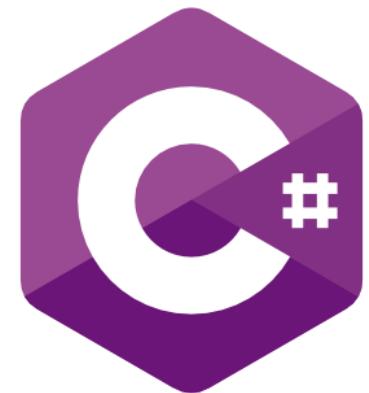


# C# 15 – Khái niệm nâng cao

Giảng viên: **ThS. Lê Thiện Nhật Quang**  
Email: [quangltn.dotnet.vn@gmail.com](mailto:quangltn.dotnet.vn@gmail.com)  
Website: <http://dotnet.edu.vn>  
Điện thoại: **0868.917.786**



# MỤC TIÊU

- Tìm hiểu định nghĩa hệ thống generic delegate.
- Định nghĩa biểu thức lambda.
- Giải thích biểu thức truy vấn – query.
- Tìm hiểu mô hình phát triển ứng dụng hướng dịch vụ (WCF).
- Giải thích lập trình parallel.
- Giải thích lập trình dynamic.

## 1.1. ĐỊNH NGHĨA GENERIC DELEGATE

**Delegate** (*hàm ủy quyền*) là một kiểu dữ liệu, nó dùng để tham chiếu (trỏ đến) đến các hàm (phương thức) có tham số và kiểu trả về phù hợp với khai báo kiểu. Khi dùng đến **delegate** bạn có thể gán vào nó một, nhiều hàm (phương thức) có sự tương thích về tham số, kiểu trả về, sau đó dùng nó để gọi hàm (*giống con trỏ trong C++*), các **event** trong C# chính là các hàm được gọi thông qua **delegate**, bạn cũng có thể dùng delegate để xây dựng các hàm callback, đặc biệt là các **Event Delegate** được dùng phổ biến khi gán các biểu thức **lambda**

## 1.2. FUNC VÀ ACTION TRONG GENERIC DELEGATE

Func và Action là hai mẫu delegate định nghĩa sẵn, giúp bạn nhanh chóng tạo ra biến kiểu delegate mà không mất công khai báo.

```
public delegate void ShowLog(string message);
```

## 1.2. FUNC VÀ ACTION TRONG GENERIC DELEGATE (2)

### Sử dụng Func

Func là mẫu delegate có kiểu trả về. Để khai báo biến delegate dùng cú pháp như sau:

*Func<kiểu\_tham\_số\_1, kiểu\_tham\_số\_2, ..., kiểu\_trả\_về> var\_delegate;*

Kiểu cuối cùng trong khai báo Func là kiểu trả về của hàm, có thể thiếu tham số nhưng không được thiếu kiểu trả về

Ví dụ muốn có biến delegate tên **bien1** tương đương với hàm có 2 tham số, tham số 1 kiểu int, tham số 2 kiểu string, và hàm trả về kiểu bool thì tạo biến đó như sau:

*Func<int, string, bool> bien1;*

```
// Khai báo delegate ở lớp
delegate bool DelegateName(int a, string b);
// Khai báo biến trong phương thức
DelegateName bien1;
```

## 1.2. FUNC VÀ ACTION TRONG GENERIC DELEGATE (3)

```
class FuncAction
{
    static int Sum(int x, int y)
    {
        return x + y;
    }

    public static void TestFunc(int x, int y)
    {
        Func<int,int,int> tinh tong;           // biến tinh tong phù hợp với các hàm trả về kiểu int, có 2 tham số kiểu int
        tinh = Sum;                            // Hàm Sum phù hợp nên có thể gán cho biến

        var ketqua = tinh(x, y);
        Console.WriteLine(ketqua);
    }
}
```

```
FuncAction.TestFunc(5, 6); // In ra: 11
```

## 1.2. FUNC VÀ ACTION TRONG GENERIC DELEGATE (4)

### Sử dụng Action

Action tương tự như Func, điều khác duy nhất là không có kiểu trả về.

Khai báo cơ bản như sau:

Action<kiểu\_tham\_số\_1, kiểu\_tham\_số\_2, ...> var\_delegate;

Nghĩa là biến kiểu Action có thể gán bằng các hàm có kiểu trả về void

```
public static void TestAction(string s)
{
    Action<string> showLog = null;

    showLog += Logs.Warning;           // Nối thêm Warning vào delegate
    showLog += Logs.Info;              // Nối thêm Info vào delegate
    showLog += Logs.Warning;           // Nối thêm Warning vào delegate

    // Một lần gọi thi hành tất cả các phương thức trong chuỗi delegate
    showLog("TestLog");
}
```

```
public class L
{
    // Khai báo
    public delegate void Log(string message);

    // Phương thức
    static public void ShowLogs(Log log)
    {
        Console.WriteLine(log("Warning"));
        Console.WriteLine(log("Info"));
        Console.WriteLine(log("Warning"));
    }
}

// Phương thức
static public void ShowLogs(Log log)
{
    Console.WriteLine(log("Warning"));
    Console.WriteLine(log("Info"));
    Console.WriteLine(log("Warning"));
}

public static void TestAction(string s)
{
    Action<string> showLog = null;

    showLog += Logs.Warning;           // Nối thêm Warning vào delegate
    showLog += Logs.Info;              // Nối thêm Info vào delegate
    showLog += Logs.Warning;           // Nối thêm Warning vào delegate

    // Một lần gọi thi hành tất cả các phương thức trong chuỗi delegate
    showLog("TestLog");
}
```

## 2.1. BIỂU THỨC LAMBDA

Biểu thức **lambda** còn gọi là biểu thức **hàm nặc danh** (Anonymous), một biểu thức khai báo giống phương thức (hàm) nhưng thiếu tên. Cú pháp để khai báo biểu thức **lambda** là sử dụng toán tử =>

(các\_tham\_số) => biểu\_thức;

Thậm chí là cả một cấu trúc lệnh sau toán tử =>

(các\_tham\_số) => { // các câu lệnh // Sử dụng return nếu có giá trị trả về }

Các biểu thức **lambda** đều có thể chuyển đổi thành delegate, do vậy nó có thể gán cho các delegate phù hợp.

## 2.2. SỬ DỤNG LAMBDA

Ví dụ, đây là một biểu thức **lambda**, nó tính tổng của hai số nguyên

(int x, int y) => { **return** x + y; };

Có thể gán biểu thức này cho biến delegate.

```
class Program {
    public delegate int TinhToan (int a, int b);
    static void Main (string[] args) {

        // Gán biểu thức lambda cho delegate
        TinhToan tinhTong = (int x, int y) => {
            return x + y;
        };

        int kq = tinhTong (5, 1); // kq = 6;
        Console.WriteLine(kq);
    }
}
```

## 2.2. SỬ DỤNG LAMBDA (2)

Như phần trình bày về Delegate, có thể gán biểu thức lambda có kết quả trả về cho **Func** hoặc biểu thức không có kết quả trả về cho **Action**

```
class Program {
    public delegate int TinhToan (int a, int b);
    static void Main (string[] args) {

        //Gán lambda cho Func
        Func<int, int, int> tinh tong1 = (int x, int y) => {
            return x + y;
        };
        // Gán lambda cho Action
        Action<int> thongbao = (int vl) => {
            Console.WriteLine (vl);
        };

        int kq1 = tinh tong1 (5, 3); // kq1 = 8
        int kq2 = tinh tong1 (5, 5); // kq2 = 10
        thongbao (kq1); // In ra: 8
        thongbao (kq2); // In ra: 10
    }
}
```

## 2.3. ĐỊNH NGHĨA PHƯƠNG THỨC VỚI LAMBDA

Bạn có thể sử dụng toán tử `=>` sau khai báo tên phương thức (loại phương thức có kiểu trả về khác void) của lớp, rồi đến ngay một biểu thức có kết quả trả về (không dùng `{ }` ), biểu thức này như là định nghĩa thân của phương thức.

Ví dụ, khai báo phương thức:

`int Tong(int x, int y) => x + y;`

Nó tương đương với dạng đầy đủ đã biết

`int Tong(int x, int y) { return x + y; }`

### 3.1. BIỂU THÚC TRUY VẤN – QUERY

**LINQ (Language Integrated Query)** - ngôn ngữ truy vấn tích hợp - nó tích hợp cú pháp truy vấn (gần giống các câu lệnh SQL) vào bên trong ngôn ngữ lập trình C#, cho nó C# khả năng truy cập các nguồn dữ liệu khác nhau (SQL Db, XML, List ...) với *cùng cú pháp*.

Nguồn dữ liệu phục vụ cho LINQ, phải là các đối tượng lớp triển khai giao diện (interface) **IEnumerable** và **IEnumerable<T>** tức là các mảng, danh sách thuộc Collection đã biết ở phần trước.

## 3.2. VÍ DỤ BIỂU THỨC TRUY VẤN

Tạo ra một nguồn dữ liệu đó là một danh sách (sẽ dùng lớp **List**) các sản phẩm (các sản phẩm sẽ sử dụng lớp **Product** tự xây dựng). Xây dựng lớp **Product** như sau:

```
public class Product
{
    public int ID {set; get;}
    public string Name {set; get;}          // tên
    public double Price {set; get;}         // giá
    public string[] Colors {set; get;}       // các màu sắc
    public int Brand {set; get;}            // ID Nhãn hiệu, hãng
    public Product(int id, string name, double price, string[] colors, int brand)
    {
        ID = id; Name = name; Price = price; Colors = colors; Brand = brand;
    }
    // Lấy chuỗi thông tin sản phẩm gồm ID, Name, Price
    override public string ToString()
        => $"{ID,3} {Name, 12} {Price, 5} {Brand, 2} {string.Join(",", Colors)}";
}
```

## 3.2. VÍ DỤ BIỂU THỨC TRUY VẤN (2)

Xây dựng lớp **Brand** để biểu diễn nhãn hiệu hàng hóa:

```
public class Brand {  
    public string Name {set; get;}  
    public int ID {set; get;}  
}
```

## 3.2. VÍ DỤ BIỂU THỨC TRUY VẤN (3)

Khởi tạo ra 2 đối tượng danh sách dùng để thực hiện các truy vấn linq: danh sách sản phẩm **products**, danh sách nhãn hiệu **brands**

```
var brands = new List<Brand>()
{
    new Brand{ID = 1, Name = "Công ty AAA"},
    new Brand{ID = 2, Name = "Công ty BBB"},
    new Brand{ID = 4, Name = "Công ty CCC"},
};

var products = new List<Product>()
{
    new Product(1, "Bàn trà",      400, new string[] {"Xám", "Xanh"},      2),
    new Product(2, "Tranh treo",   400, new string[] {"Vàng", "Xanh"},      1),
    new Product(3, "Đèn trùm",     500, new string[] {"Trắng"},            3),
    new Product(4, "Bàn học",      200, new string[] {"Trắng", "Xanh"},     1),
    new Product(5, "Túi da",       300, new string[] {"Đỏ", "Đen", "Vàng"}, 2),
    new Product(6, "Giường ngủ",   500, new string[] {"Trắng"},            2),
    new Product(7, "Tủ áo",        600, new string[] {"Trắng"},            3),
};
```

### 3.3. VIẾT CÂU TRUY VẤN ĐẦU TIÊN

Lọc ra những sản phẩm có giá bằng 400

```
public class Products
{
    // ...
    // In ra các sản phẩm có giá 400
    public static void ProductPrice400()
    {
        // Viết câu quy vấn, lưu vào ketqua
        var ketqua = from product in products
                     where product.Price == 400
                     select product;

        foreach (var product in ketqua)
            Console.WriteLine(product.ToString());
    }
    // ...
}
```

Trong hàm Main chạy thử:

```
Products.ProductPrice400();

// Kết quả
// ID 3 - Bàn trà, giá 400
// ID 4 - Tranh treo, giá 400
```

Câu truy vấn LINQ thường bắt đầu bằng mệnh đề **from** và kết thúc bằng mệnh đề **select** hoặc **group**, giữa chúng là những mệnh đề **where**, **orderby**, **join**, **let**

### 3.4. MỆNH ĐỀ FROM...IN...

Mệnh đề **from** để xác định nguồn dữ liệu mà truy vấn sẽ thực hiện, nguồn dữ liệu là tập hợp những phần tử lưu trong đối tượng có kiểu lớp triển khai giao diện **IEnumerable** như mảng Array, List ...

Ví dụ, **products** ở trên là kiểu List (*nó triển khai IEnumerable*), thì một giải (một đoạn, hoặc tất cả) các phần tử trong nó (ví dụ **product in products**) có thể làm nguồn, vậy mở đầu truy vấn bằng mệnh đề **from** sẽ là:

**from product in products**

- *products* là nguồn dữ liệu
- *product* là tên tùy đặt đại diện cho phần tử trong nguồn, tên này sẽ được dùng bởi các mệnh đề sau.

### 3.5. MỆNH ĐỀ SELECT

Một câu truy vấn phải kết thúc bằng mệnh đề **select** hoặc **group**. Mệnh đề **select** chỉ ra các dữ liệu được lấy ra (xuất ra) của câu lệnh LINQ.

Ví dụ:

```
var ketqua = from product in products select product;
```

Có nghĩa là với mỗi *product* trong *products*, dữ liệu lấy ra là các *product* đó (trả về collection gồm các phần tử *Product*).

### 3.5. MỆNH ĐỀ SELECT (2)

Bạn có thể chỉ lấy ra một loại dữ liệu nào đó của phần tử , ví dụ:

```
var ketqua = from product in products select product.Name;
```

```
foreach (var name in ketqua)
```

```
Console.WriteLine(name); // Bàn trà // Túi da //...
```

Câu LINQ trên có nghĩa là với mỗi product có trong products lấy ra tên (Name) của nó (trả về một collection gồm các phần tử chuỗi)

### 3.5. MỆNH ĐỀ SELECT (3)

Bạn có thể trả về những đối tượng phức tạp mà dữ liệu được trích xuất ra **from ... in ...** hay những dữ liệu do tính toán ... Ví dụ trả về kiểu vô danh chứa các trường thông tin trả về với toán tử **new {}**

```
var ketqua = from product in products
    select new {
        ten = product.Name.ToUpper(),
        mausac = string.Join(',', product.Colors)
    };
foreach (var item in ketqua) Console.WriteLine(item.ten + " - " + item.mausac);
// Bàn trà - Xám,Xanh
// Tranh treo - Vàng,Xanh
```

Có nghĩa là với mỗi sản phẩm lấy ra, trả về một đối tượng chứa tên lấy bằng tên sản phẩm chuyển hết thành in hoa, mausac là chuỗi nối các màu sắc của sản phẩm.

## 3.6. MỆNH ĐỀ QUERY LỌC DỮ LIỆU TRONG LINQ

Mệnh đề **where** để lọc dữ liệu, sau từ khóa where là biểu thức logic xác định các phần tử lọc ra. Ví dụ:

*where product.Price == 500*

Bạn có thể viết các điều kiện phức tạp

*where (product.Price >= 600 && product.Price < 700) //  
product.Name.StartsWith("Bàn")*

Thậm chí, trong một truy vấn có thể viết nhiều mệnh đề **where**

*var ketqua = from product in products*

*where product.Price >= 500*

*where product.Name.StartsWith("Giường")*

*select product;*

*// ID 6 - Giường ngủ, giá 500*

### 3.7. FROM KẾT HỢP

Để lọc dữ liệu phức tạp hơn, có thể dùng From kết hợp để lọc phức tạp và chi tiết hơn. Ví dụ, mỗi tên sản phẩm nó có một mảng màu. Lọc ra những sản phẩm có chứa màu nào đó.

```
var ketqua = from product in products  
    from color in product.Colors  
    where product.Price < 500  
    where color == "Vàng"  
    select product;  
foreach (var product in ketqua) Console.WriteLine(product.ToString());  
// ID 2 - Túi da, giá 300  
// ID 4 - Tranh treo, giá 400
```

### 3.8. MỆNH ĐỀ ORDERBY SẮP XẾP KẾT QUẢ

Để sắp xếp kết quả sử dụng **orderby** viết sau mệnh đề **where** nếu có, ví dụ sắp xếp tăng dần (từ nhỏ đến lớn) theo thuộc tính dữ liệu (hoặc thuộc tính) **thuoctinh** thì viết **orderby thuoctinh**

Nếu muốn xếp theo thứ tự giảm dần (lớn đến bé)

**orderby thuoctinh descending**

Ví dụ:

```
var ketqua = from product in products  
             where product.Price <= 300  
             orderby product.Price descending  
             select product;
```

```
foreach (var product in ketqua) Console.WriteLine($"{product.Name} - {product.Price}");  
// Túi da - 300  
// Bàn học - 200
```

Cũng có thể sắp xếp theo nhiều dữ liệu, viết cách nhau bởi ,

**orderby thuoctinh1 descending, thuoctinh2, thuoctinh3 descending ...**

### 3.9. MỆNH ĐỀ GROUP... BY

Cuối truy vấn có thể sử dụng group thay cho select, nếu sử dụng group thì nó trả về theo từng nhóm (nhóm lại theo trường dữ liệu nào đó), mỗi phần tử của câu truy vấn trả về là kiểu **IGrouping< TKey, TElement >**, chứa các phần tử thuộc một nhóm.

Ví dụ, lấy sản phẩm có giá từ 400 - 500, nhóm lại theo giá (cùng giá cho vào một nhóm)

```
var ketqua = from product in products
```

```
    where product.Price >= 400 && product.Price <= 500
```

```
    group product by product.Price;
```

```
foreach (var group in ketqua) {
```

```
    // Key là giá trị dùng để phân loại (nhóm): là giá
```

```
    Console.WriteLine(group.Key);
```

```
    foreach (var product in group) {
```

```
        Console.WriteLine($" {product.Name} - {product.Price}");
```

```
    }
```

```
}
```

```
// 400
```

```
    // Bàn trà - 400
```

```
    // Tranh treo - 400
```

```
// 500
```

```
    // Đèn trùm - 500
```

```
    // Giày - 500
```

### 3.9. MỆNH ĐỀ GROUP.. BY (2)

Có thể lưu tạm group trong truy vấn vào một biến bằng cách sử dụng **into**, sau đó thi hành các mệnh đề khác trên biến tạm và dùng mệnh đề select để trả về kết quả

// Kết quả giống câu truy vấn trên, nhưng các nhóm xếp từ lớn xuống nhỏ

```
var ketqua = from product in products  
            where product.Price >= 400 && product.Price <= 500  
            group product by product.Price into gr
```

```
            orderby gr.Key descending
```

```
            select gr;
```

// 500

// Đèn chùm - 500

// Giường ngủ - 500

// 400

// Bàn trà - 400

// Tranh treo - 400

### 3.10. LET – DÙNG BIẾN TRUY VẤN

Dùng thêm biến vào LINQ, lưu kết quả của một biểu thức tính toán nào đó, thêm vào mệnh đề bằng cách viết **let tenvien = biểu\_thức**, ví dụ với mỗi loại giá - có bao nhiêu sản phẩm

```
var ketqua = from product in products           // các sản phẩm trong products
              group product by product.Price into gr // nhóm thành gr theo giá
              let count = gr.Count()                  // số phần tử trong nhóm
              select new {                            // trả về giá và số sản phẩm có giá này
                price = gr.Key,
                number_product = count
              };

foreach (var item in ketqua)
{
    Console.WriteLine($"{item.price} - {item.number_product}");
}

// 200 - 1
// 300 - 1
// 400 - 2
// 500 - 2
// 600 - 1
```

### 3.11. MỆNH ĐỀ JOIN – KHỚP NỐI NGUỒN TRUY VẤN

join là thực hiện kết hợp hai nguyên dữ liệu lại với nhau để truy vấn thông tin, ví dụ trong mỗi sản phẩm đều có **Brand** nó là ID của nhãn - vậy mỗi sản phẩm dùng thông tin này để có được thông tin chi tiết về nhãn hàng mà nhãn hàng lại lưu ở nguồn khác.

