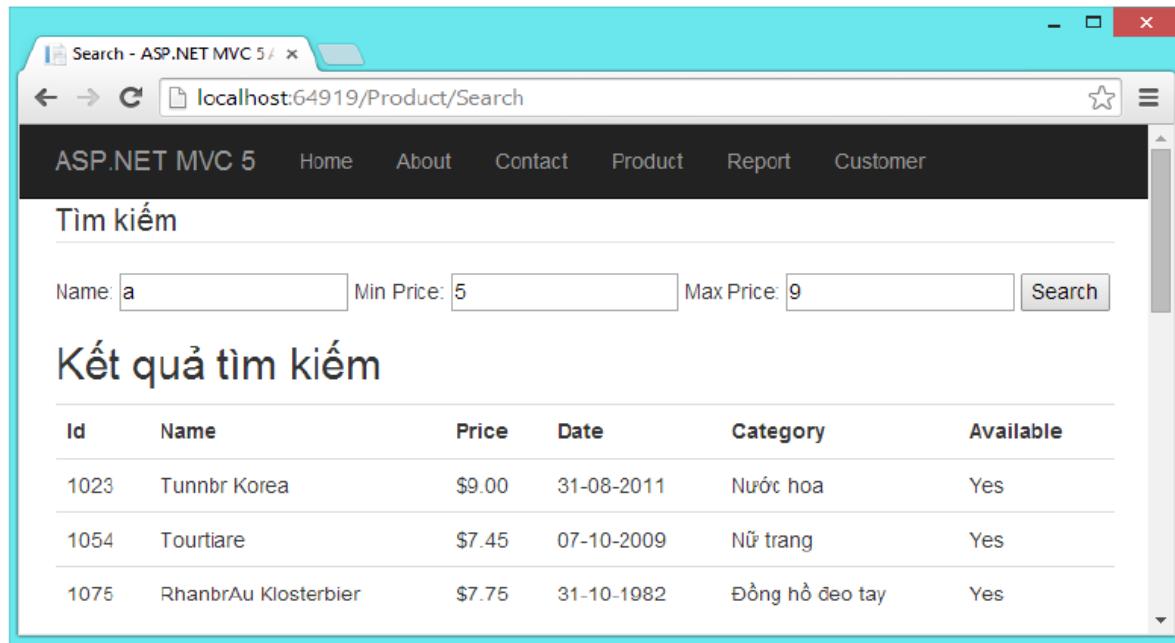
```
var items8 = db.OrderDetails.GroupBy(d=>d.Product)
    .Select(g => new ReportInfo
    {
        Group = g.Key.Name, //--tên hàng hóa
        Sum = g.Sum(p => p.UnitPrice * p.Quantity), //--tổng giá trị đã bán
        Count = g.Sum(p => p.Quantity), //--tổng số lượng đã bán
        Min = g.Min(p => p.UnitPrice), //--giá thấp nhất
        Max = g.Max(p => p.UnitPrice), //--giá cao nhất
        Avg = g.Average(p => p.UnitPrice) //--giá trung bình
    });

```

```
var items9 = db.OrderDetails.GroupBy(d => d.Product.Category)
    .Select(g => new ReportInfo
{
    Group = g.Key.Name, //--tên loại hàng
    Sum = g.Sum(p => p.UnitPrice * p.Quantity), //--tổng giá trị hàng hóa đã bán
    Count = g.Sum(p=>p.Quantity), //--tổng số lượng đã bán
    Min = g.Min(p => p.UnitPrice), //--giá thấp nhất
    Max = g.Max(p => p.UnitPrice), //--giá cao nhất
    Avg = g.Average(p => p.UnitPrice) //--giá trung bình
});
```

```
var items10 = db.OrderDetails.GroupBy(d => d.Order.Customer)
    .Select(g => new ReportInfo
{
    Group = g.Key.Fullname, //--họ và tên khách hàng
    Sum = g.Sum(p => p.UnitPrice * p.Quantity), //--tổng giá trị hàng hóa đã mua
    Count = g.Sum(p=>p.Quantity), //--tổng số lượng đã mua
    Min = g.Min(p => p.UnitPrice), //--giá thấp nhất
    Max = g.Max(p => p.UnitPrice), //--giá cao nhất
    Avg = g.Average(p => p.UnitPrice) //--giá trung bình
});
```