Các sản phẩm ở ví dụ phía trên, sản phẩm nào có Brand là 2 thì tra cứu ở Brand.brands thì tên nhãn là "Công ty AAA"

Giờ muốn thực hành truy vấn kết hợp khớp nối giữa hai nguồn dữ liệu danh sách sản phẩm products và danh sách nhãn hàng brands

### 3.11. MỆNH ĐỀ JOIN – KHỚP NỐI NGUỒN TRUY VẤN (2)

#### Inner join

Để kết nối, dùng mệnh đề **join** để chỉ ra nguồn (nguồn bên phải join) sẽ kết nối với nguồn của **from** (nguồn bên trái join), tiếp theo chỉ ra sự ràng buộc các phần tử bằng từ khóa **on**

Ví dụ, truy vấn sản phẩm, mỗi sản phẩm căn cứ vào Brand ID của nó - lấy tên Brand tương ứng

### 3.11. MỆNH ĐỀ JOIN – KHỚP NỐI NGUỒN TRUY VẤN (3)

```
var ketqua = from product in products
              join brand in brands on product.Brand equals brand.ID
              select new {
                  name = product.Name,
                  brand = brand.Name,
                  price = product.Price
              };

foreach (var item in ketqua)
{
    Console.WriteLine($"{item.name,10} {item.price, 4} {item.brand,12}");
}

//      Túi da  300  Công ty BBB
//      Bàn trà  400  Công ty BBB
// Tranh treo  400  Công ty AAA
// Giường ngủ 500  Công ty BBB
```

### 3.11. MỆNH ĐỀ JOIN – KHỚP NỐI NGUỒN TRUY VẤN (4)

Phân tích truy vấn trên ta thấy

- nguồn bên trái join là : **product in products**
- nguồn bên phải join là : **brand in brands**
- liên kê kết nối là: **on product.Brand equals brand.ID** (Brand trong product bằng (equals) ID của brand)

Với truy vấn join trên, chỉ những dữ liệu product mà có brand tương ứng mới trả về, nên nó gọi là inner join (tức giá trị product.brand hay brand.ID có ở cả 2 nguồn). Kết quả trả về có 4 dòng như trên. Để ý trong danh sách sản phẩm có sản phẩm "Tủ áo", sản phẩm này có brand là 3, nhưng ở danh sách brands lại không có phần tử nào ID bằng 3 nên truy vấn trên sẽ bỏ sản phẩm "Tủ áo"

### **3.11. MỆNH ĐỀ JOIN – KHỚP NỐI NGUỒN TRUY VẤN (5)**

#### **Left join**

Trong truy vấn trên, sản phẩm nào không tìm được thông tin brand ở nguồn bên phải join thì sẽ bỏ qua. Giờ muốn, các sản phẩm kể cả không thấy brand cũng trả về - có nghĩa nguồn bên trái lấy không phụ thuộc vào bên phải thì dùng kỹ thuật left join như sau:

### 3.11. MỆNH ĐỀ JOIN – KHỚP NỐI NGUỒN TRUY VẤN (6)

```
var ketqua = from product in products
              join brand in brands on product.Brand equals brand.ID into t
              from brand in t.DefaultIfEmpty()
              select new {
                  name  = product.Name,
                  brand = (brand == null) ? "NO-BRAND" : brand.Name,
                  price = product.Price
              };

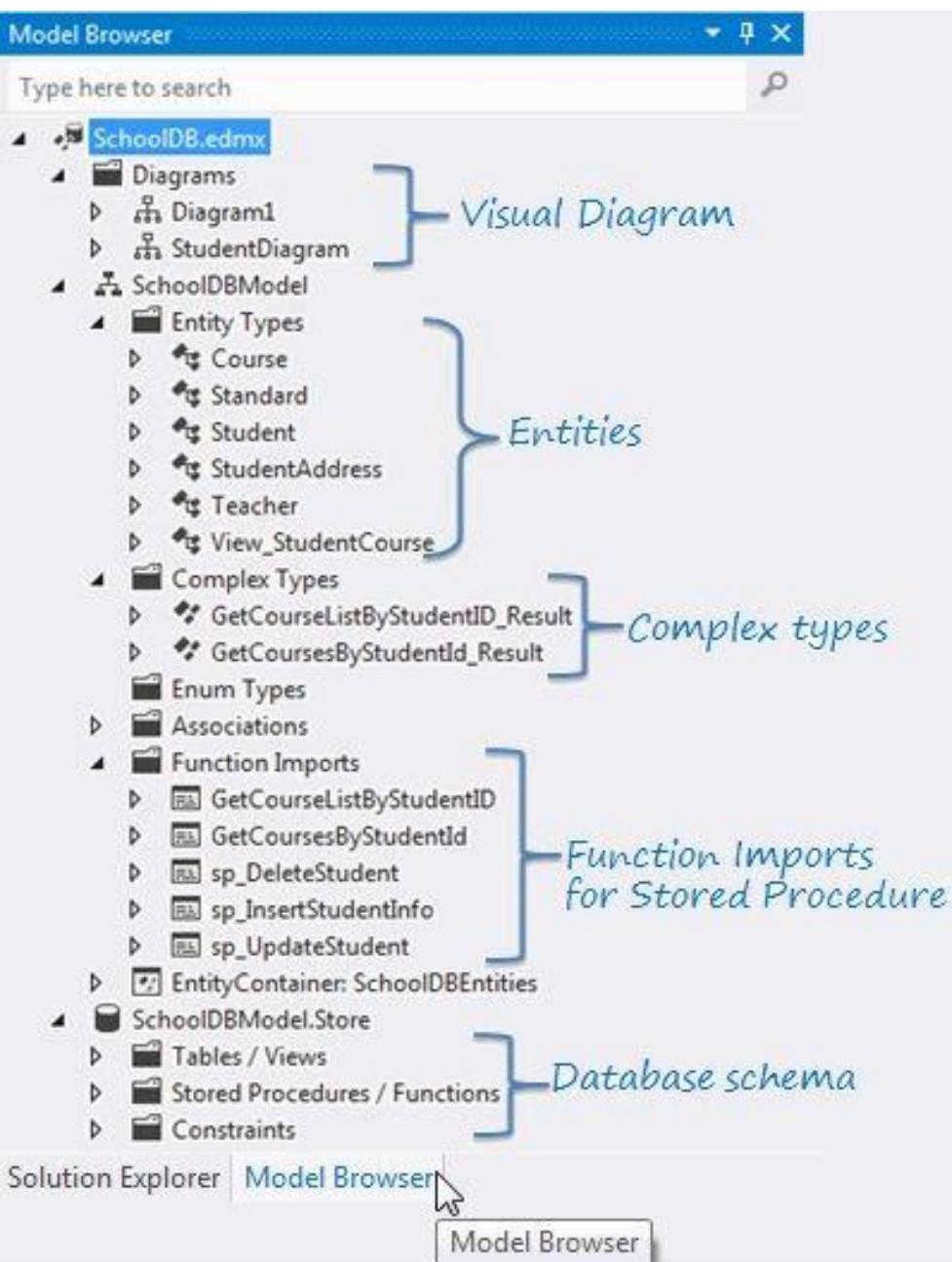
foreach (var item in ketqua)
{
    Console.WriteLine($"{item.name,10} {item.price, 4} {item.brand,12}");
}

//      Bàn học  200  Công ty AAA
//      Túi da   300  Công ty BBB
//      Bàn trà   400  Công ty BBB
// Tranh treo  400  Công ty AAA
//  Đèn trùm   500    NO-BRAND
// Giường ngủ  500  Công ty BBB
//      Tủ áo   600    NO-BRAND
```

## 4.1. GIỚI THIỆU ENTITY FRAMEWORK

- Entity Framework phát hành đầu tiên vào năm 2008 nhằm hỗ trợ sự tương tác giữa các ứng dụng trên nền tảng .NET với các cơ sở dữ liệu quan hệ.
- Hay nói cách khác, nó là một công cụ giúp ánh xạ giữa các đối tượng trong phần mềm của bạn với các bảng của một cơ sở dữ liệu quan hệ.
- Trình duyệt mô hình (Model Browser) cung cấp thông tin về tất cả các đối tượng và chức năng mà Entity Data Model (EDM) đã tạo.
- Để mở Trình duyệt mô hình, nhấp chuột phải vào chỗ trống trong trình thiết kế EDM và chọn Model Browser từ menu ngữ cảnh
- Model Browser sẽ xuất hiện ở Solution Explorer và Properties.
- Trình duyệt mô hình chứa tất cả thông tin về EDM, mô hình khái niệm, mô hình lưu trữ và thông tin ánh xạ, như được hiển thị bên dưới.

## 4.1. GIỚI THIỆU ENTITY FRAMEWORK (2)



**Diagrams:** Trình duyệt mô hình chứa các sơ đồ trực quan của EDM. Chúng ta đã thấy một sơ đồ trực quan mặc định được tạo bởi EDM. Bạn cũng có thể tạo nhiều sơ đồ cho một EDM.

**Entity Types:** Chứa danh sách các lớp thực thể ánh xạ tới các bảng trong cơ sở dữ liệu.

**Complex Types:** Các kiểu dữ liệu phức tạp là các lớp được tạo bởi EDM để chứa kết quả của các stored procedure, các table-valued function, v.v. Các kiểu dữ liệu phức tạp này là các lớp tùy chỉnh cho các mục đích khác nhau.

**Enum Types:** Chứa các thực thể được sử dụng làm enum trong Entity Framework.

**Associations:** Chứa tất cả các mối quan hệ khóa ngoại giữa các thực thể.

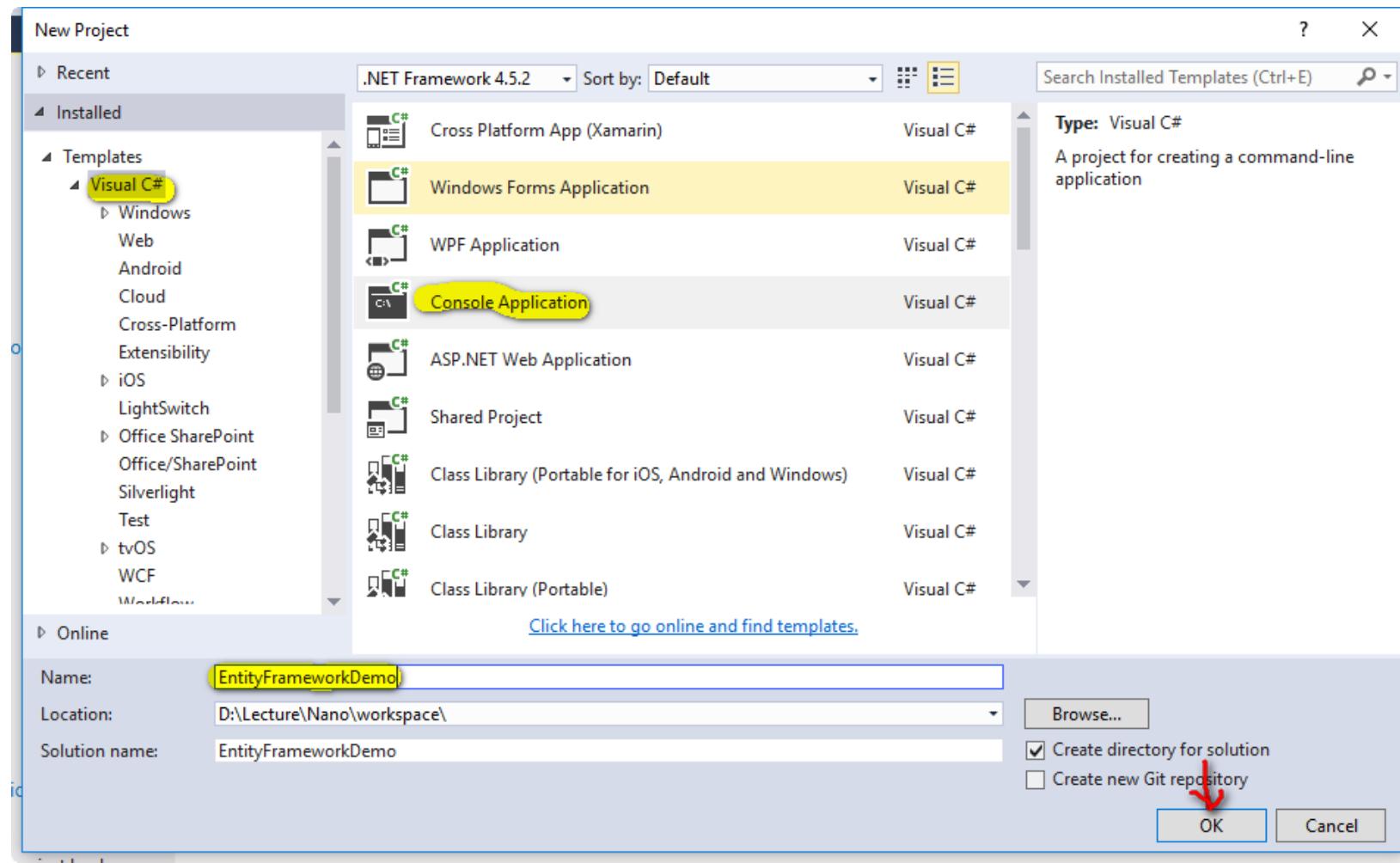
**Function Imports:** Chứa tất cả các chức năng sẽ được ánh xạ tới các stored procedure, các table-valued function, v.v ... Các stored procedure và các table-valued function sẽ được sử dụng như các phương thức chứ không phải là các thực thể trong Entity Framework.

**Store:** Đại diện cho lược đồ cơ sở dữ liệu (SSDL)

## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL

### Bước 1: Tạo project và cài đặt Entity Framework

1/ Tạo project: Mở Visual Studio -> tạo một project và đặt tên **EntityFrameworkDemo**

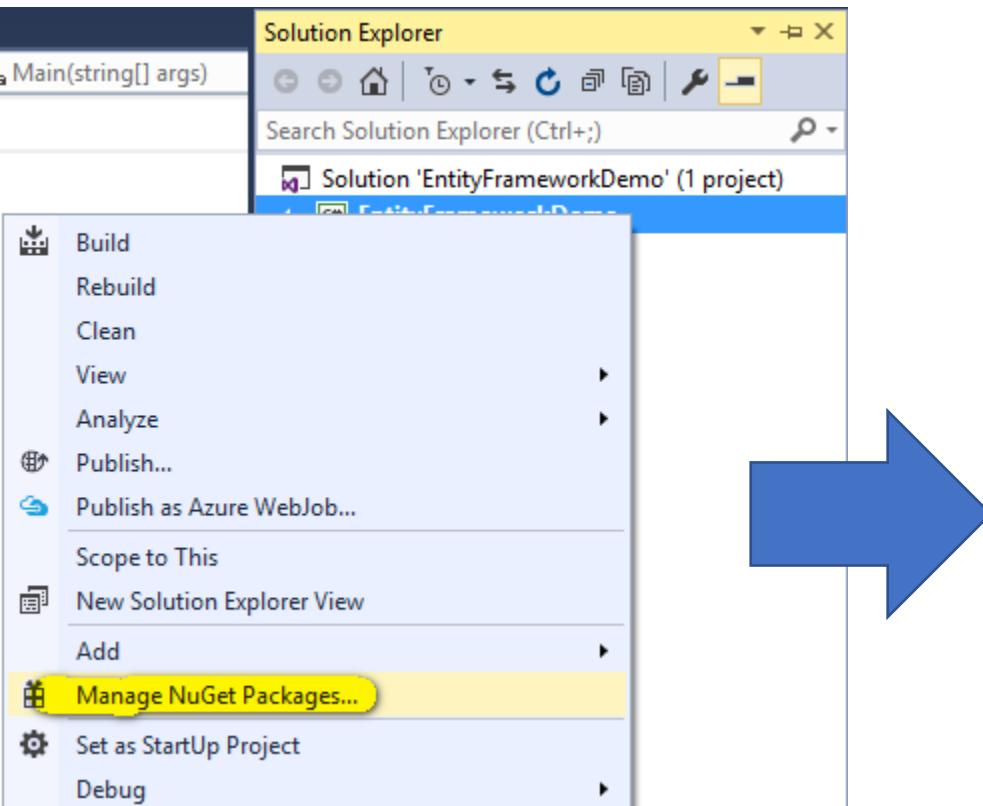


## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (2)

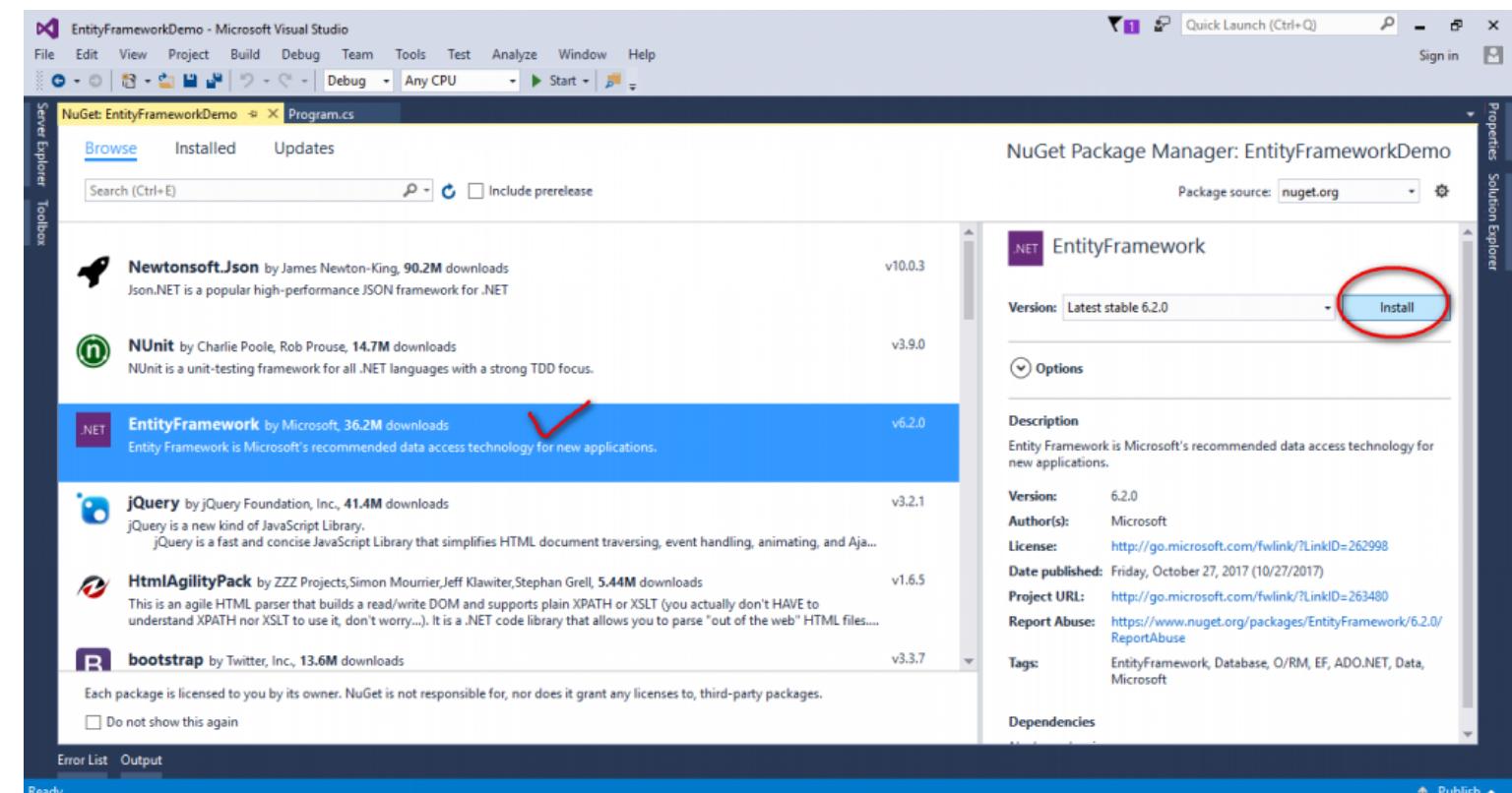
Bước 1: Tạo project và cài đặt Entity Framework

2/ Cài đặt Entity Framework phiên bản mới nhất

Trong Solution Explorer, chuột phải lên project -> chọn Manage NuGet Packages...



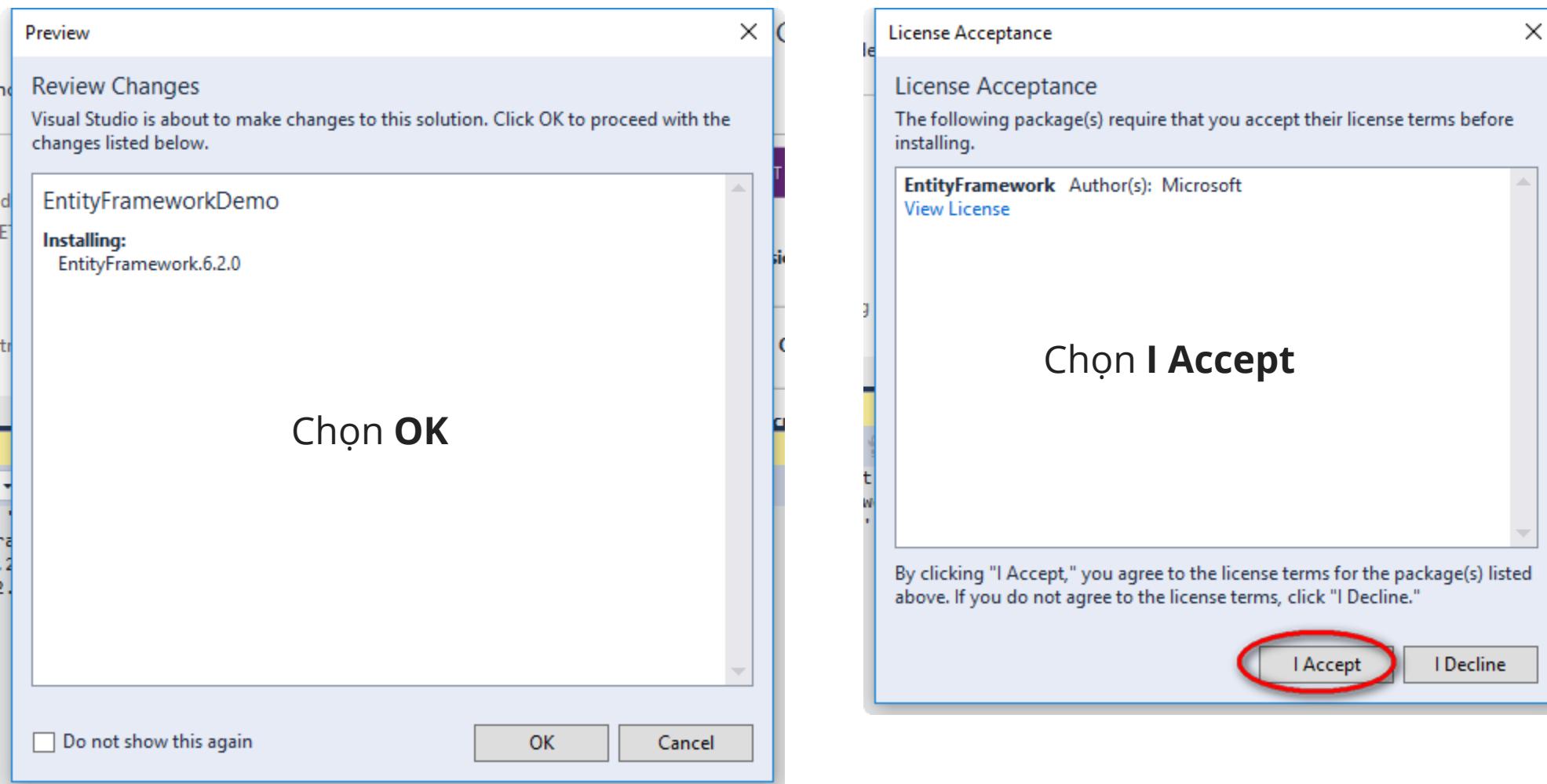
Chọn Entity Framework > chọn Install



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (3)

Bước 1: Tạo project và cài đặt Entity Framework

2/ Cài đặt Entity Framework phiên bản mới nhất



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (4)

Bước 1: Tạo project và cài đặt Entity Framework

### 2/ Cài đặt Entity Framework phiên bản mới nhất

Sau khi cài đặt thành công, chúng ta sẽ nhìn thấy thông báo như hình bên dưới

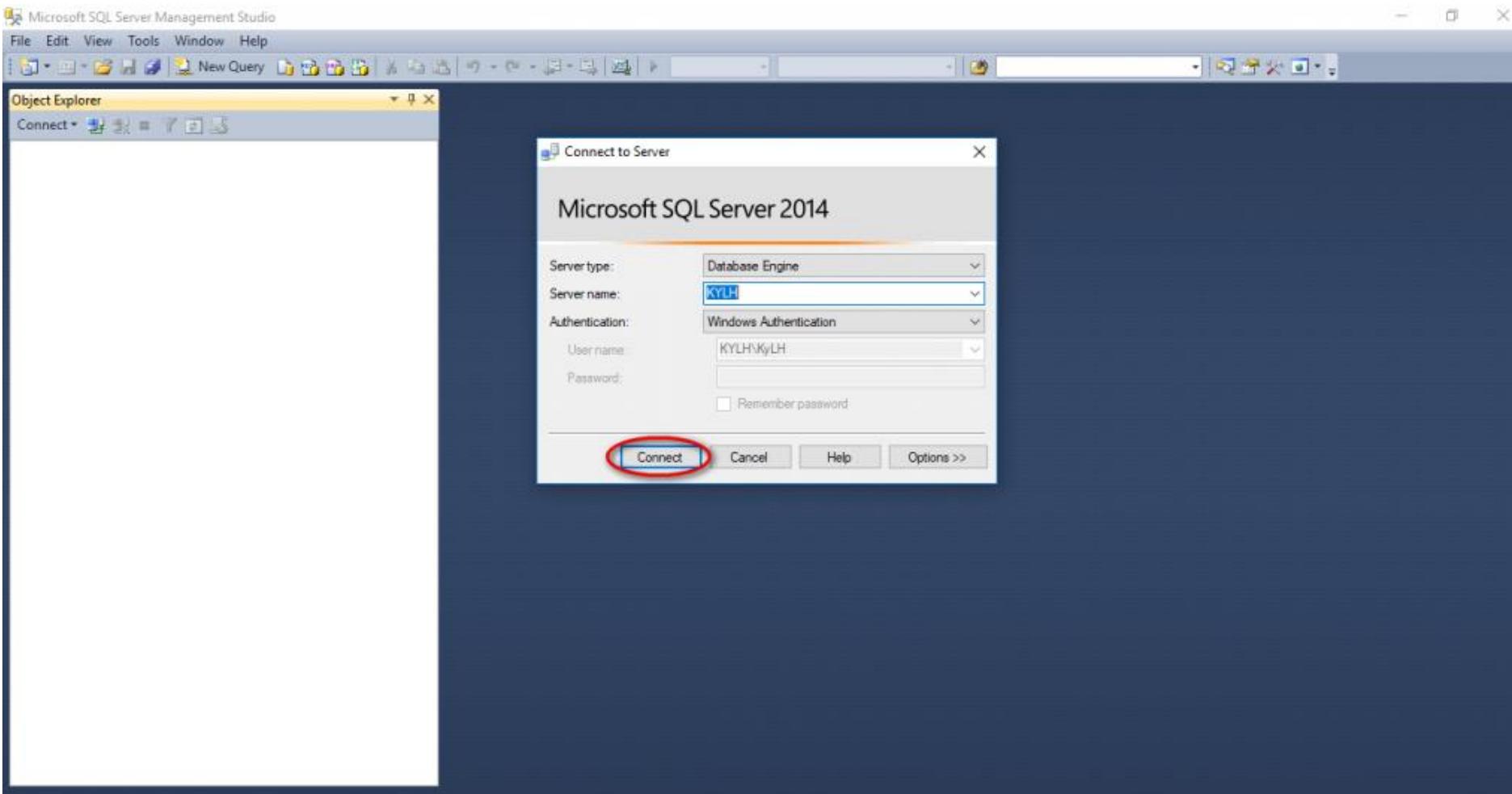
```
Output
Show output from: Package Manager
Resolving actions to install package 'EntityFramework.6.2.0'
Resolved actions to install package 'EntityFramework.6.2.0'
  GET https://api.nuget.org/v3-flatcontainer/entityframework/6.2.0/entityframework.6.2.0.nupkg
  OK https://api.nuget.org/v3-flatcontainer/entityframework/6.2.0/entityframework.6.2.0.nupkg 1236ms
Installing EntityFramework 6.2.0.
Adding package 'EntityFramework.6.2.0' to folder 'D:\Lecture\Nano\workspace\EntityFrameworkDemo\packages'
Added package 'EntityFramework.6.2.0' to folder 'D:\Lecture\Nano\workspace\EntityFrameworkDemo\packages'
Added package 'EntityFramework.6.2.0' to 'packages.config'
Executing script file 'D:\Lecture\Nano\workspace\EntityFrameworkDemo\packages\EntityFramework.6.2.0\tools\init.ps1'...
Executing script file 'D:\Lecture\Nano\workspace\EntityFrameworkDemo\packages\EntityFramework.6.2.0\tools\install.ps1'...

Type 'get-help EntityFramework' to see all available Entity Framework commands.
Successfully installed 'EntityFramework 6.2.0' to EntityFrameworkDemo
===== Finished =====
```

## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (5)

### Bước 2: Tạo Cơ sở dữ liệu

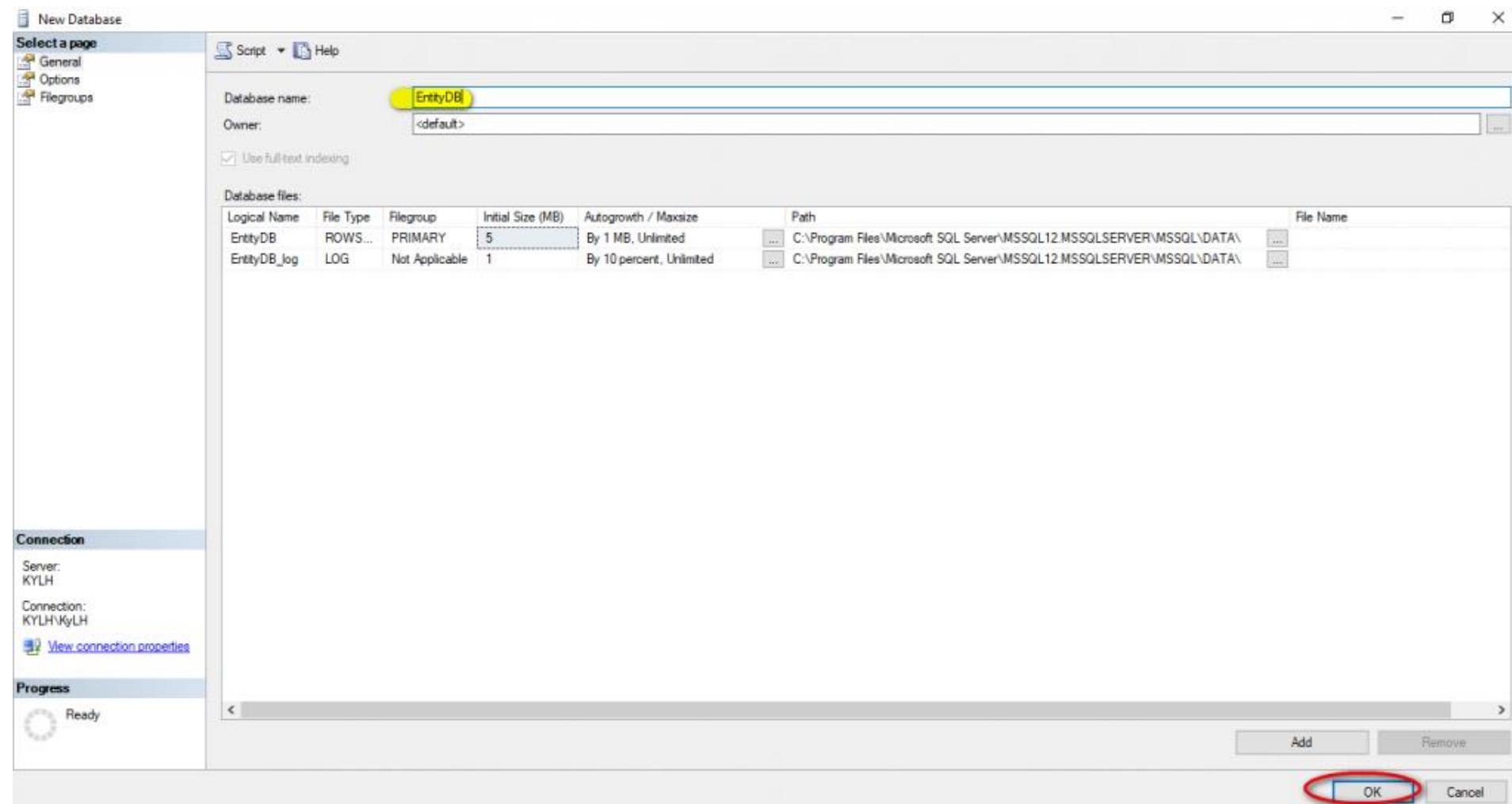
1/ Mở SQL Server Management Studio -> kết nối đến server



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (6)

### Bước 2: Tạo Cơ sở dữ liệu

2/ Tạo một cơ sở dữ liệu tên EntityDB



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (7)

### Bước 2: Tạo Cơ sở dữ liệu

3/ Tạo một table tên **Student** có 4 column là **StudentID**, **StudentName**, **StudentGender** và **Address**

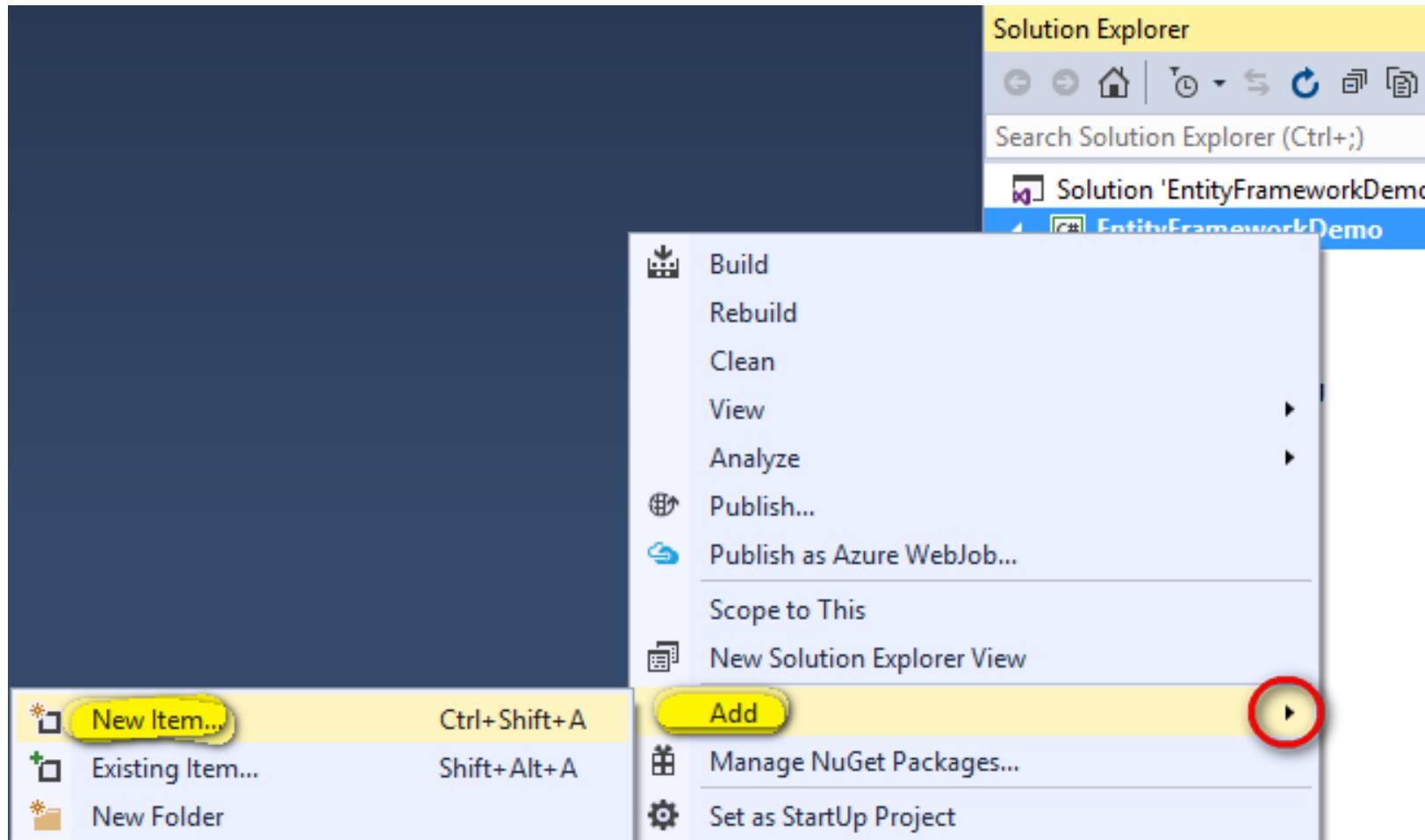
The screenshot shows the Microsoft SQL Server Object Explorer interface. The left pane displays a tree structure of database objects. At the top level is 'KYLH'. Under 'KYLH', there are 'Databases' (including 'System Databases', 'Database Snapshots', '32551-import', 'EduSV', and 'EntityDB'), 'Database Diagrams', and 'Tables'. The 'Tables' node has two sub-nodes: 'System Tables' and 'FileTables'. The 'dbo.Student' table is highlighted with a red box and is expanded to show its four columns: 'StudentID (PK, int, not null)', 'StudentName (nvarchar(50), null)', 'StudentGender (nvarchar(50), null)', and 'Address (nvarchar(100), null)'. The 'EntityDB' database is also highlighted with a yellow box.

## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (8)

Bước 3: Sử dụng Entity Framework

### 1/ Tạo ADO.NET Entity Data Model

Trong **Solution Explorer**, chuột phải lên project -> chọn **Add** -> chọn **New Item...**

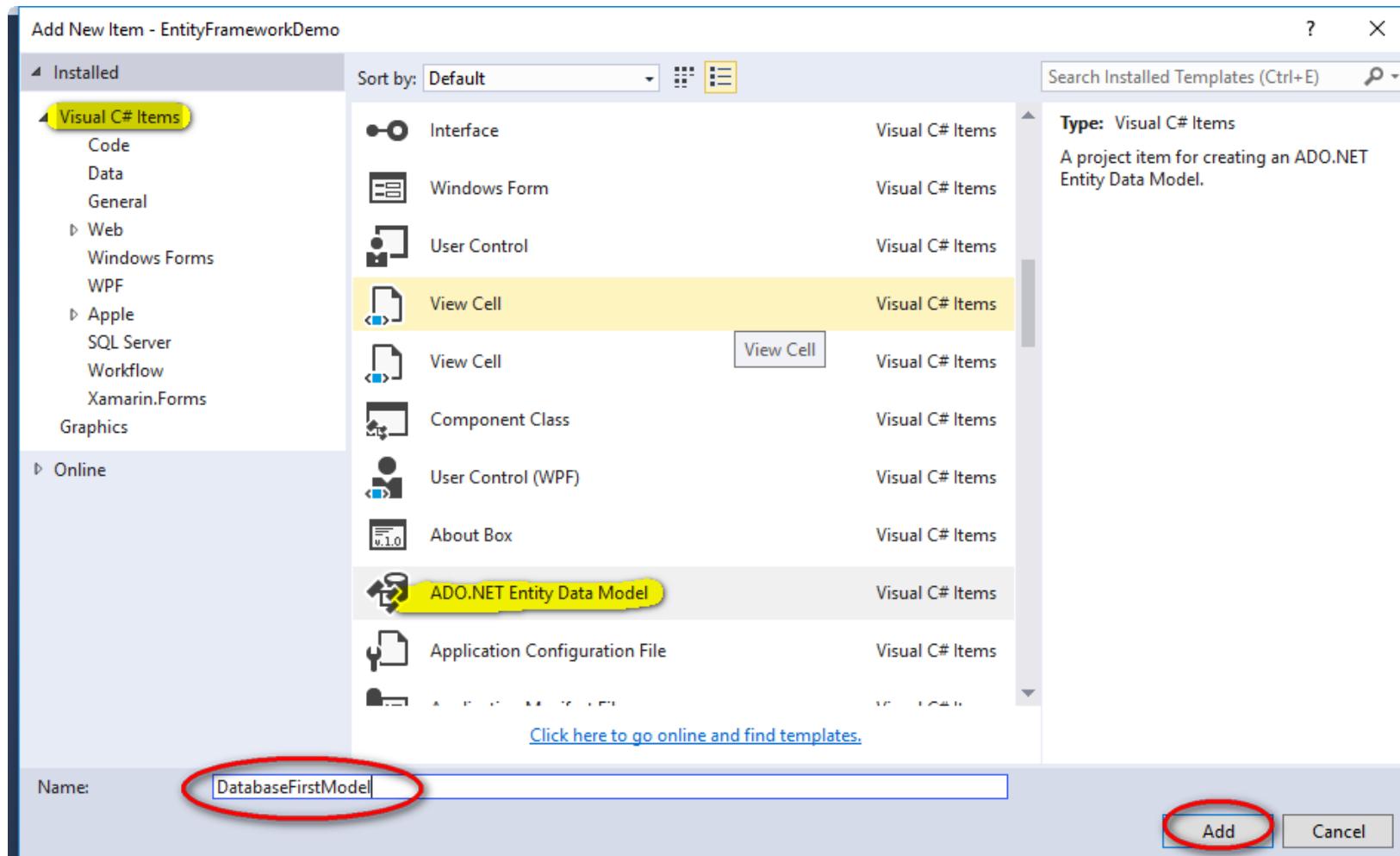


## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (9)

### Bước 3: Sử dụng Entity Framework

#### 1/ Tạo ADO.NET Entity Data Model

Chọn Visual C# Items -> chọn ADO.NET Entity Data Model -> nhập DatabaseFirstModel -> chọn Add



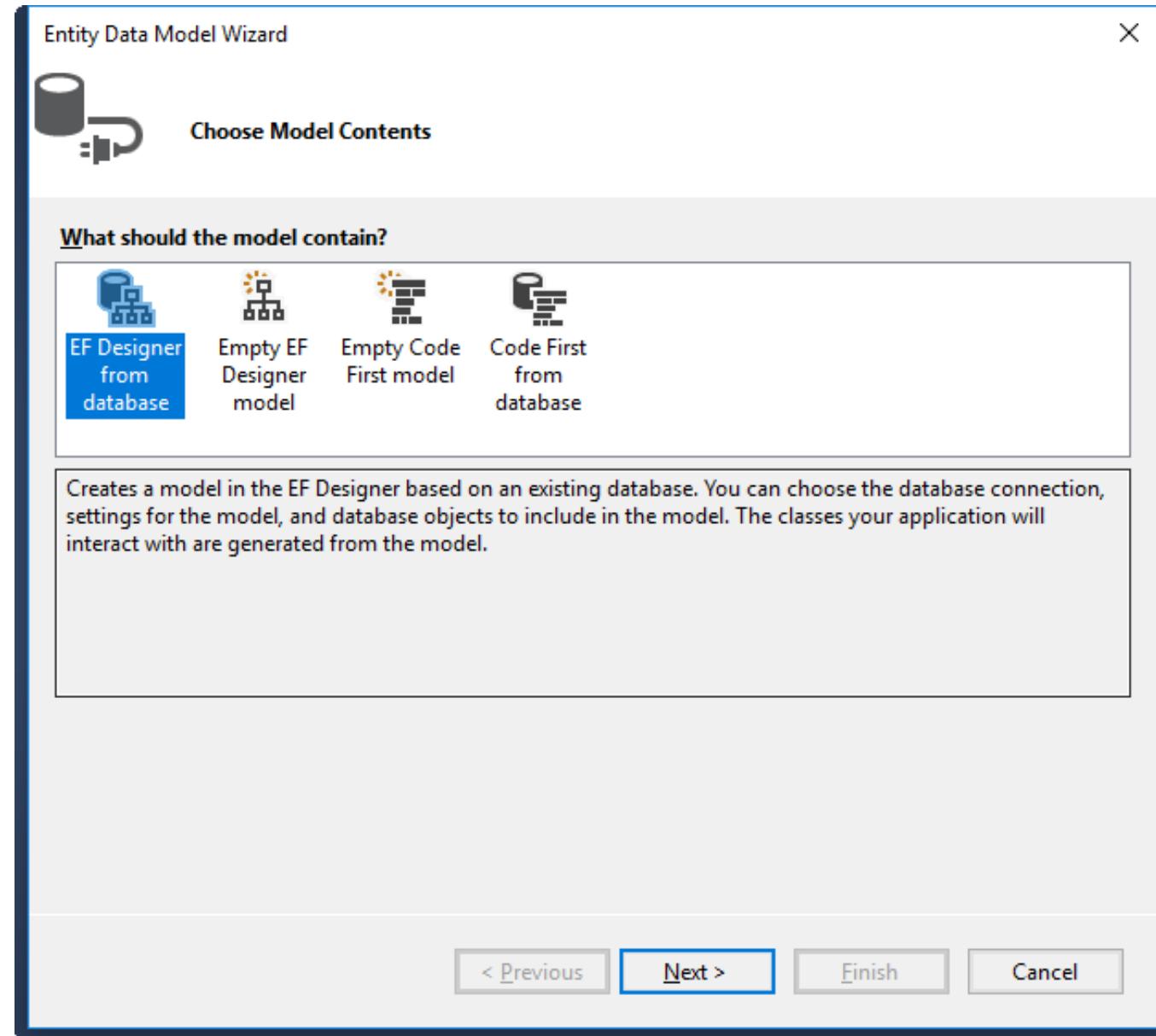
## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (10)

Bước 3: Sử dụng Entity Framework

1/ Tạo ADO.NET Entity Data Model

Chọn EF Designer from database

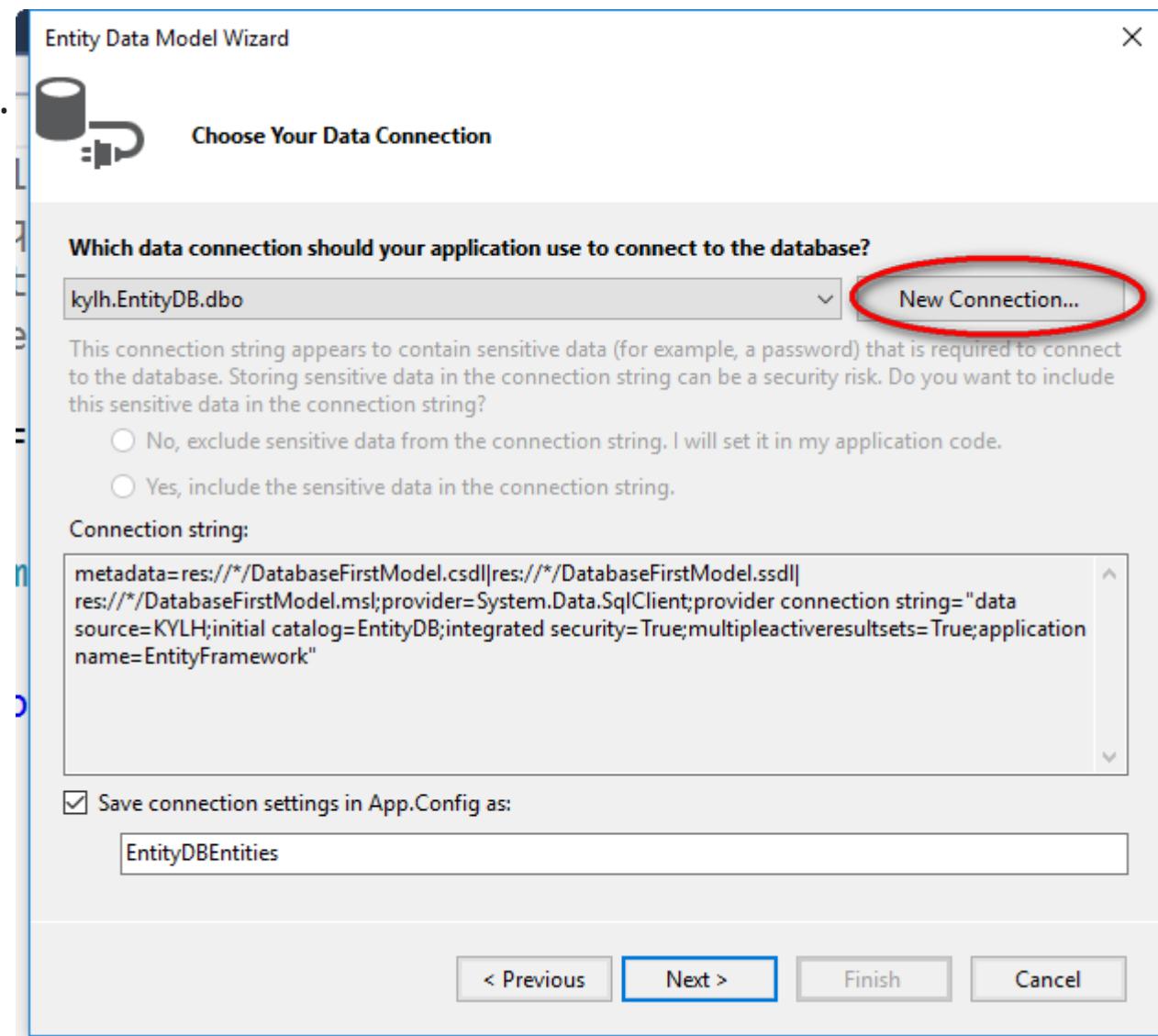
-> chọn Next



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (11)

### Bước 3: Sử dụng Entity Framework

Chọn kết nối hiện có hoặc  
chọn **New Connection...** để thiết lập kết nối mới.