Ví dụ, Lọc dữ liệu:



```
public ActionResult Search(String Name = "",  
                           double Min = double.MinValue, double Max = double.MaxValue)  
{  
    var list = db.Products  
        .Where(p => p.Name.Contains(Name) && p.UnitPrice >= Min  
                                         && p.UnitPrice <= Max).ToList();  
    return View(list);  
}
```

DATA ANNOTATIONS & VALIDATION

1. 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.
- 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.

a. Thuộc tính hiện thị

- Cú pháp: [DisplayName("tên thuộc tính mới")]

b. Thuộc tính xác nhận dữ liệu

- *Timestamp*: Thuộc tính này dùng để tạo ra một cột dữ liệu thời gian trong database SQL khi cập nhật dữ liệu.
- *Required*: chỉ định thuộc tính phải có dữ liệu nhập vào trước khi submit về server.
- *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.

c. 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.

Ví dụ:

```
[Table("DocGia")]
public partial class DocGia
{
    [Key]
    [Required(ErrorMessage = "Chưa nhập số thẻ")]
    [StringLength(10)]
    [DisplayName("Mã nhân viên")]
    public string SoThe { get; set; }
    [Required(ErrorMessage = "Chưa nhập họ độc giả")]
    [StringLength(50)]
    public string HoDG { get; set; }
    [Required(ErrorMessage = "Chưa nhập tên độc")]
    [StringLength(10)]
    public string TenDG { get; set; }
    ...
}
```

2. Validation

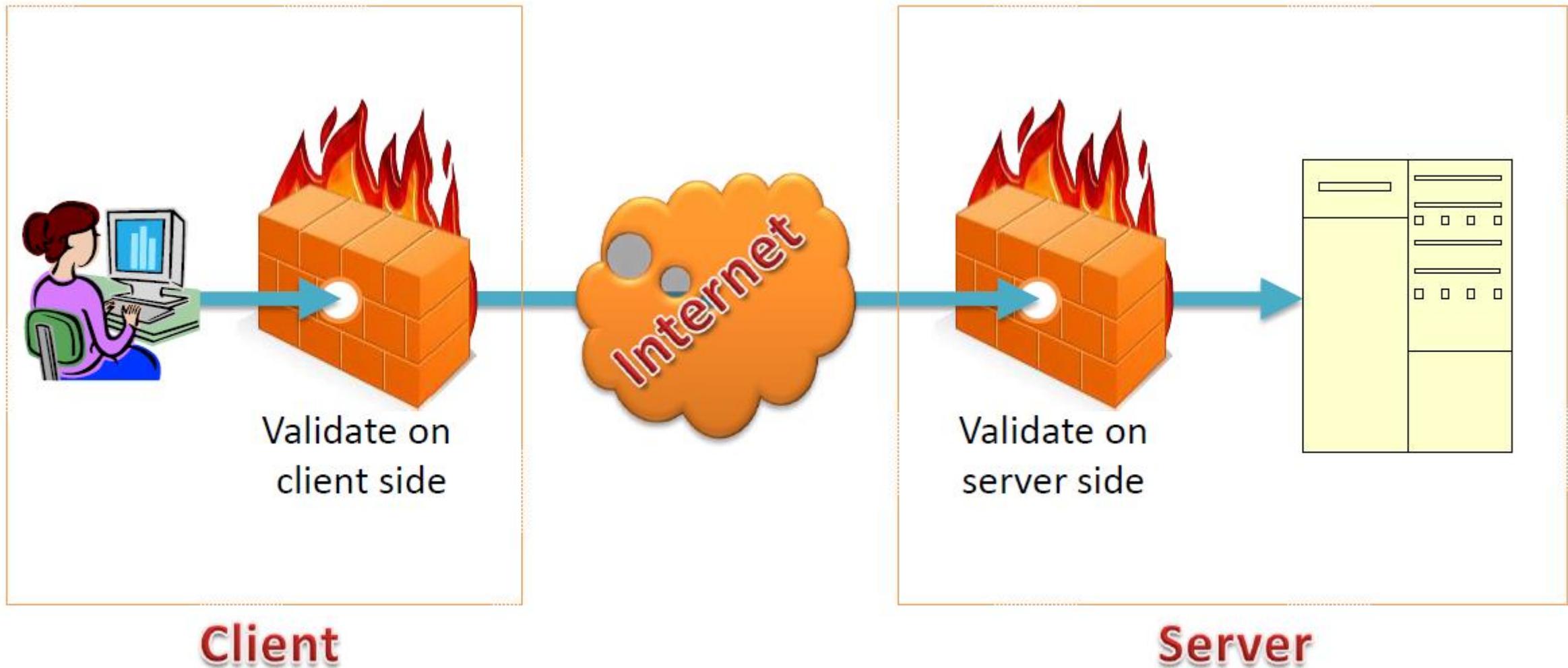
- ❑ Dữ liệu không hợp lệ nhập từ người sử dụng sẽ gây các lỗi khó lường.
- ❑ Vì vậy việc kiểm soát dữ liệu vào luôn đóng vai trò quan trọng.
- ❑ Các lỗi thường gặp
 - ☞ Để trống ô nhập...
 - ☞ Không đúng định dạng email, creditcard, url...
 - ☞ Sai kiểu số nguyên, số thực, ngày giờ...
 - ☞ Không hợp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