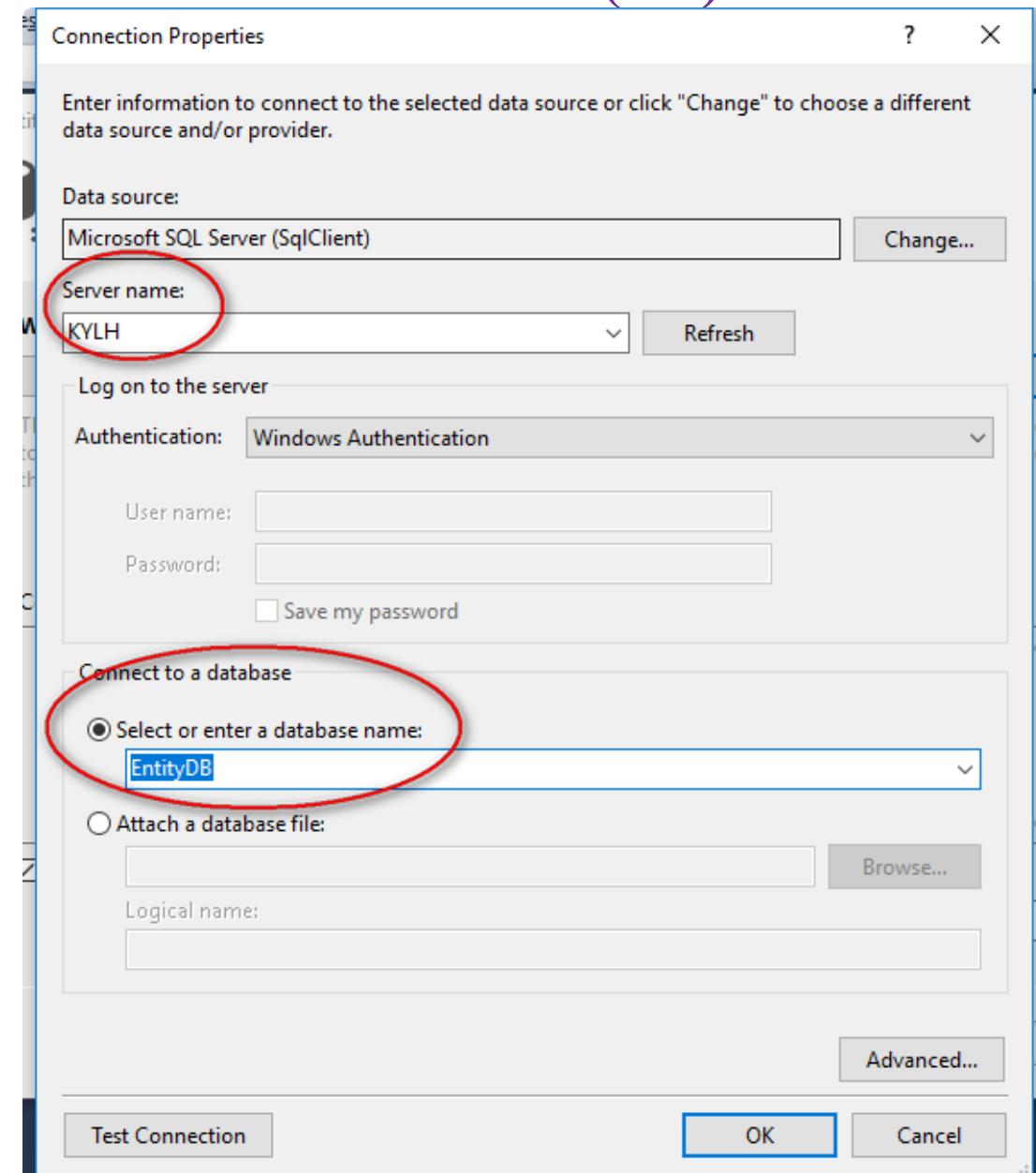


## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (11)

### Bước 3: Sử dụng Entity Framework

Trường hợp chọn **New Connection...**

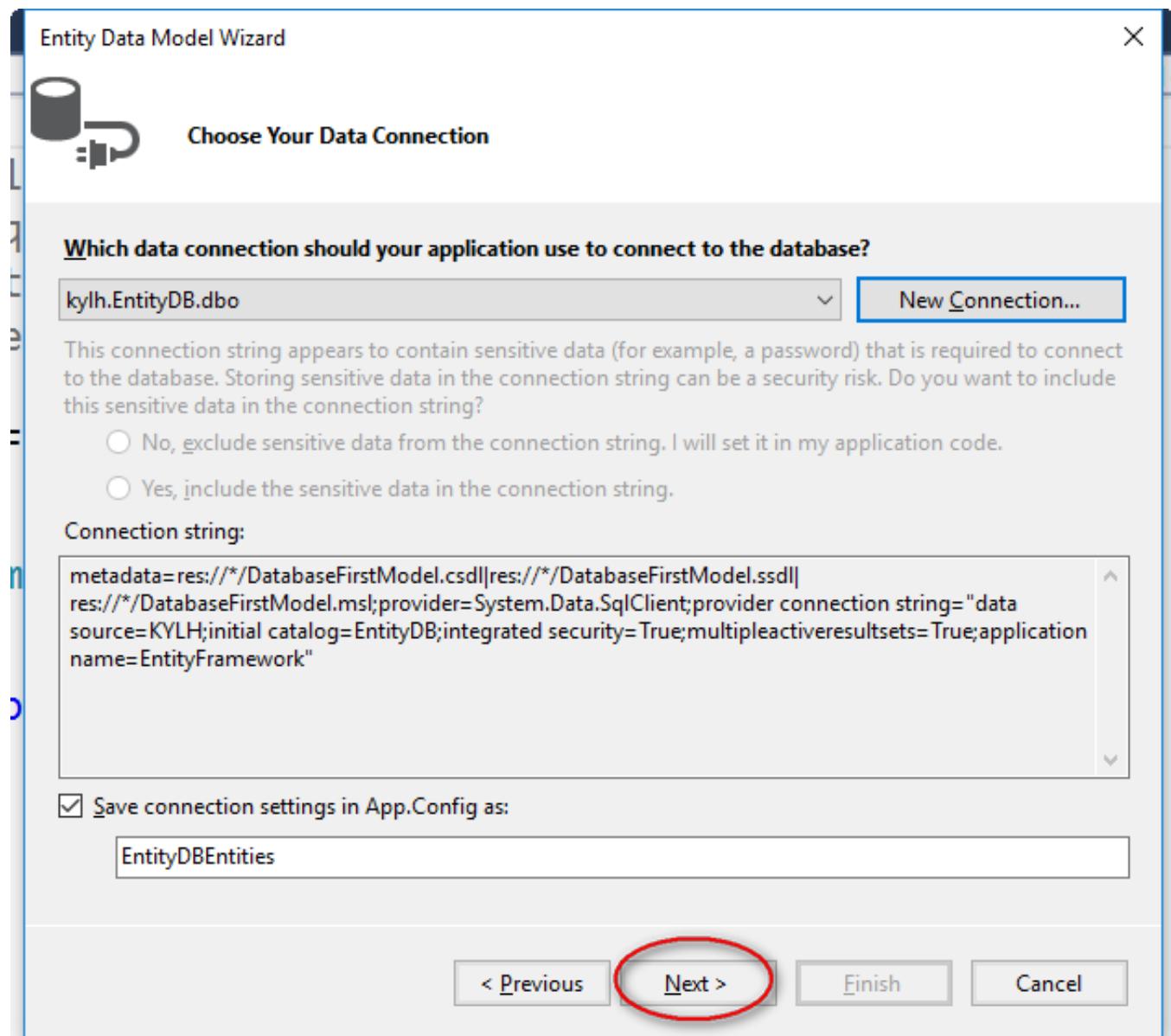
- > chọn tên máy chủ (**Server name**),  
chọn cơ sở dữ liệu (**Database name**)
- > chọn **OK**



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (12)

### Bước 3: Sử dụng Entity Framework

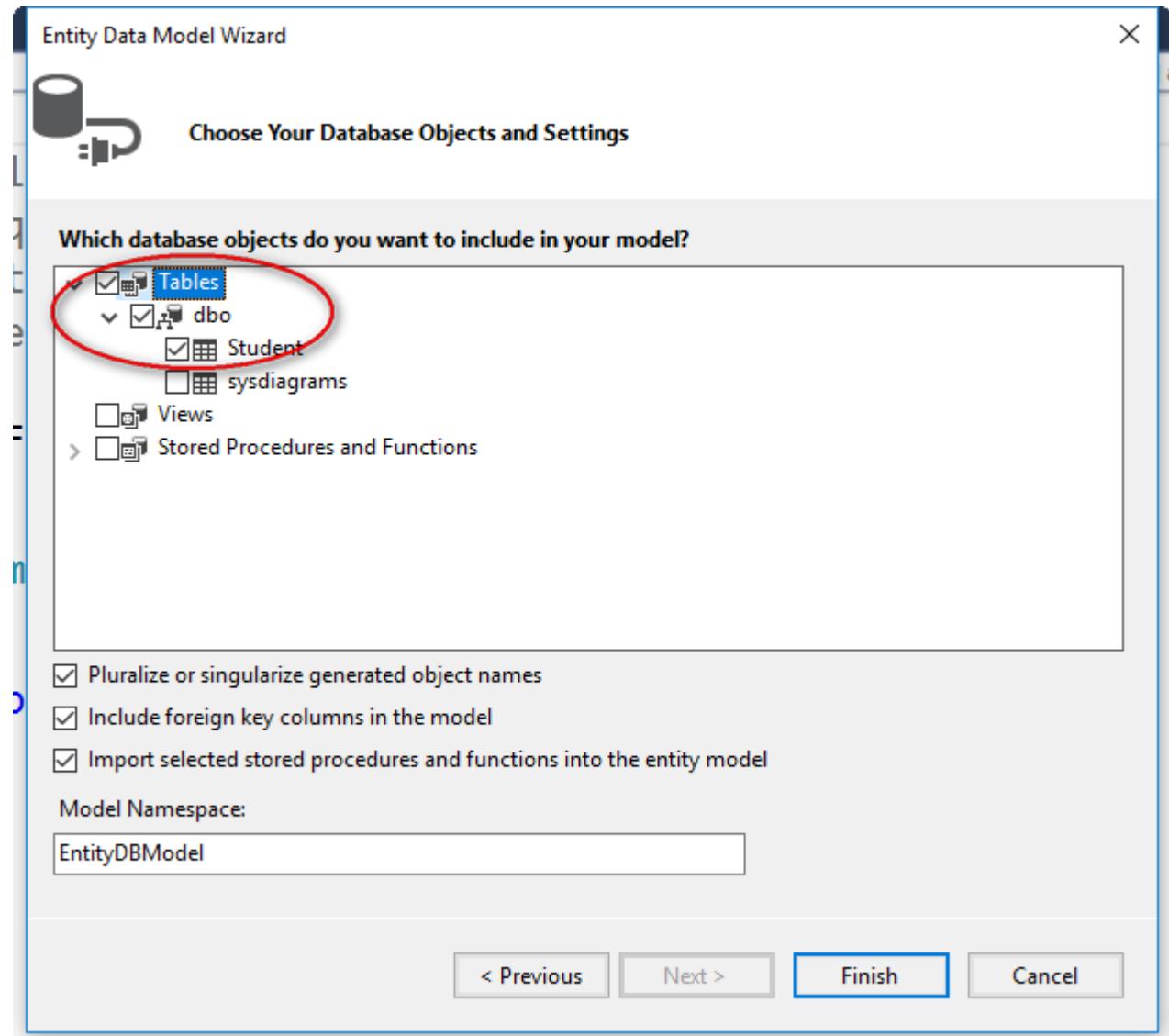
Chọn Next



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (13)

Bước 3: Sử dụng Entity Framework

Chọn table và chọn **Finish**



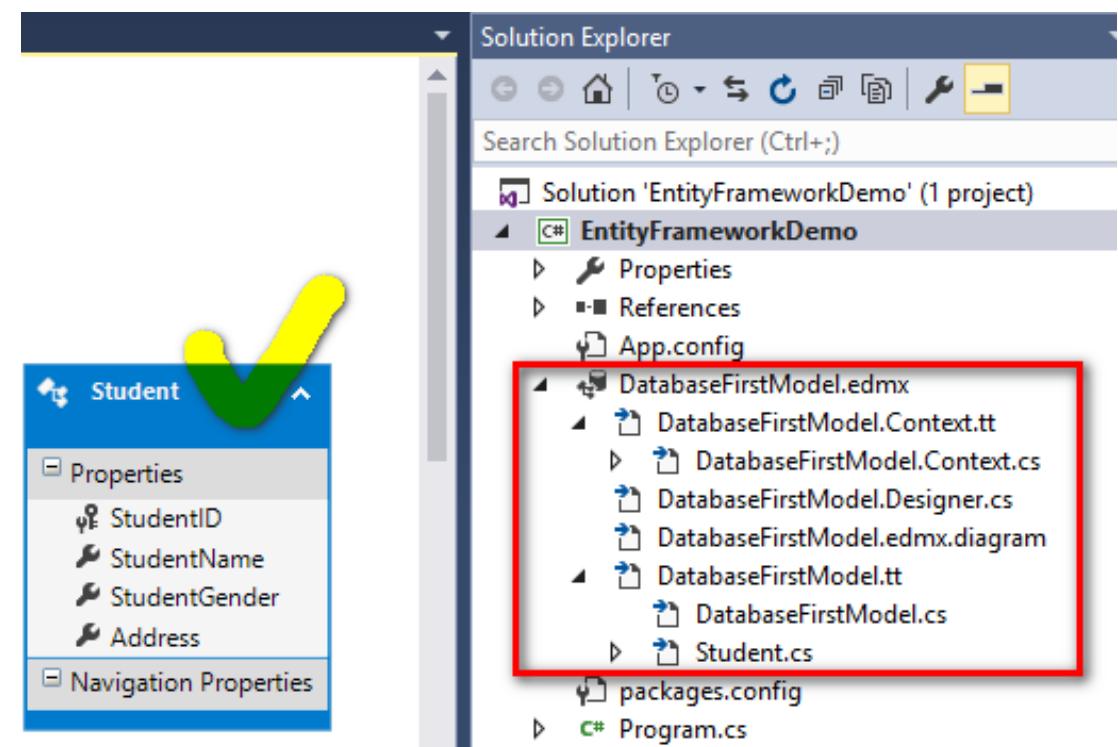
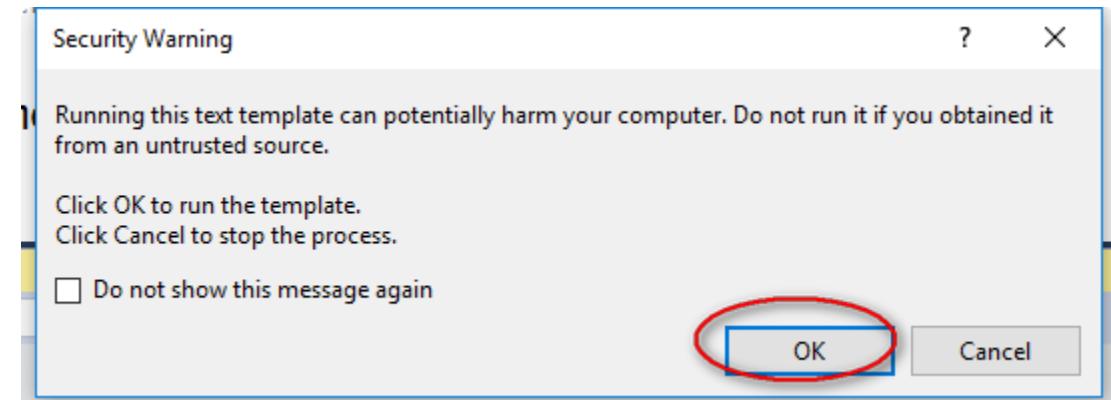
## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (14)

### Bước 3: Sử dụng Entity Framework

Chọn 2 lần OK

Kết quả thu được sẽ như hình bên dưới.

**Lưu ý phải lưu DatabaseFirstModel.edmx trước khi thực hiện viết code**



## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (15)

Bước 3: Sử dụng Entity Framework

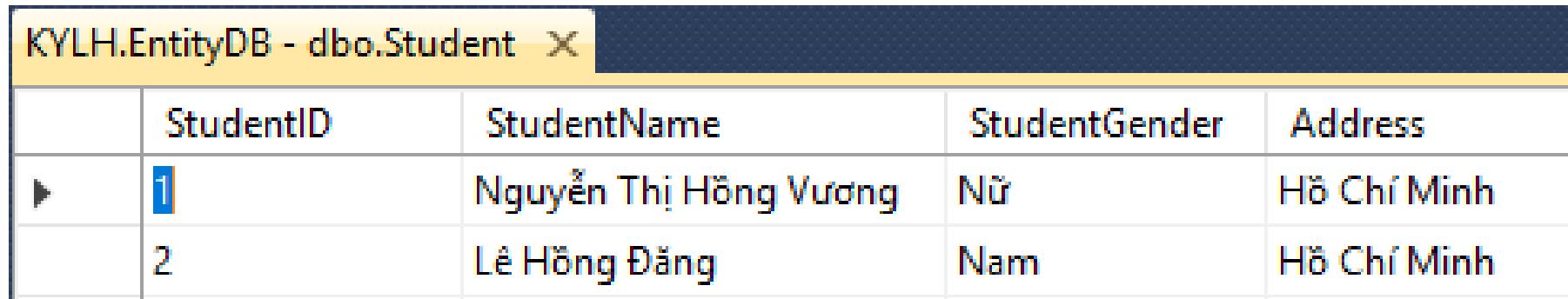
2/ Viết code: Mở Program.cs và nhập code như sau

```
9  class Program
10 {
11     static void Main(string[] args)
12     {
13         Console.OutputEncoding = Encoding.UTF8;
14         using (var db = new EntityDBEntities())
15         {
16             var select = from s in db.Students select s;
17             foreach(var data in select)
18             {
19                 Console.WriteLine("ID: {0}", data.StudentID);
20                 Console.WriteLine("Name: {0}", data.StudentName);
21                 Console.WriteLine("Gender: {0}", data.StudentGender);
22                 Console.WriteLine("Address: {0}", data.Address);
23             }
24         }
25         Console.ReadLine();
26     }
27 }
```

## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (16)

Bước 3: Sử dụng Entity Framework

3/ Mở SQL Server Management Studio -> mở bảng Student và thêm dữ liệu



The screenshot shows a database table named 'Student' from the 'KYLH.EntityDB' database. The table has five columns: StudentID, StudentName, StudentGender, and Address. There are two rows of data:

	StudentID	StudentName	StudentGender	Address
▶	1	Nguyễn Thị Hồng Vương	Nữ	Hồ Chí Minh
	2	Lê Hồng Đăng	Nam	Hồ Chí Minh

## 4.2. HƯỚNG DẪN TẠO ENTITY DATA MODEL (17)

### Bước 3: Sử dụng Entity Framework

#### 4/ Build và run chương trình

```
 file:///D:/Lecture/Nano/workspace/EntityFrameworkDemo,
```

```
ID: 1
```

```
Name: Nguyễn Thị Hồng Vương
```

```
Gender: Nữ
```

```
Address: Hồ Chí Minh
```

```
ID: 2
```

```
Name: Lê Hồng Đăng
```

```
Gender: Nam
```

```
Address: Hồ Chí Minh
```

## 4.3. MỘT SỐ XỬ LÝ KHÁC - EF

### 1/ Thêm dữ liệu với Entity Framework

```
// Thông tin của sinh viên được thêm mới  
var student = new Student();  
student.StudentID = 3;  
student.StudentName = "Lê Thị Hồng";  
student.StudentGender = "Nam";  
student.Address = "Bình Dương";  
// Thêm vào database  
using (var db = new EntityDBEntities()) {  
    db.Students.Add(student);  
    db.SaveChanges();  
}
```

## 4.3. MỘT SỐ XỬ LÝ KHÁC – EF (2)

2/ Cập nhật dữ liệu Entity Framework (Cập nhật giới tính thành Nữ cho sinh viên có id là 3)

```
using (var db = new EntityDBEntities()) {  
    var update = (from u in db.Students where u.StudentID == 3 select  
u).Single();  
    u.StudentGender = "Nữ";  
    db.SaveChanges();  
}
```

## 4.3. MỘT SỐ XỬ LÝ KHÁC – EF (3)

### 3/ Xóa dữ liệu Entity Framework (Xóa sinh viên có id là 3)

```
using (var db = new EntityDBEntities()) {  
    var delete = (from d in db.Students where d.StudentID == 3 select  
d).Single();  
  
    db.Students.Remove(delete);  
  
    db.SaveChanges();  
}
```

## 5.1. WEB SERVICES VỚI WCF

WCF là công nghệ nền tảng nhằm thống nhất nhiều mô hình lập trình giao tiếp được hỗ trợ trong .NET 2.0 thành một mô hình duy nhất. Vào tháng 11 năm 2005, .NET 2.0 được Microsoft phát hành trong đó có cung cấp các hàm API riêng biệt cho các liên lạc dựa trên SOAP để tối đa hoá sự làm việc giữa các nền tảng sử dụng Web Services.

.NET 2.0 còn cung cấp các API để tối ưu việc liên lạc dựa trên mã nhị phân giữa các ứng dụng chạy trên hệ thống Windows gọi là .NET Remoting, các API cho các giao dịch phân tán, và API cho liên lạc dị bộ. WCF thống nhất các API này thành một mô hình duy nhất nhằm đáp ứng mô hình lập trình hướng dịch vụ.

## 5.2. CÁC THÀNH PHẦN CỦA WCF

Ba thành phần chính của một WCF service là:

- **Service class:** định nghĩa các contact (sẽ trình bày ở bên dưới).
- **Hosting environment:** WCF service có thể được host lên nhiều “môi trường” khác nhau, bao gồm IIS, Windows service, Self-hosting.
- **End point:** Các “cổng” kết nối giữa client và service.

## 5.3. KIẾN TRÚC CỦA WCF

Kiến trúc của Windows Communication Foundation (WCF) có các lớp chủ yếu sau:

- Contracts (Hợp đồng/Hiệp định)
- Runtime service (Dịch vụ thực thi)
- Host and activation (Chứa và kích hoạt)

## 5.4. CONTRACTS

Contract được coi là một bản “hợp đồng” quy định cách thức và phương tiện để hai bên có thể hợp tác và làm việc với nhau.

Trong WCF, các contact được coi như những bản mô tả nhiệm vụ, kiểu dữ liệu của service. Client sẽ có được các contact sau khi tham chiếu đến service và thông qua đó để gửi và nhận các thông điệp.

Như vậy, service và client không sử dụng cùng các kiểu dữ liệu như nhau, mà chỉ sử dụng chung các contact.

## 5.4. CONTRACTS (2)

**Data contract:** Mô tả các tham số cho các bản tin mà một dịch vụ có thể tạo ra hay sử dụng. Các tham số bản tin được định nghĩa bằng các tài liệu sử dụng ngôn ngữ đặc tả XML Schema (XSD), điều này cho phép các hệ thống hiểu XML có thể xử lý tài liệu dễ dàng. Các dịch vụ khi liên lạc với nhau có thể không cần đồng ý với nhau về các kiểu, nhưng cần đồng ý về contract dữ liệu, nghĩa là đồng ý về các tham số và các kiểu trả về.

**Message contract:** định nghĩa cấu trúc của các thông điệp được truyền tải dưới dạng SOAP (Simple Object Access Protocol). và nó cho phép điều khiển sâu hơn tới các phần trong bản tin khi có yêu cầu sự chính xác như vậy.

**Server contract:** Đặc tả chi tiết các phương thức của dịch vụ, và được phân phối như là một giao diện trong các ngôn ngữ lập trình như Visual Basic hay Visual C#.

**Policy and binding:** Các chính sách và các kết nối (bindings) mô tả các điều kiện cần có để giao tiếp với một dịch vụ. Các chính sách sẽ bao gồm cả các yêu cầu về bảo mật và các điều kiện khác cần có khi kết nối với một dịch vụ.

## 5.5. RUNTIME SERVICES

- **Throttling behavior:** Điều khiển luồng nhằm quy định xem có bao nhiêu bản tin được xử lý
- **Error behavior:** Hành vi xử lý lỗi quy định những hành động khi lỗi xảy ra trong hệ thống.
- **Metadata behavior:** Quy định xem làm thế nào và khi nào thì các siêu dữ liệu được đưa ra bên ngoài dịch vụ.
- **Instance behavior:** Hành xử thực thể quy định xem có bao nhiêu thực thể của dịch vụ đó được chạy.
- **Transaction behavior:** Hành xử giao dịch cho phép việc rollback các giao dịch nếu xảy ra lỗi
- **Message inspection:** Kiểm tra bản tin đem lại cho dịch vụ khả năng kiểm tra tất cả hay một số phần của bản tin
- **Dispatch behavior:** xác định bản tin được xử lý như thế nào khi được xử lý bởi nền tảng WCF.

## 5.5. RUNTIME SERVICES (2)

- **Concurrency behavior:** Hành xử đồng thời xác định xem việc xử lý thế nào với việc đa luồng của mỗi dịch vụ hay mỗi thực thể của dịch vụ. Hành xử này giúp cho việc điều khiển số lượng luồng có thể truy nhập tới một thực thể của dịch vụ.
- **Parameter filtering:** Khi một bản tin được đưa tới một dịch vụ, sẽ xảy ra một số hành động dựa trên nội dung phần đầu đề của bản tin. Phần lọc tham số sẽ thực hiện lọc các đầu đề bản tin và thực hiện các hành động đặt sẵn dựa trên việc lọc đầu đề bản tin.
- **Message (Bản tin)** Lớp bản tin là tập hợp các kênh, mỗi kênh là một thành phần xử lý bản tin theo một cách nào đó. Một tập các kênh thường được gọi là ngăn xếp kênh. Các kênh làm việc trên bản tin và trên đầu đề của bản tin. Lớp này khác với lớp thực thi dịch vụ chủ yếu bởi sự khác nhau trong việc xử lý nội dung bản tin. Có 2 kênh khác nhau như sau:
  - **Kênh vận chuyển (transport channel):** phụ trách việc đọc và ghi các bản tin từ mạng (network) hoặc từ một số điểm giao dịch bên ngoài)
  - **Kênh điều khiển (control channel):** Thực hiện xử lý bản tin theo giao thức, thông thường làm việc bằng cách đọc và ghi thêm các đầu đề cho bản tin.

## 5.6. HOST AND ACTIVATION (CHÚA VÀ KÍCH HOẠT)

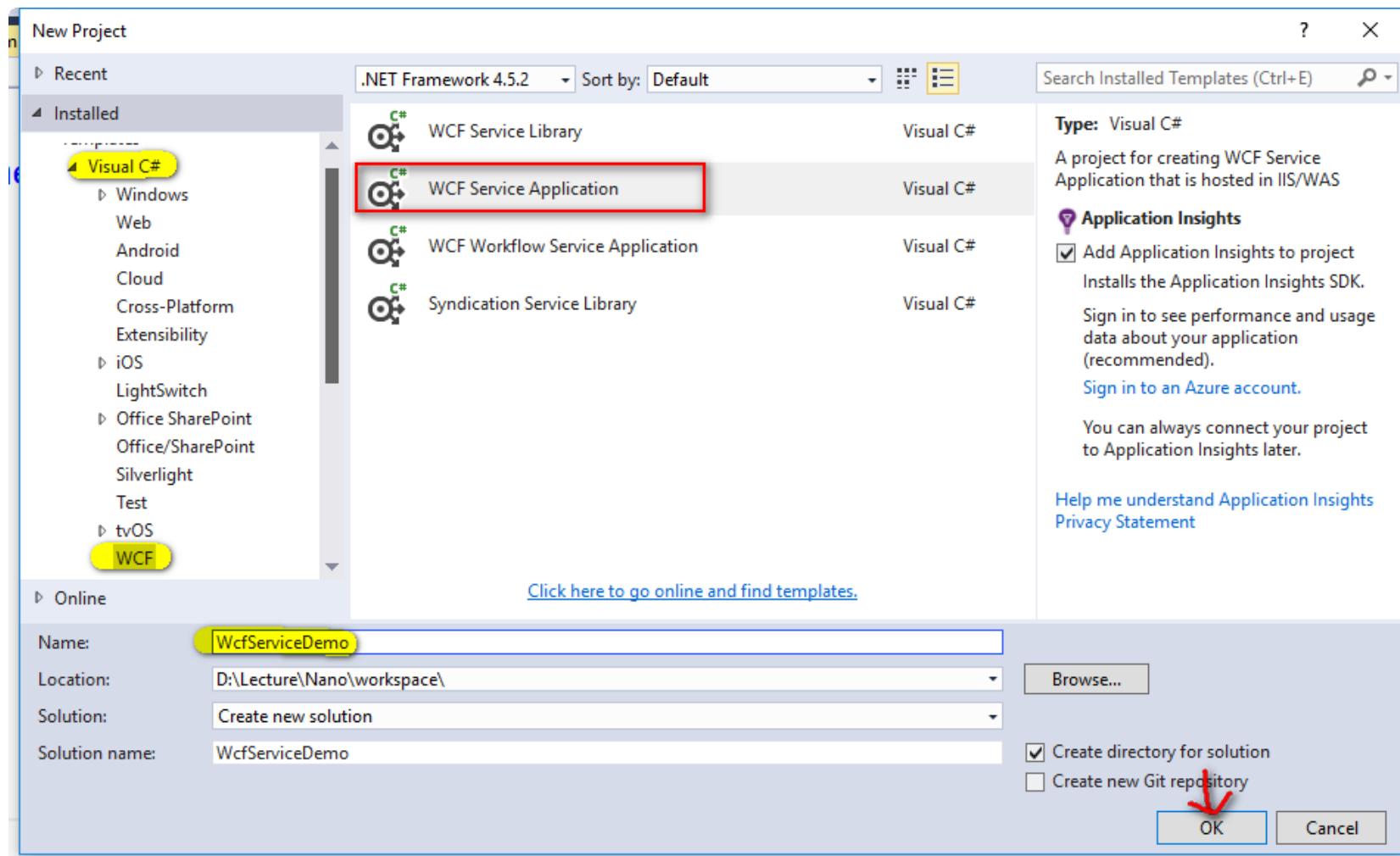
Nhìn chung một dịch vụ thực chất là một chương trình. Cũng giống như các chương trình khác, một dịch vụ cần phải chạy trong một tệp thực thi. Dịch vụ này thường được gọi là dịch vụ tự chúa.

Các dịch vụ còn có thể được chúa, hoặc chạy trong một tệp thực thi được quản lý bởi một agent bên ngoài như IIS hay Windows Activation Services (WAS). WAS cho phép WCF được kích hoạt một cách tự động khi phân phối tới một máy tính có chạy WAS

# 5.7. HƯỚNG DẪN SỬ DỤNG WCF SERVICE

## Bước 1: Tạo project

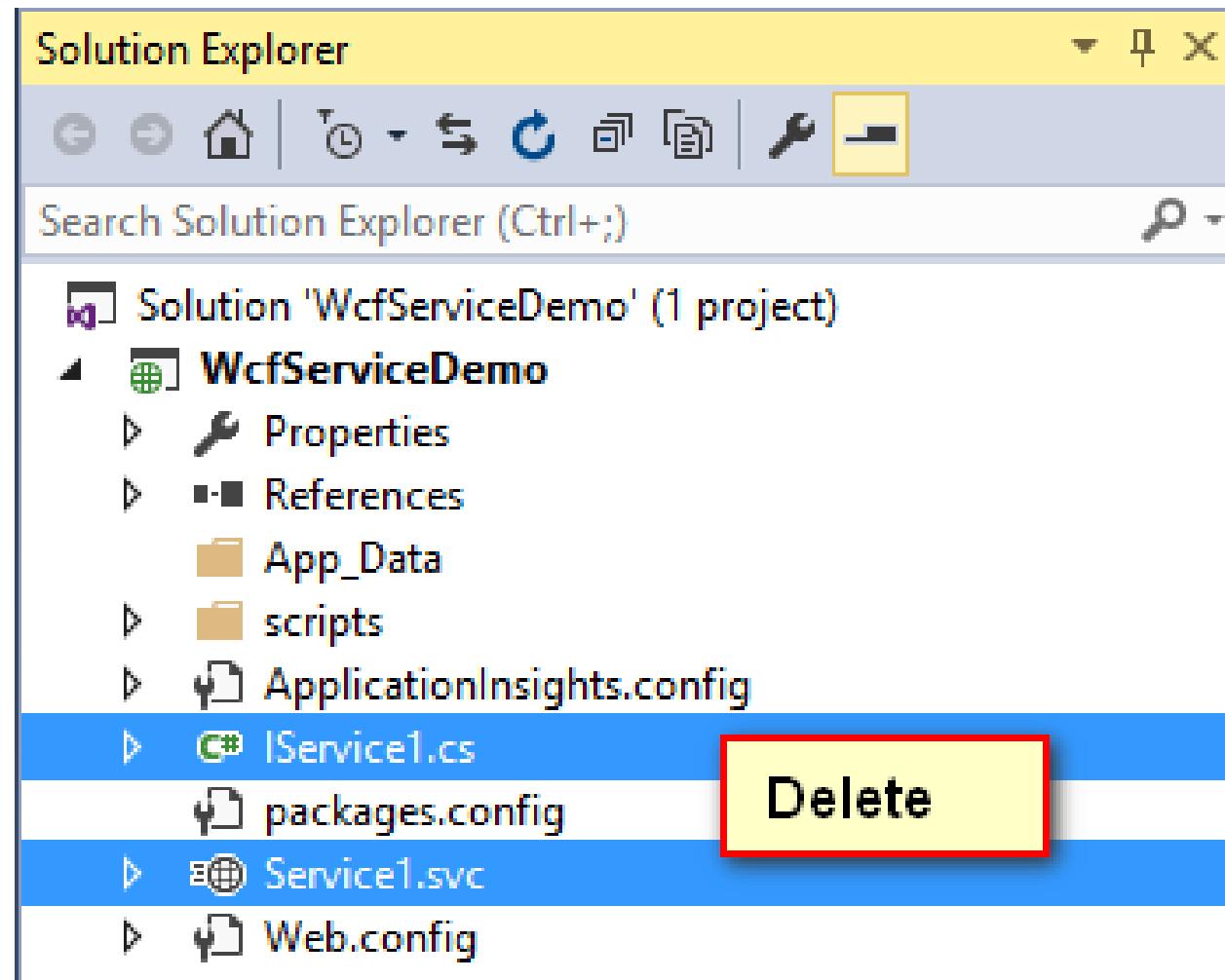
Mở Visual Studio và tạo một project với tên **WcfServiceDemo**



## 5.7. HƯỚNG DẪN SỬ DỤNG WCF SERVICE (2)

### Bước 1: Tạo project

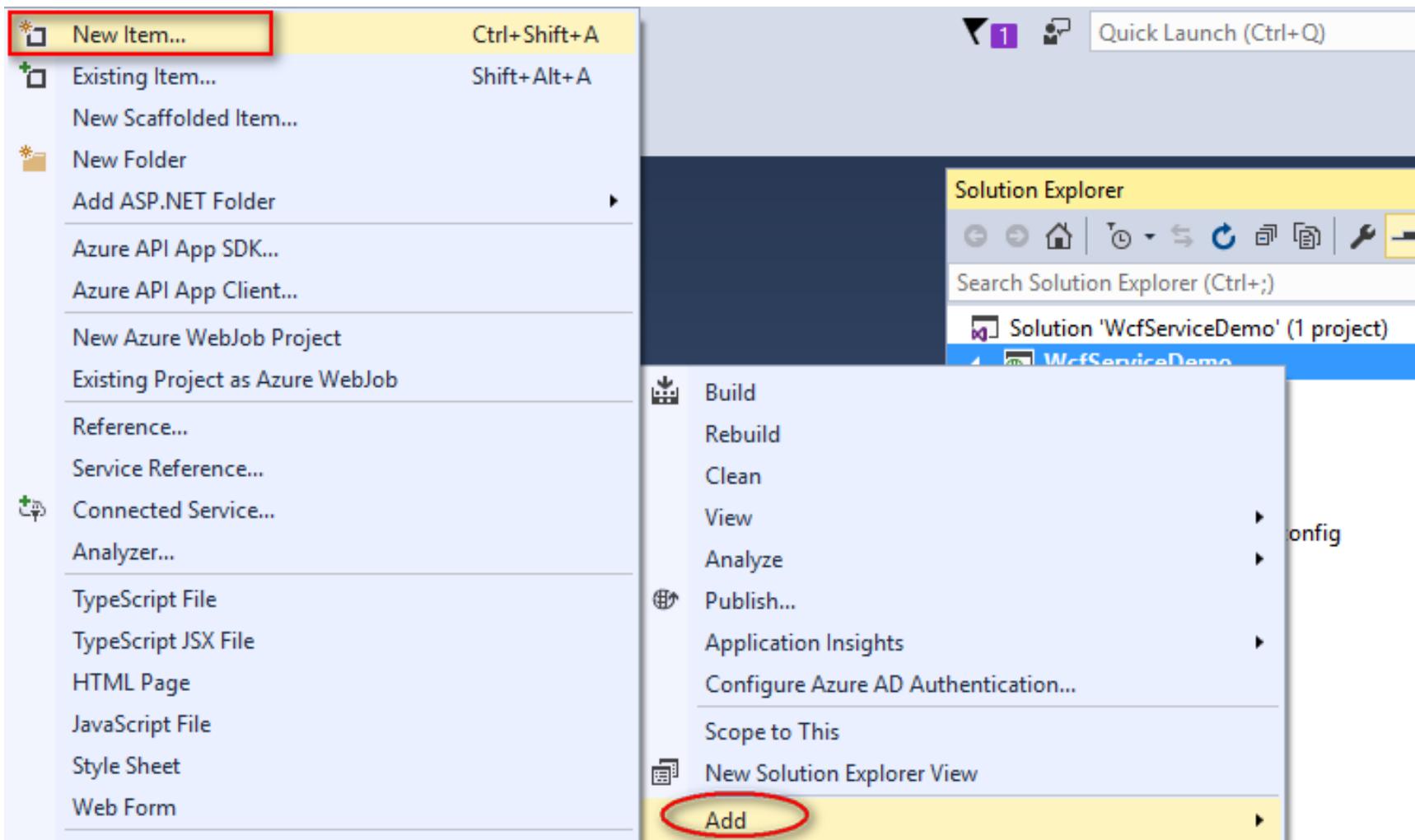
Chúng ta có thể xóa 2 file **IService1.cs** và **Service1.svc** vì không cần sử dụng chúng



# 5.7. HƯỚNG DẪN SỬ DỤNG WCF SERVICE (3)

## Bước 2: Tạo service

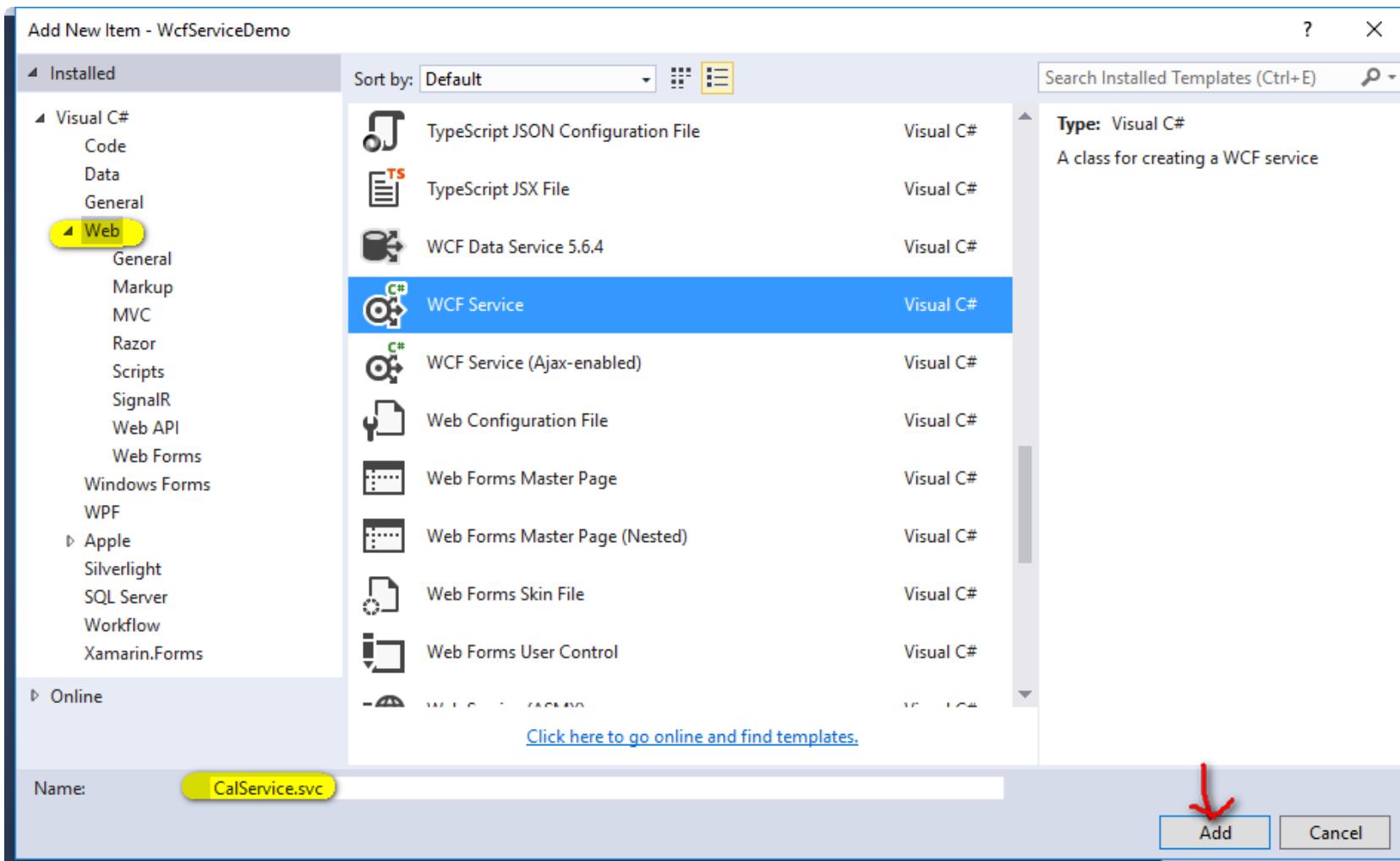
Chuột phải lên project -> chọn Add -> chọn New Item...