Ví dụ:

➊ ĐĂNG KÝ THÀNH VIÊN

| | | |
|-------------------------|---|-----------------------------------|
| Tên đăng nhập : | <input type="text" value="nnghiem"/> | Mã này đã được sử dụng. |
| Mật khẩu : | <input type="password"/> | Trường bắt buộc. |
| Nhập lại mật khẩu mới : | <input type="password" value="abc"/> | Giá trị nhập không giống. |
| Họ và tên : | <input type="text"/> | Trường bắt buộc. |
| Giới tính : | <input checked="" type="radio"/> Nam <input type="radio"/> Nữ | |
| Thư điện tử : | <input type="text" value="abc"/> | Không đúng dạng email. |
| Điện thoại di động : | <input type="text"/> | |
| Ngày sinh : | <input type="text"/> | Trường bắt buộc. |
| Địa chỉ : | <input type="text"/> | |
| Hình ảnh : | <input type="text" value="C:\NetworkCfg.xml"/> <input type="button" value="Browse..."/> | Không chấp nhận loại tập tin này. |
| Mã bảo mật : | <input type="text" value="AS"/> | Sai mã bảo mật. AEBC44 |

Mô hình kiểm lỗi:



Kiểm lỗi trong MVC:

- Kiểm soát dữ liệu có thể thực hiện cả 2 phía là client và server:
 - Kiểm lỗi phía client sẽ phản ứng nhanh cho người sử dụng để có thể sửa chữa ngay.
 - Kiểm lỗi phía server sẽ thực hiện các lỗi mà client không thể làm được nếu dữ liệu có liên quan đến tài nguyên server.
 - Với MVC chỉ cần viết 1 lần nhưng kiểm tra cả 2 phía là client và server. Nếu vì một lý do nào đó mà client không thực hiện được thì đã có server thay thế.