# 5.7. HƯỚNG DẪN SỬ DỤNG WCF SERVICE (4)

## Bước 2: Tạo service

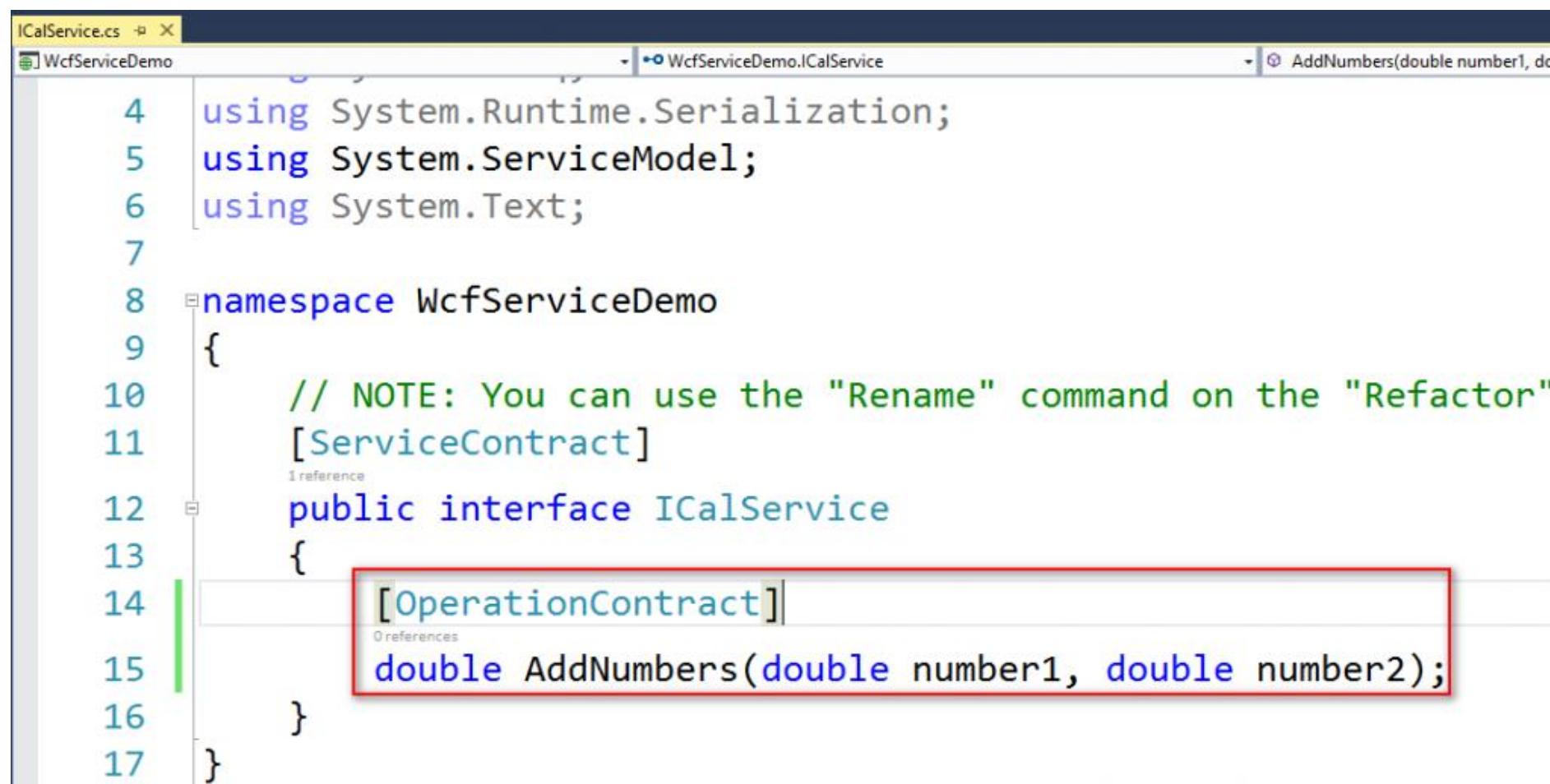
Chọn **WCF Service** -> nhập **CalService** -> chọn **Add**



## 5.7. HƯỚNG DẪN SỬ DỤNG WCF SERVICE (5)

### Bước 2: Tạo service

Mở **ICalService.cs**, xóa hàm “**void DoWork()**” và thay vào đó chúng ta sẽ bổ sung chức năng tính tổng 2 số thực (hàm AddNumbers) như sau



```
ICalService.cs  X
WcfServiceDemo  WcfServiceDemo(ICalService)
AddNumbers(double number1, dou

4  using System.Runtime.Serialization;
5  using System.ServiceModel;
6  using System.Text;
7
8  namespace WcfServiceDemo
9  {
10     // NOTE: You can use the "Rename" command on the "Refactor"
11     [ServiceContract]
12     public interface ICalService
13     {
14         [OperationContract]
15         double AddNumbers(double number1, double number2);
16     }
17 }
```

## 5.7. HƯỚNG DẪN SỬ DỤNG WCF SERVICE (6)

### Bước 3: Cài đặt Service

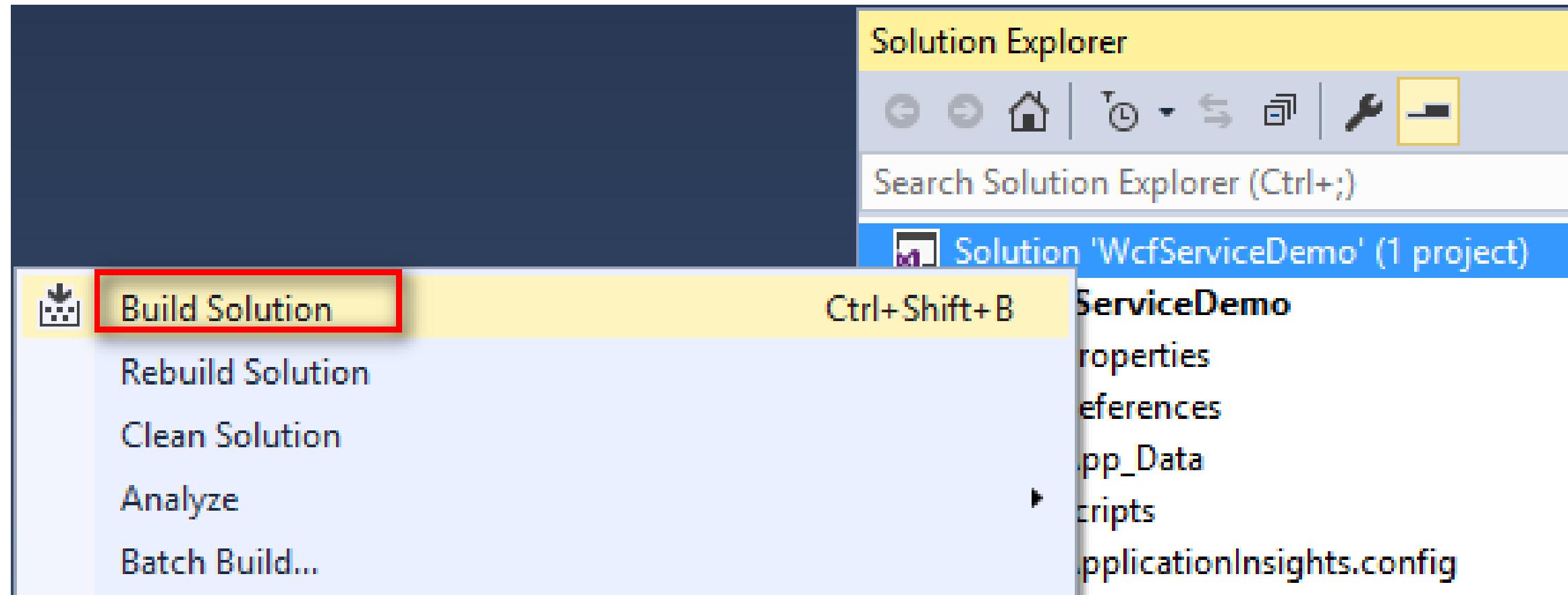
Mở **CalService.svc**, xóa hàm “**public void DoWork()**” và cài đặt xử lý cho hàm **AddNumbers()**.  
Hàm AddNumbers này đã được khai báo tại **ICalService.cs**

```
8  namespace WcfServiceDemo
9  {
10     // NOTE: You can use the "Rename" command on the "Refactor"
11     // NOTE: In order to launch WCF Test Client for testing this
12     public class CalService : ICalService
13     {
14         public double AddNumbers(double number1, double number2)
15         {
16             return number1 + number2;
17         }
18     }
19 }
```

## 5.7. HƯỚNG DẪN SỬ DỤNG WCF SERVICE (7)

### Bước 4: Build

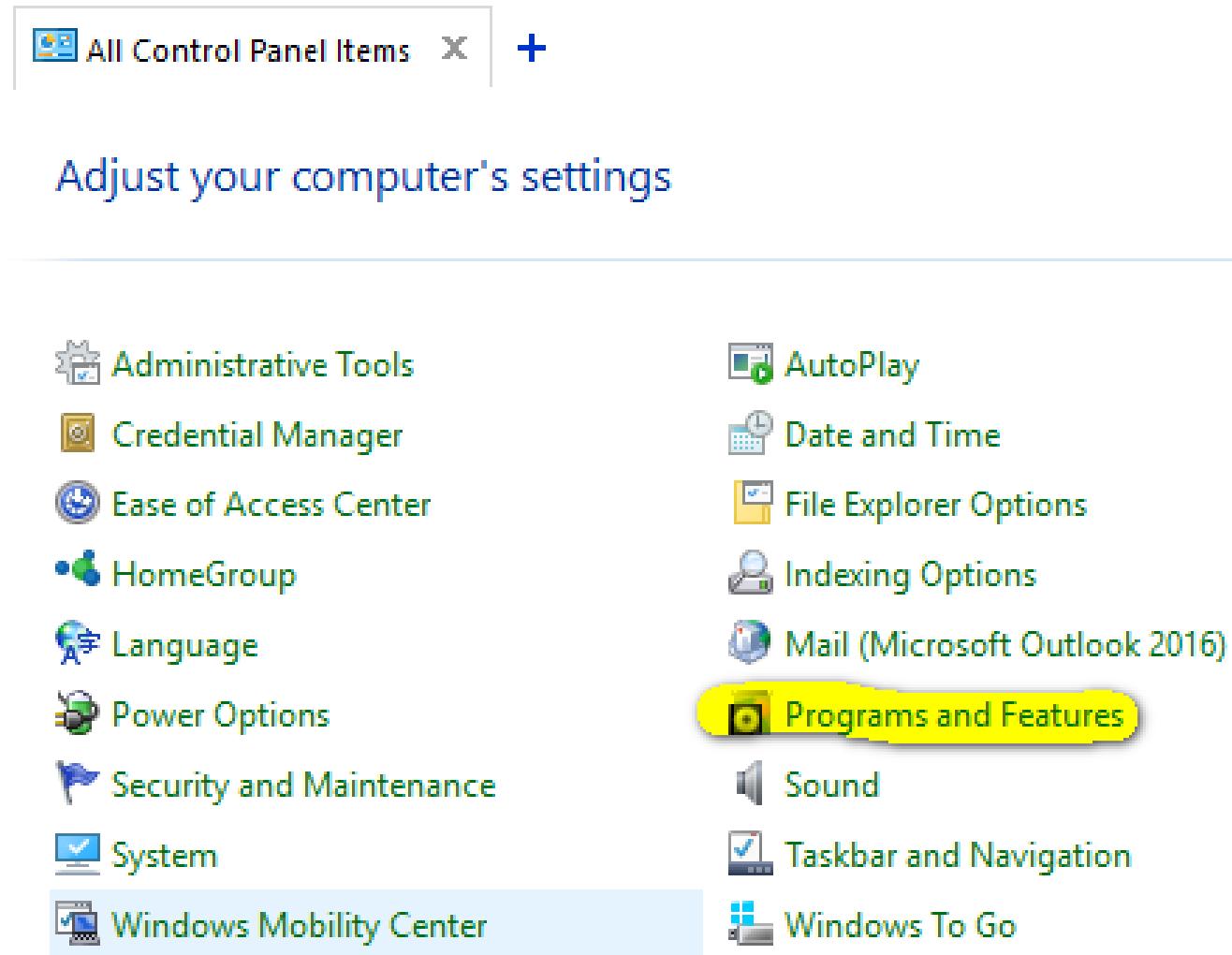
Trong Solution Explorer, chuột phải lên **Solution 'WcfServiceDemo'** -> chọn **Build Solution**



## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS

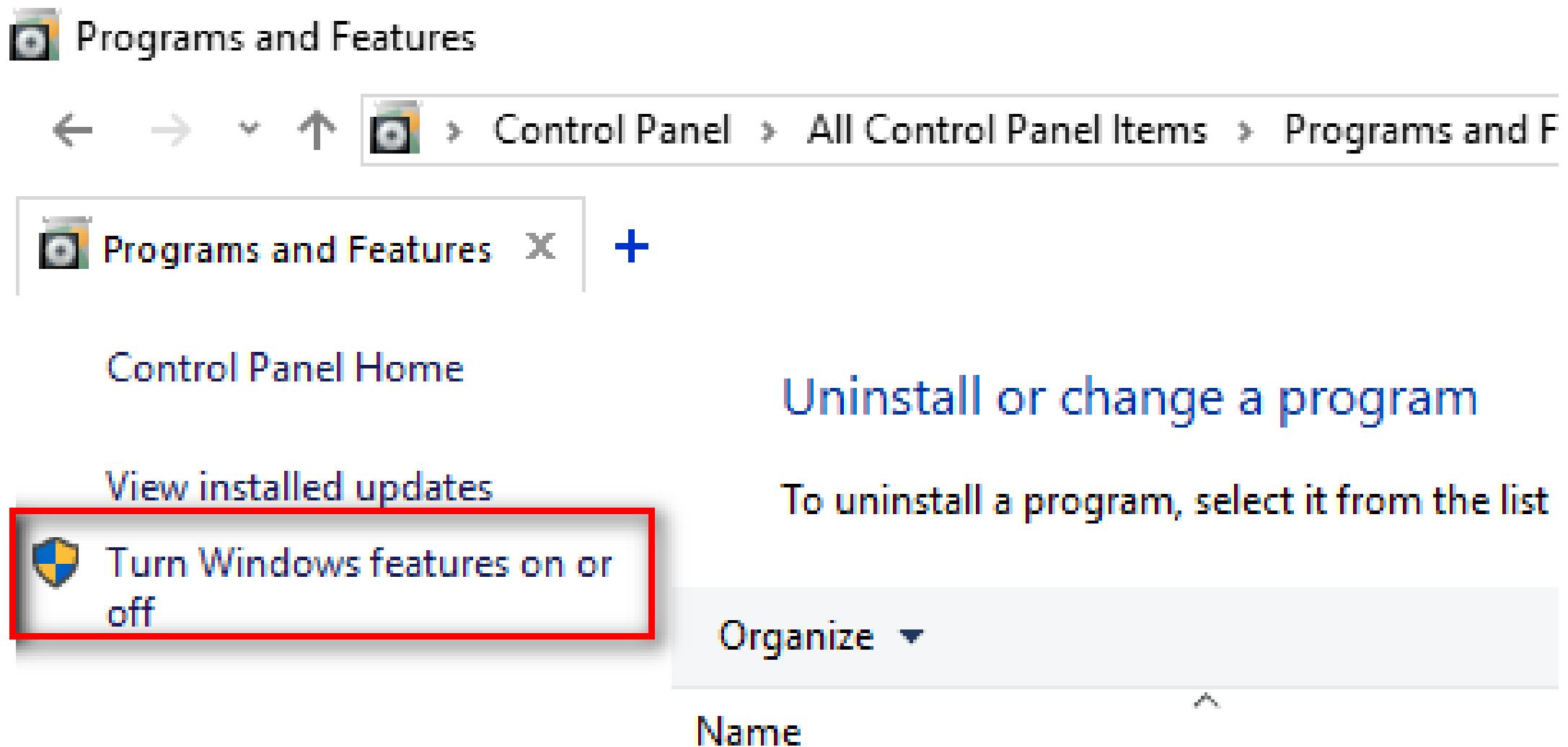
**Bước 1: Cài đặt IIS (Nếu máy tính của bạn đã cài IIS thì bỏ qua bước này)**

Mở Control Panel -> chọn Programs and Features (trong hình tôi đang sử dụng Windows 10)



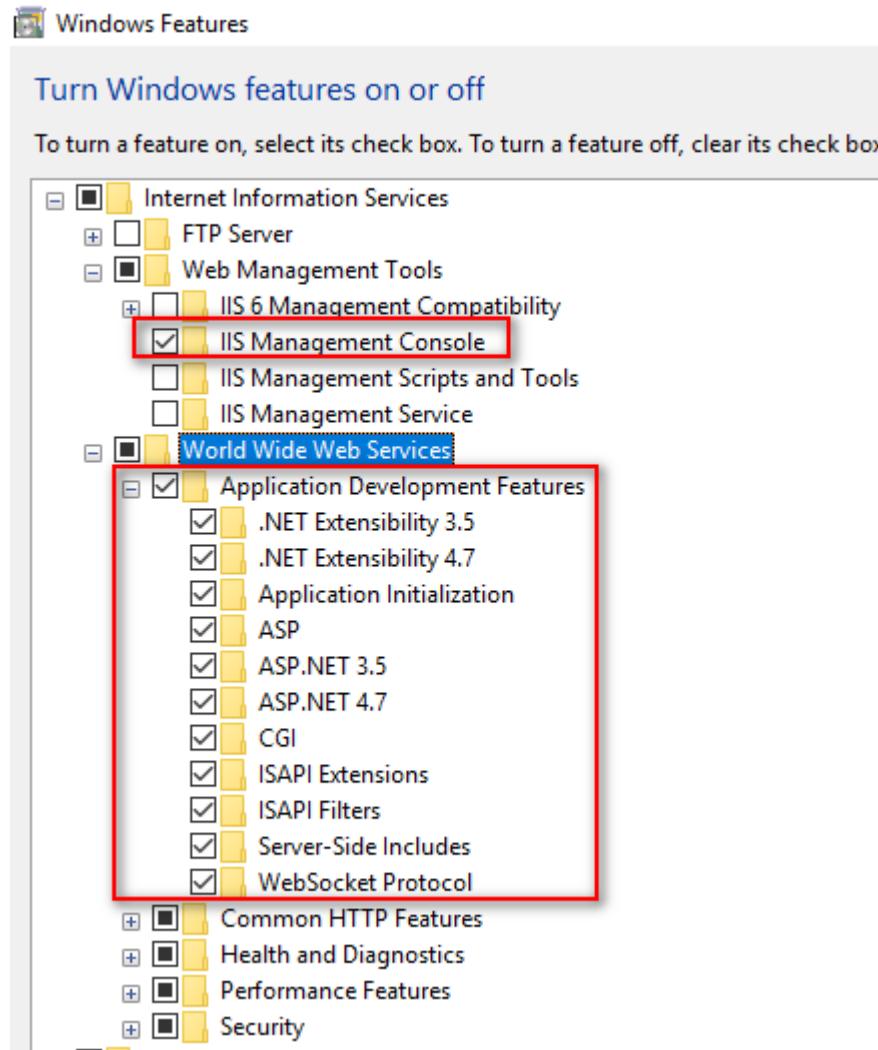
## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (2)

Chọn Turn Windows features on or off



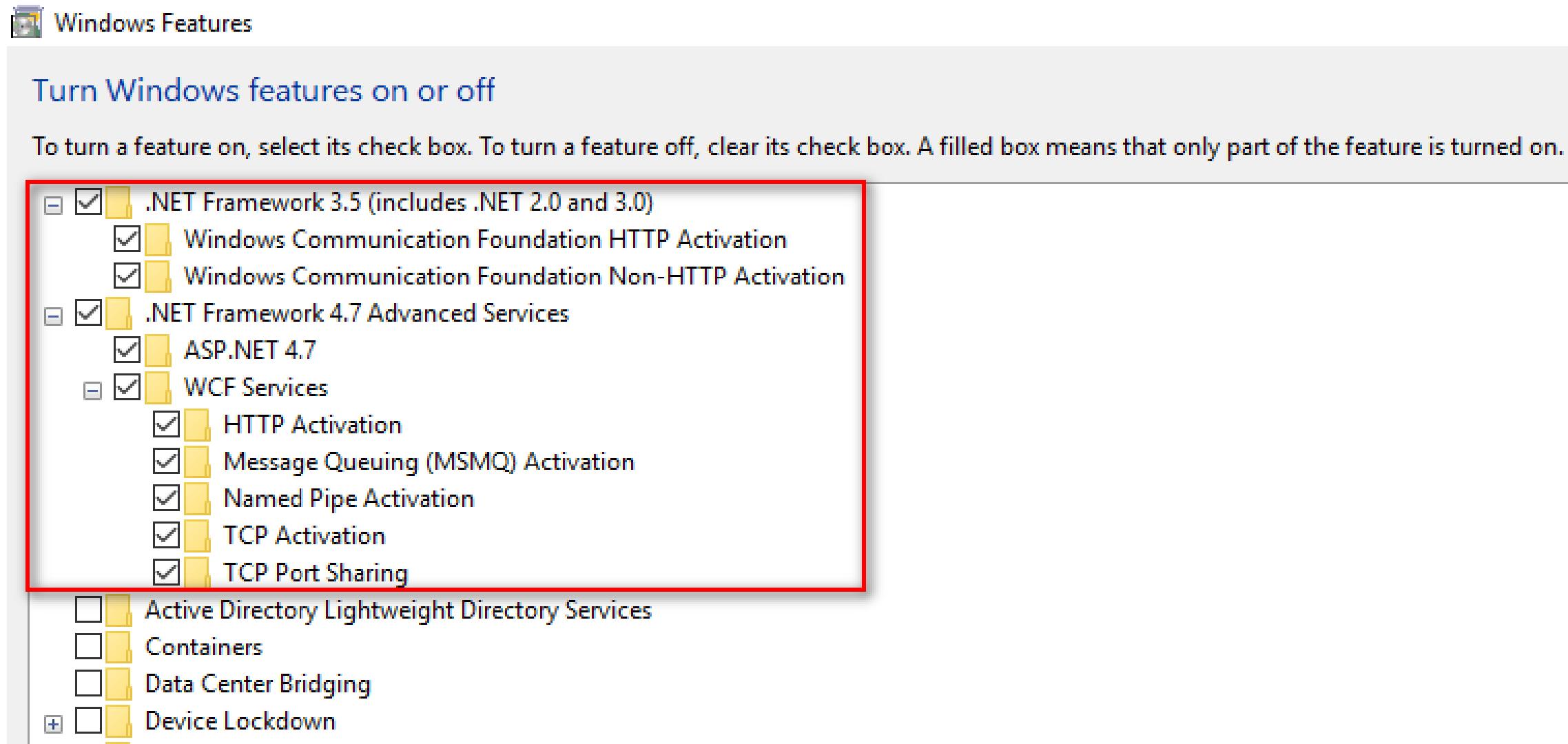
## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (3)

Chọn **Internet Information Services** như hình bên dưới



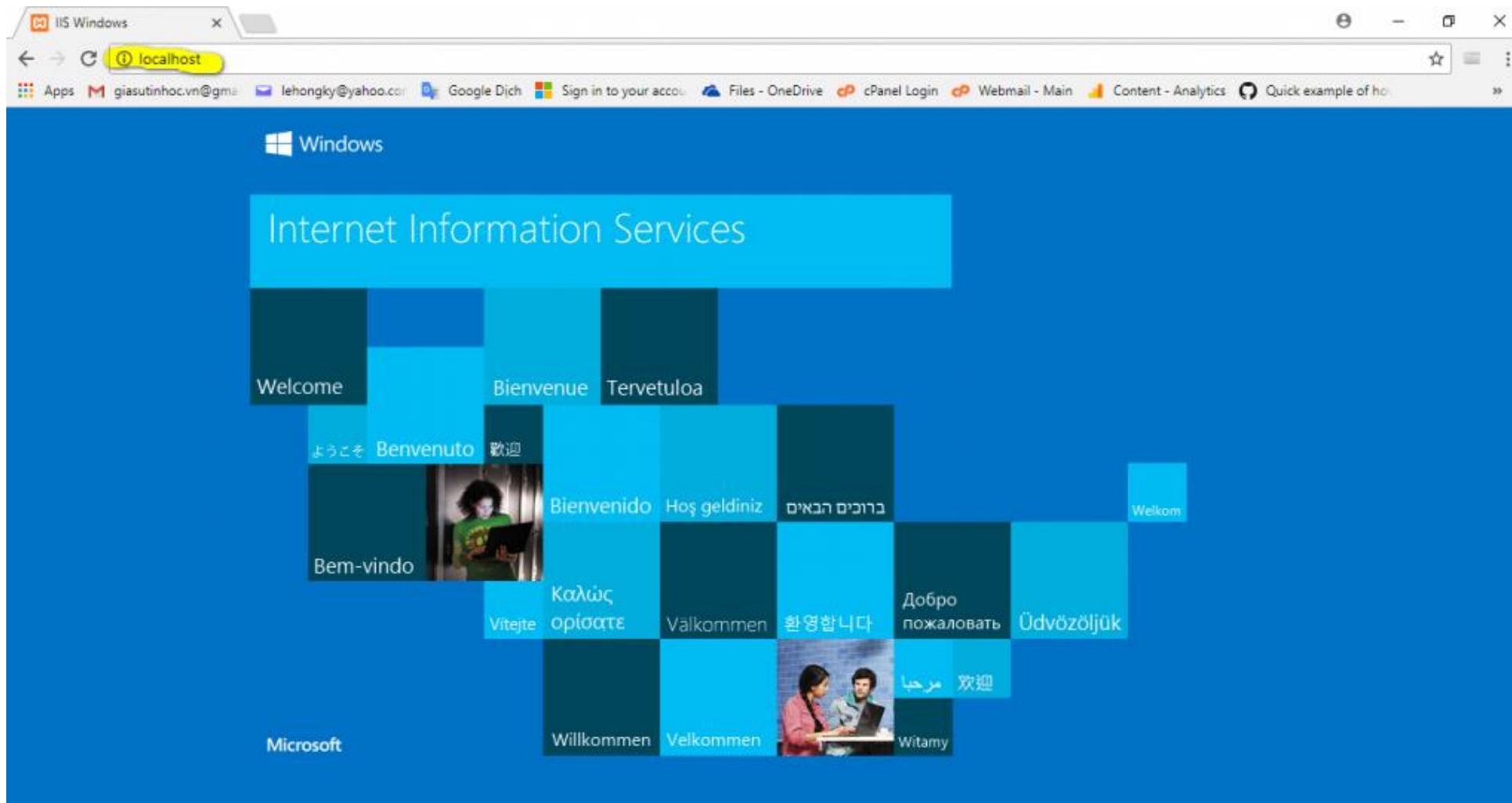
## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (4)

Ngoài ra, chúng ta cũng phải cài **.NET Framework** như hình sau



## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (5)

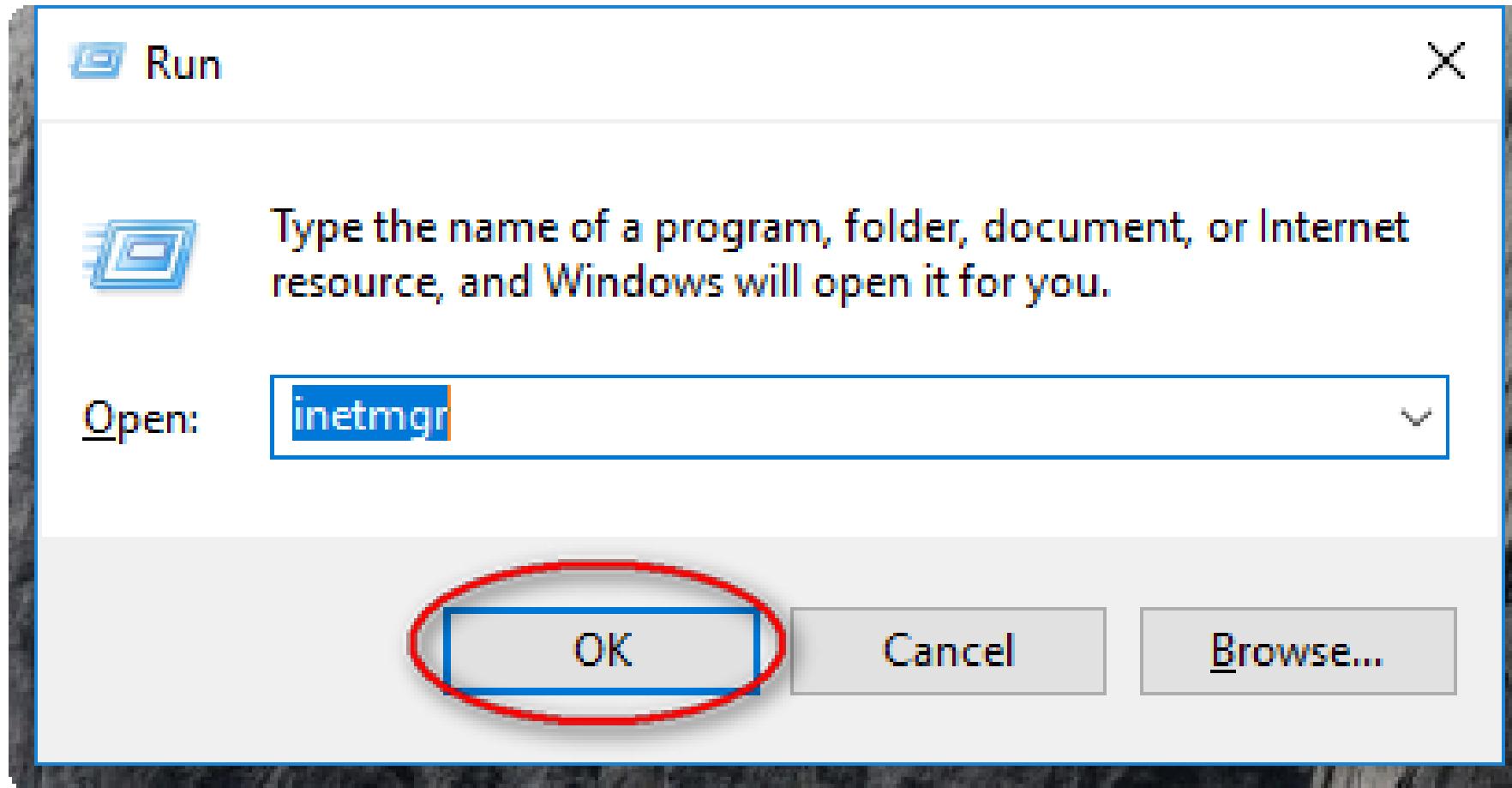
Kiểm tra IIS bằng cách mở trình duyệt và nhập vào **http://localhost**. Nếu các bạn thấy như hình bên dưới nghĩa là việc cài đặt IIS đã thành công.



## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (6)

### Bước 2: Tạo Application

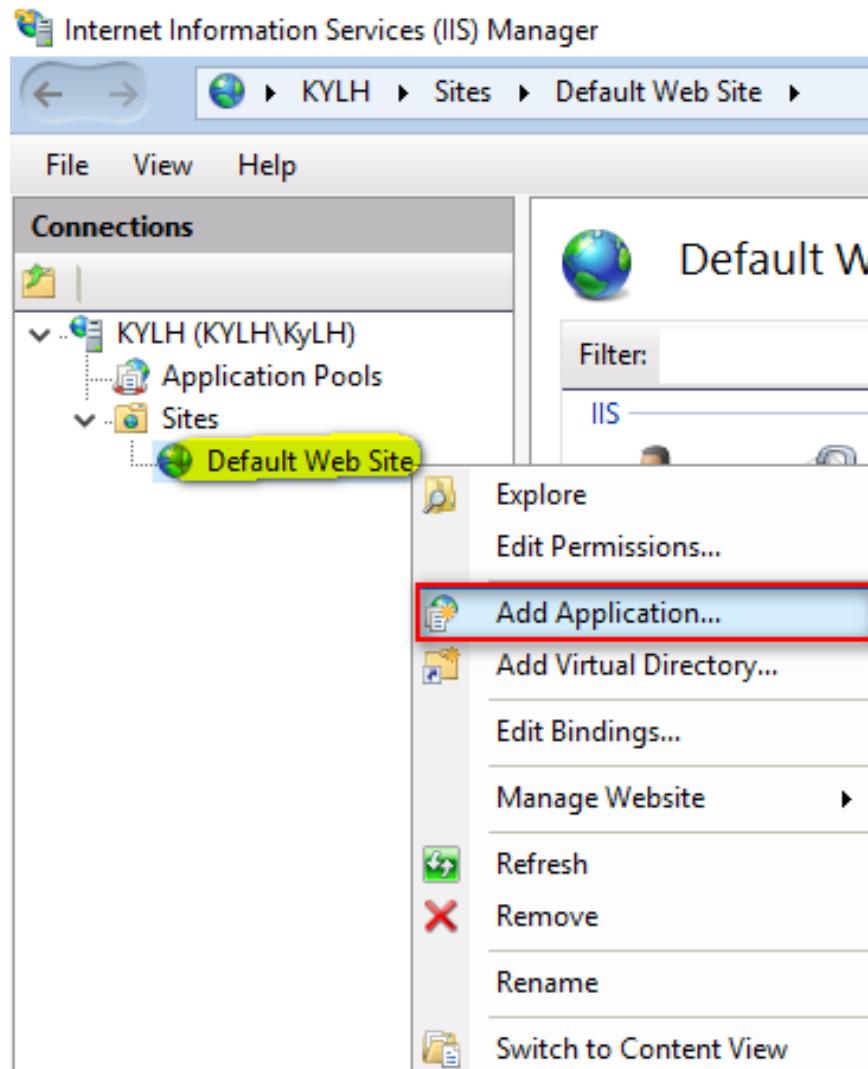
Để khởi động IIS bằng dòng lệnh, chúng ta ở hộp thoại **Run** và nhập **inetmgr** -> chọn **OK**



## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (7)

### Bước 2: Tạo Application

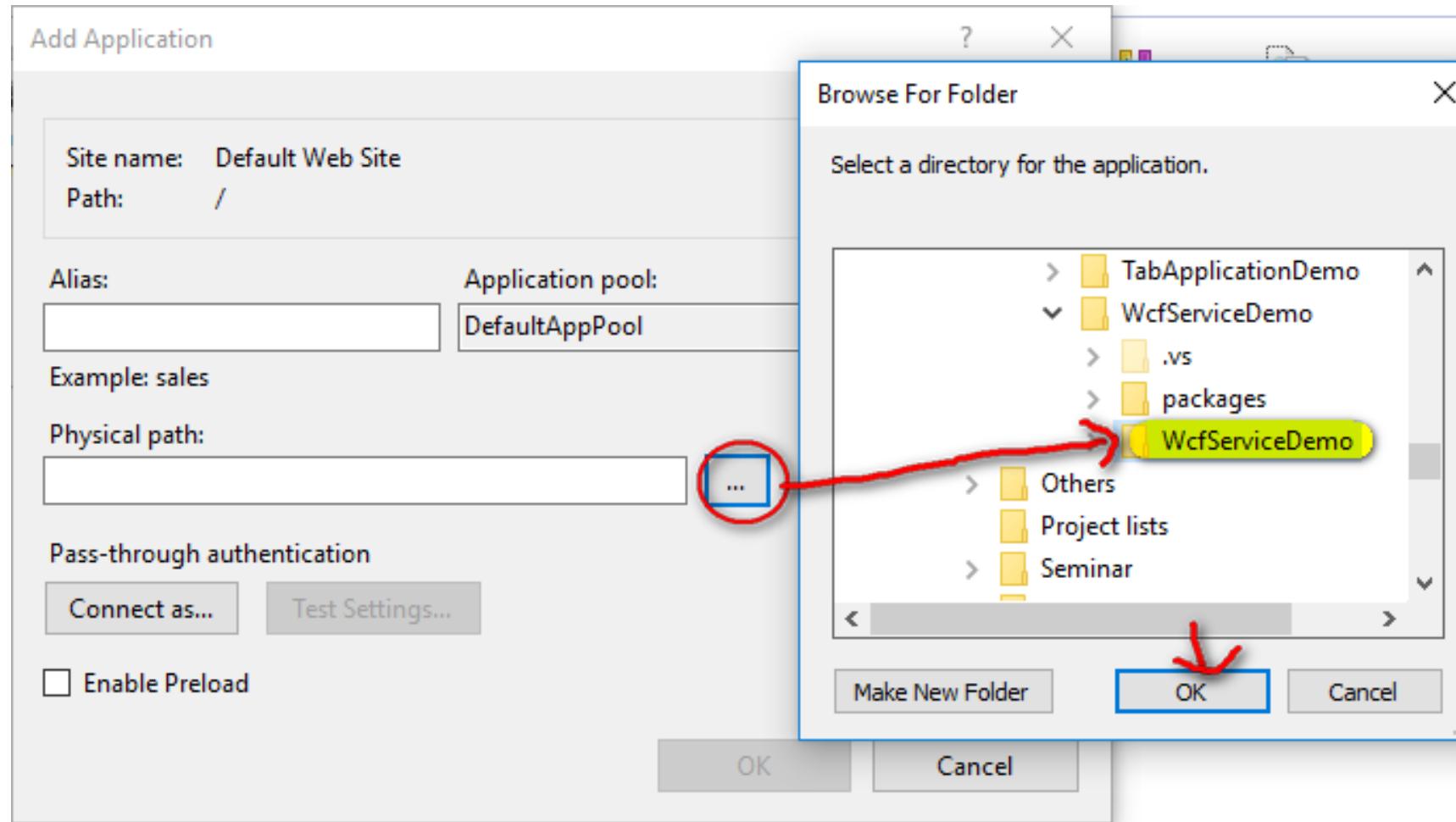
Chuột phải lên Default Web Site -> chọn Add Application...



## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (8)

### Bước 2: Tạo Application

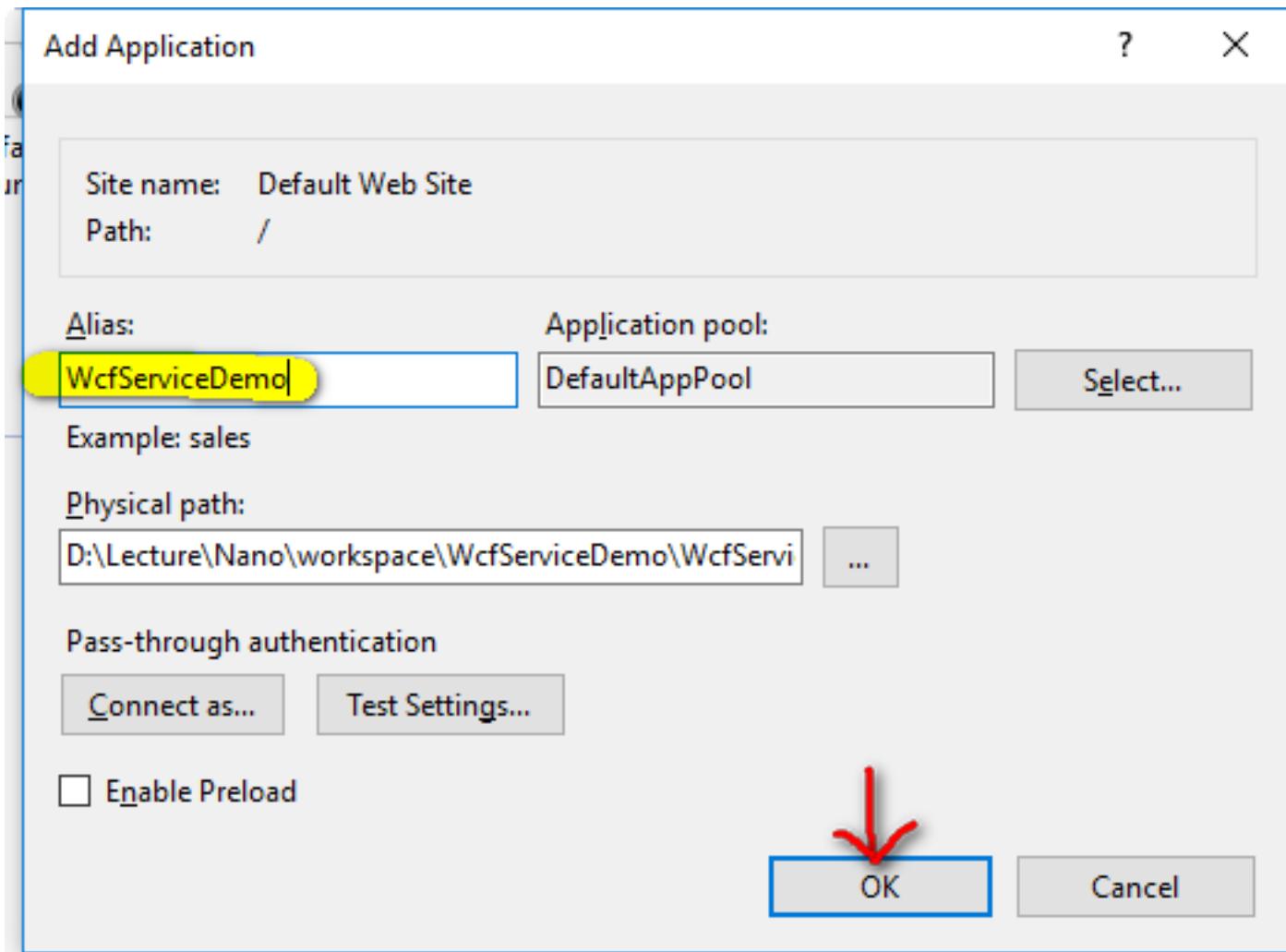
Ở hộp thoại Add Application, chúng ta chỉ định thư mục chứa WCF Service project (**WcfServiceDemo**) mà chúng ta đã thực hiện ở phần trên (5 bước tạo WCF Service)



## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (9)

### Bước 2: Tạo Application

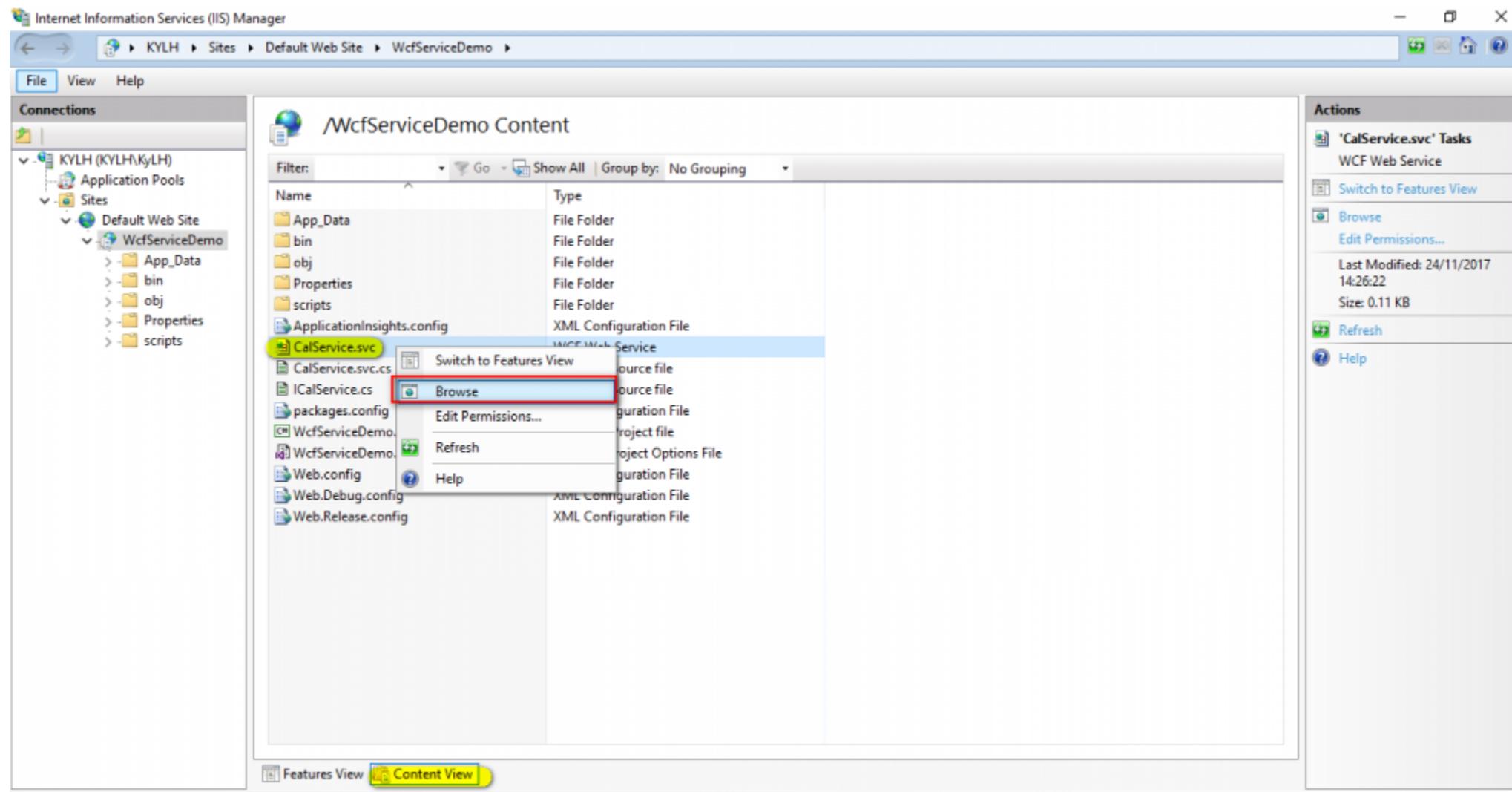
Nhập **WcfServiceDemo** tại **Alias** và chọn **OK**