❑ Mã trên Model

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

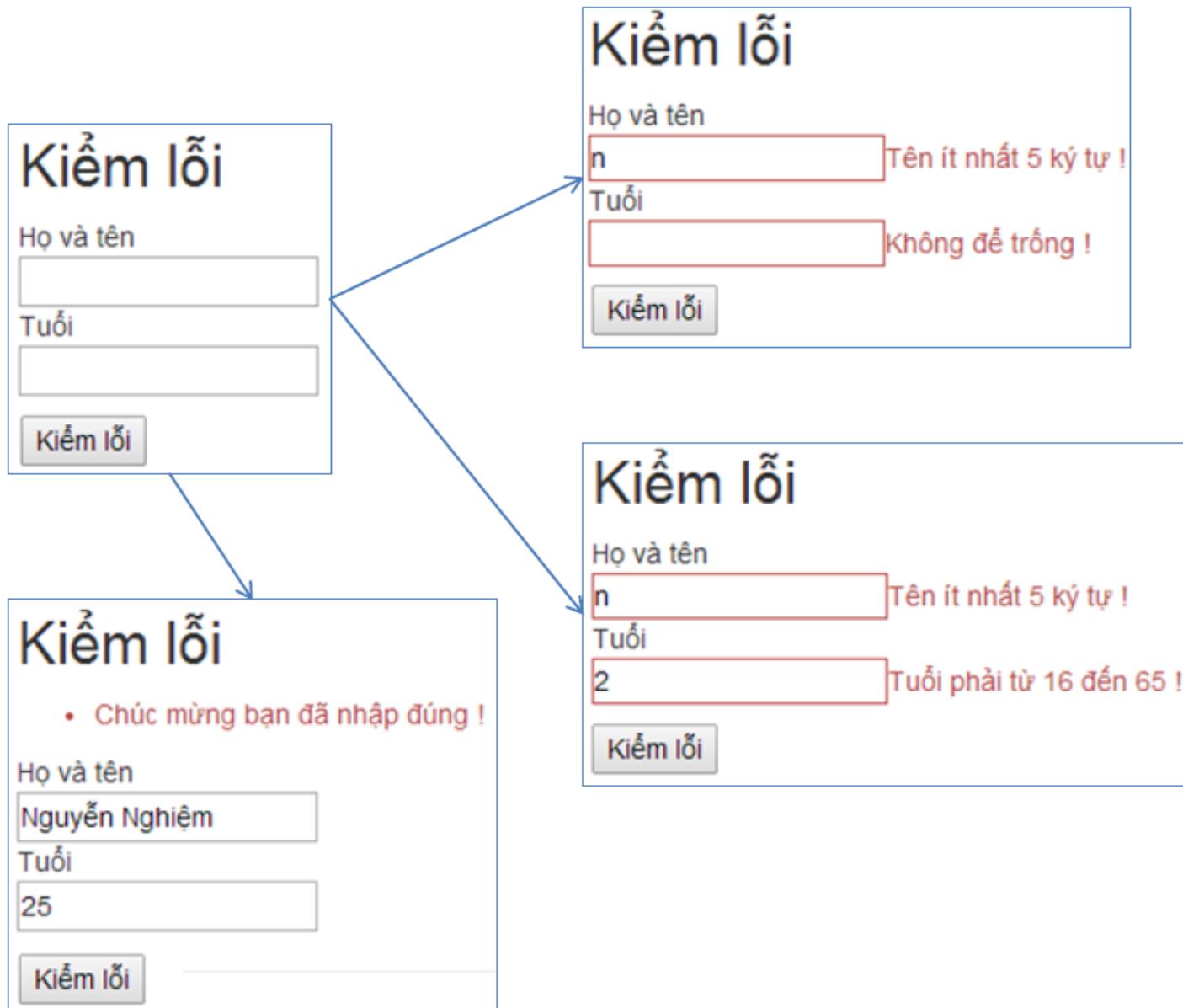
❑ Mã trên View

- 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()

Ví dụ:



Model?
Controller?
View?

Lớp Model:

```
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 |

Controller:

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

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

Kiểm lỗi phía server

Kiểm lỗi

Họ và tên

Tuổi

Kiểm lỗi

Bổ sung thông báo lỗi model

The diagram illustrates the flow of validation logic. It starts with the 'ValidatorController' code, which contains two actions: 'Index' and 'Validate'. The 'Index' action returns a view. The 'Validate' action takes an 'EmployeeInfo' model and checks if it is valid. If not, it adds an error message to the ModelState. Both the 'Validate' action and the 'Index' action are connected by arrows to a central validation form titled 'Kiểm lỗi'. This form has fields for 'Họ và tên' (Name) and 'Tuổi' (Age), and a 'Kiểm lỗi' (Check) button. A callout box labeled 'Kiểm lỗi phía server' (Server-side validation) points to the 'Validate' action. Another callout box labeled 'Bổ sung thông báo lỗi model' (Add model error message) points to the line of code where an error is added to the ModelState.

View:

```
@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" />  
}
```

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

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

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;} |

The HTML 5:

□ [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 lập trình:

```
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");
}
```

Thuộc tính 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}

Chống các yêu cầu giả mạo:

- ☐ Bổ sung **@Html.AntiForgeryToken()** vào form để tránh các request giả mạo

```
@using (Html.BeginForm("Withdraw", "Bank")) {  
    @Html.AntiForgeryToken()  
    <fieldset>  
        <legend>Fields</legend>  
        <p>  
            <label for="Amount">Amount:</label>  
            @Html.TextBox("Amount")  
        </p>  
        <p>  
            <input type="submit" value="Withdraw" />  
        </p>  
    </fieldset>  
}
```

~~~~  
~~<link href="http://.../Bank/Withdraw?Amount=9999">~~