## 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (10)

### Bước 2: Tạo Application

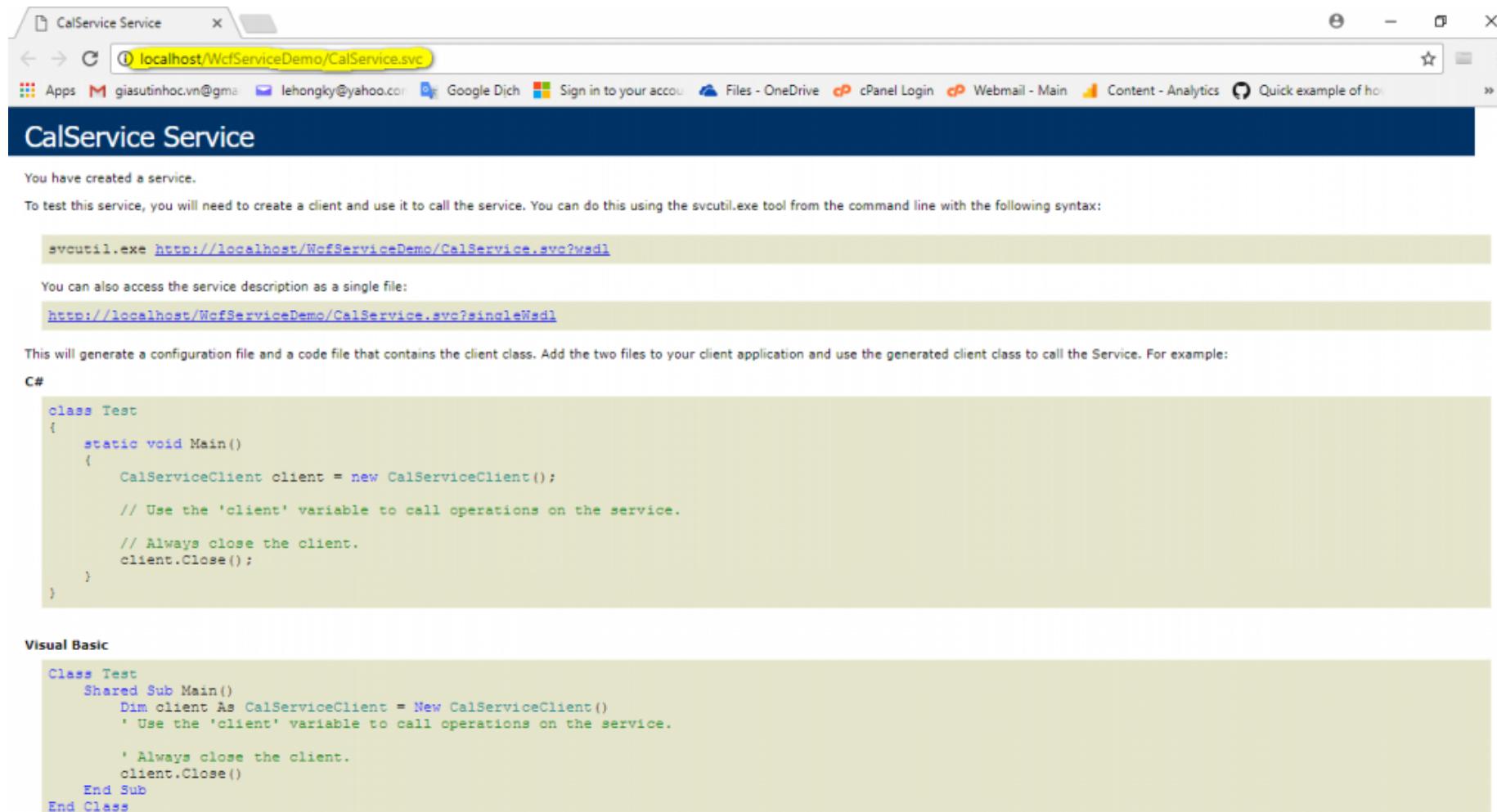
Chuột phải lên **CalService.svc** -> chọn **Browse**



# 5.8. TRIỂN KHAI WCF SERVICE TRÊN IIS (11)

## Bước 2: Tạo Application

Chúng ta sẽ nhìn thấy như hình sau (Lưu ý trên thanh địa chỉ của trình duyệt, nơi mà chúng tôi tô màu vàng. Đây sẽ là URL mà chúng ta sẽ dùng để gọi Service tại phía Client)



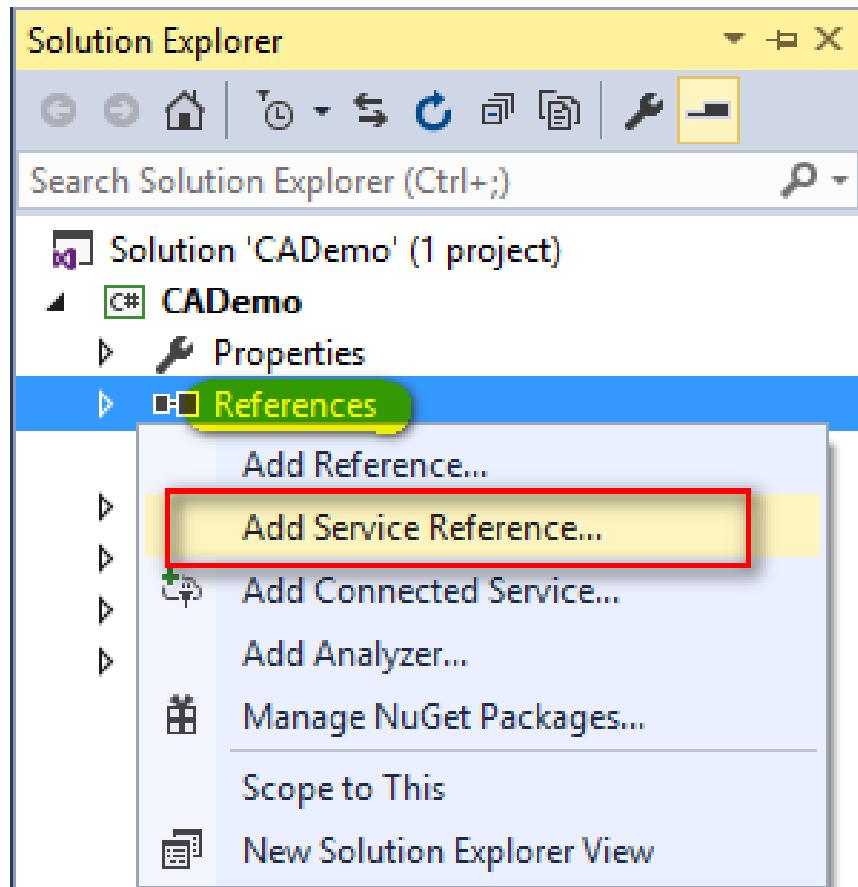
# 5.9. SỬ DỤNG WCF SERVICE

## Bước 1: Tạo Console Application

Mở Visual Studio, tạo một project dạng Console Application và đặt tên CADemo

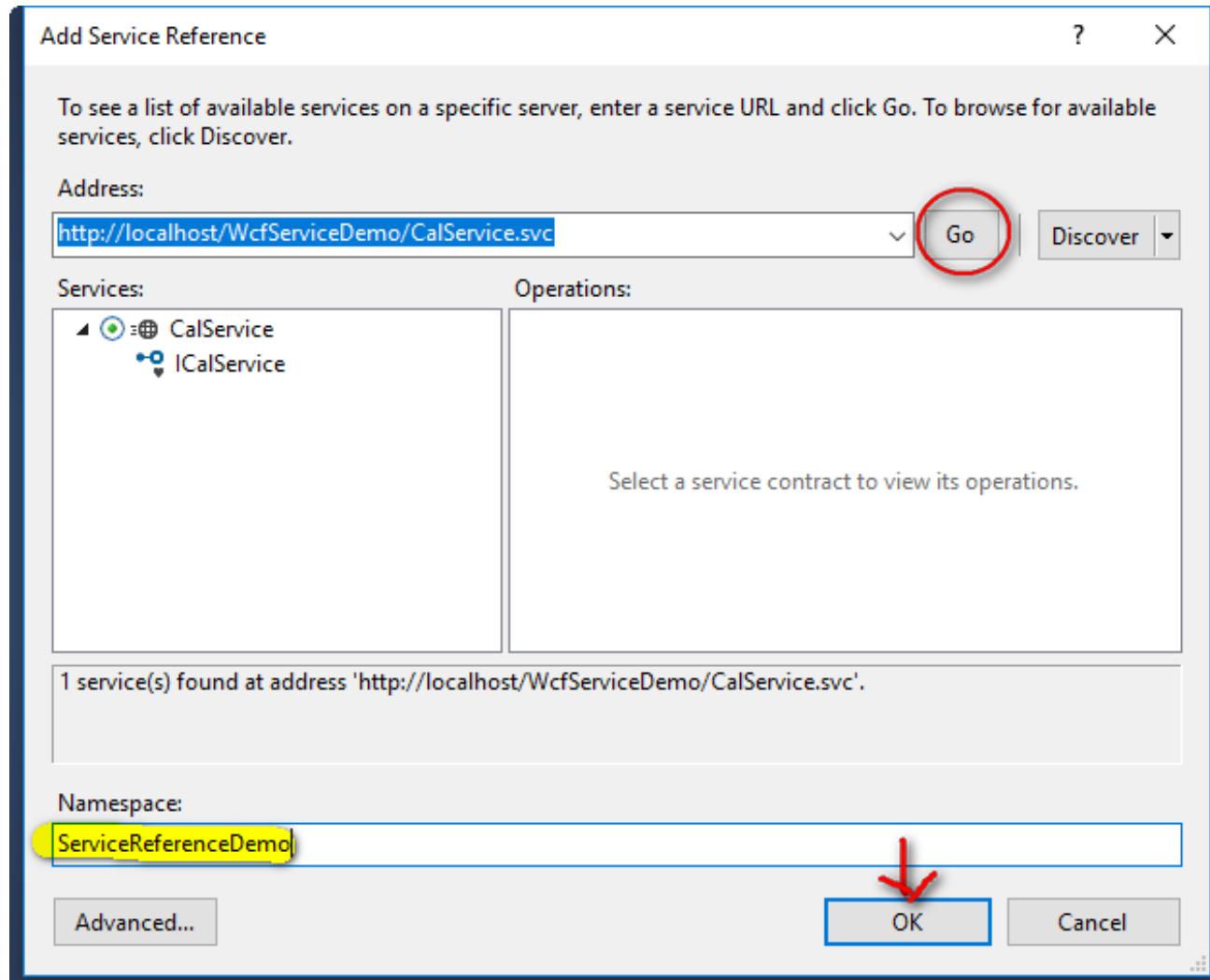
## Bước 2: Thêm Service

Chuột phải lên **References** -> chọn **Add Service Reference...**



## 5.9. SỬ DỤNG WCF SERVICE (2)

Nhập **http://localhost/WcfServiceDemo/CalService.svc** tại **Address**, chọn **Go** và nhập **ServiceReferenceDemo** tại **Namespace**



## 5.9. SỬ DỤNG WCF SERVICE (3)

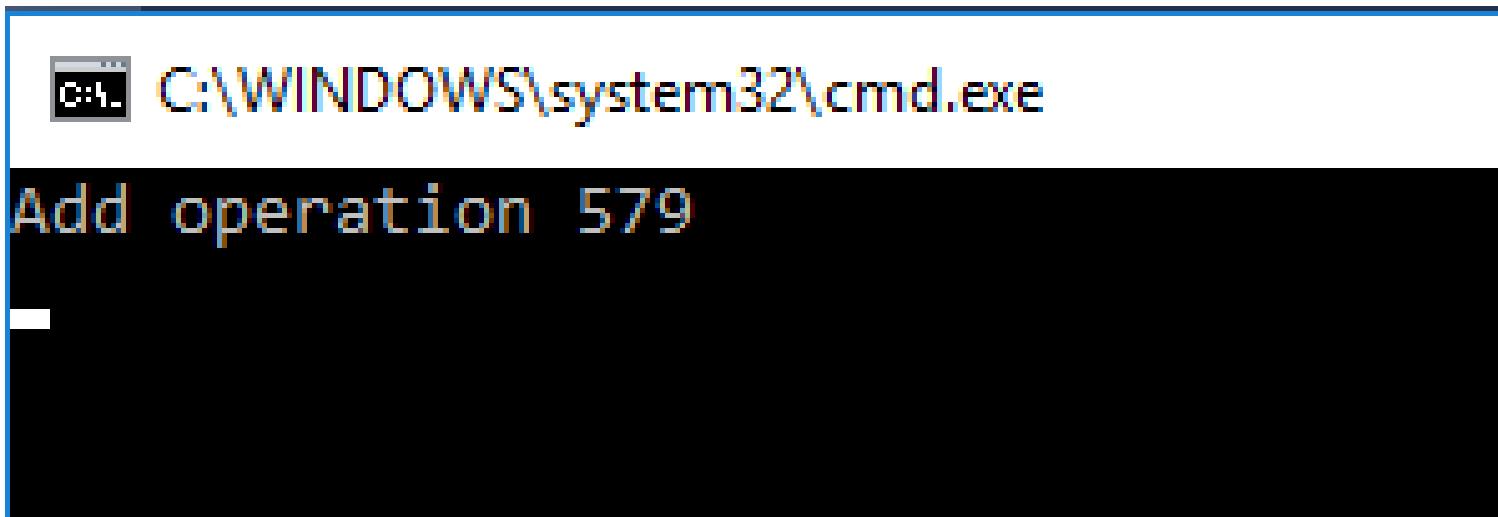
Bước 3: Mở Program.cs và thêm đoạn code sau

```
5  using System.Threading.Tasks;
6  using CADemo.ServiceReferenceDemo;
7  namespace CADemo
8  {
9      class Program
10     {
11         static void Main(string[] args)
12         {
13             CalServiceClient cal = new CalServiceClient();
14             double ret = cal.AddNumbers(123, 456);
15             Console.WriteLine("Add operation {0}", ret);
16             Console.ReadLine();
17         }
18     }
19 }
```

## 5.9. SỬ DỤNG WCF SERVICE (4)

### Bước 4: Chạy ứng dụng

Để chạy ứng dụng, chúng ta nhấn Ctrl + F5. Trong hình số 579 là tổng của 123 và 456



The screenshot shows a Windows Command Prompt window with a blue title bar and a white body. The title bar displays 'C:\WINDOWS\system32\cmd.exe'. The body of the window shows the command 'Add operation 579' entered by the user. The text is colored in a standard terminal style where the command is blue and the output is white.

```
C:\WINDOWS\system32\cmd.exe
Add operation 579
```

## 6.1. LẬP TRÌNH PARALLEL

Sử dụng kỹ thuật chạy song song với lớp Parallel để chạy nhiều thread (lập trình đa luồng), trên luồng chạy các Task.

Lớp **Parallel** thuộc namespace **System.Threading.Tasks**, nó trùu tượng hóa các **thread**, lớp này có phương thức tĩnh **Parallel.For**, **Parallel.ForEach** để thực hiện vòng lặp **for** và **foreach** để chạy song song các tác vụ. Nếu với **for** và **foreach** trong C#, thì vòng lặp đó chạy trong một thread, nhưng với **Parallel** nó sử dụng đa tác vụ, đa tiến trình để thực hiện nội dung lặp. Ngoài ra là **Parallel.Invoke** để thực hiện một **Action** có khả năng chạy song song.

Parallel.For có nhiều quá tải, cú pháp bản đơn giản như sau:

```
ParallelLoopResult result = Parallel.For(i1, i2, task);
```

## 6.1. LẬP TRÌNH PARALLEL (2)

Vòng lặp chạy (biến chạy) từ số nguyên **i1** đến **i2**, mỗi lần lặp nó sẽ thực hiện Action task

**task** là một delegate, kiểu **Action<int>** có nghĩa nó làm phương thức trả về void, có một tham số kiểu int, tham số này là biến chạy. Ví dụ đây là một action phù hợp cho **Parallel.For**

```
Action<int> action = (int x) => {  
    // Doing somthing here ...  
};
```

**result** đối tượng lớp **ParallelLoopResult** trả về từ **Paralell.For**, thuộc tính **ParallelLoopResult.IsCompleted** cho biết vòng lặp đã được duyệt qua hết, tất cả các task đã khởi chạy.

```
class Program
{
    // In thông tin, Task ID và thread ID đang chạy
    public static void PrintInfo(string info) =>
        Console.WriteLine($"{info, 10}    task:{Task.CurrentId,3}    " +
                          $"thread: {Thread.CurrentThread.ManagedThreadId}");

    // Phương thức phù hợp với Action<int>, được làm tham số action của Parallel.For
    public static void RunTask(int i) {
        PrintInfo($"Start {i,3}");
        Task.Delay(1000).Wait();           // Task dừng 1s - rồi mới chạy tiếp
        PrintInfo($"Finish {i,3}");
    }

    public static void ParallelFor() {
        ParallelLoopResult result = Parallel.For(1, 20, RunTask);    // Vòng lặp tạo ra 20 lần chạy RunTask
        Console.WriteLine($"All task start and finish: {result.IsCompleted}");
    }
    static void Main(string[] args)
    {
        ParallelFor();

        Console.WriteLine("Press any key ..."); Console.ReadKey();
    }
}
```

- Lệnh **Parallel.For** khởi chạy song song nhiều tác vụ (thời điểm bắt đầu của mỗi tác vụ không giống nhau, có những tác vụ đã kết thúc thì tác vụ sau mới chạy, nó có thể phụ thuộc vào tài nguyên hệ thống RAM, CPU ...)
- Một task nó có chạy trên một thread nào đó (chứ không phải mỗi task một thread), một thread có thể sử dụng bởi nhiều task. Ví dụ nhìn vào kết quả, task với  $i = 1$ , và task với  $id = 15$  chạy trên một thread
- Khi tất vòng lặp duyệt qua hết, có nghĩa là đã khởi chạy hết các tác vụ, nếu tất cả các tác vụ trả về (khi chạy xong hàm Action, hoặc ngay khi gọi Action nếu Action là **async** - hãy xem kỹ phần **async - await** ở phần trước) thì kết quả lưu vào **result** với **result.IsCompleted** là **true**

## 6.2. PARALLEL.FOR KHI ACTION LÀ ASYNC

Để ý, bản thân vòng lặp **Parallel.For**, khi các Action chạy, mặc dù chúng chạy trên những Task và Thread, nhưng khi tất cả các Action hoàn thành thì vòng lặp mới hoàn thành. Ở ví dụ trên, bạn thấy tất cả đều Finish mới trả về kết quả **result** (*All task start and finish: True*)

Điều này, lại dẫn đến **Parallel.For** khóa(block) thread gọi nó. Để không bị khóa, có thể chuyển các Action là **async**

```
public static async void RunTask(int i) {  
    PintInfo($"Start {i,3}");  
    // Task.Delay(1000).Wait();           // Task dừng 1s - rồi mới chạy tiếp  
    await Task.Delay(1);               // Task.Delay là một async nên có thể await, RunTask chuyển điểm gọi nó tại đây  
    PintInfo($"Finish {i,3}");  
}
```

Kết quả khi chạy, vòng lặp **Parallel.For** nó trả về ngay khi tất cả các Task đã khởi chạy - (*chứ không cần chờ tất cả các task chạy và kết thúc như trường hợp trước*).

Khi **Parallel.For** hoàn thành, có một số Task đã kết thúc có những Task vẫn đang chạy.

## 6.3. PARALLEL.FOREACH CHẠY SONG SONG TÁC VỤ

Với Parallel.ForEach cũng là vòng lặp để chạy nhiều tác vụ, nhưng nó duyệt qua các Collection như Mảng, List ... tương tự như vòng lặp foreach. Cú pháp cơ bản như sau:

```
ParallelLoopResult result = Parallel.ForEach(source, RunTask);
```

Trong đó source là một Collection như mảng, List. RunTask là Action, có 1 tham số có kiểu giống kiểu phần tử trong source, giá trị tham số này là giá trị phần tử trong source trong mỗi vòng lặp. Ví dụ:

```
public static async void RunTask(string s) {
    PintInfo($"Start {s},10");
    await Task.Delay(1); // Task.Delay là một async nên có thể await, RunTask chuyển điểm gọi nó tại đây
    PintInfo($"Finish {s},10");
}

public static void ParallelFor() {

    string[] source = new string[] {"FPTEducation1", "FPTEducation2", "FPTEducation3",
                                    "FPTEducation4", "FPTEducation5", "FPTEducation6",
                                    "FPTEducation7", "FPTEducation8", "FPTEducation9"};
    // Dùng List thì khởi tạo
    // List<string> source = new List<string>();
    // source.Add("xuanthulab1");

    ParallelLoopResult result = Parallel.ForEach(
        source, RunTask
    );

    Console.WriteLine($"All task started: {result.IsCompleted}");
}
static void Main(string[] args)
{
    ParallelFor();
    Console.WriteLine("Press any key ...");
    Console.ReadKey();
}
```

## 6.4. PARALLEL.INVOKE CHẠY SONG SONG NHIỀU LOẠI TÁC VỤ (PHƯƠNG THỨC)

Với các vòng lặp ở trên, thì các tác vụ định nghĩa trọng một Action, nhưng nếu muốn chạy song song nhiều loại Action (phương thức) một lúc thì dùng Paralell.Invoke

*Parallel.Invoke(action1, action2, action3);*

Trong đó tham số là các Action

```
public static void PintInfo(string info) =>
Console.WriteLine($"{info, 10}    task:{Task.CurrentId,3}    "
+ $"thread: {Thread.CurrentThread.ManagedThreadId}");

public static async void RunTask(string s)  {
    PintInfo($"Start {s,10}");
    await Task.Delay(1);
    PintInfo($"Finish {s,10}");
}

public static void actionA() {
    PintInfo($"Finish {"ActionA",10}");
}

public static void actionB() {
    PintInfo($"Finish {"ActionB",10}");
}

public static void ParallelInvoke() {
    Action action1  = () => {
        RunTask("Action1");
    };

    Parallel.Invoke(action1, actionA, actionB);
}
```

```
static void Main(string[] args)
{
    ParallelInvoke();
    Console.WriteLine("Press any key ...");
    Console.ReadKey();
}
```

## 7.1. LẬP TRÌNH DYNAMIC

Ngôn ngữ lập trình C# cho ra phiên bản 4.0 đã cung cấp kỹ thuật lập trình mới đó là sử dụng Dynamic programming, và thêm từ khóa mới “dynamic”.

Dynamic programming là nội dung quan trọng mà ngôn ngữ lập trình C# giới thiệu tại phiên bản 4.0, đi kèm với môi trường thực thi .Net và môi trường Dynamic Language Runtime hay gọi tắt là DLR.

Dynamic Language Runtime có tác dụng cung cấp những dịch vụ vào Common Language Runrim (CLR) để hỗ trợ tính năng Dynamic programming cho các ngôn ngữ lập trình.

```
class DynamicDemo {
    static dynamic DynaMethod(dynamic param) {
        if (param is int) {
            Console.WriteLine("Dynamic parameter of type int has value {0}", param);
            return param;
        }
        else if (param is string) {
            Console.WriteLine("Dynamic parameter of type string has value {0}", param);
            return param;
        }
        else {
            Console.WriteLine("Dynamic parameter of unknown type has value {0}", param);
            return param;
        }
    }

    static void Main(string[] args) {
        dynamic dynaVar1 = DynaMethod(3);
        dynamic dynaVar2 = DynaMethod("Hello World");
        dynamic dynaVar3 = DynaMethod(12.5);
        Console.WriteLine("\nReturned dynamic values:\n{0} \n{1} \n{2}", dynaVar1,
        dynaVar2, dynaVar3);
    }
}
```

```
Dynamic parameter of type int has value 3
Dynamic parameter of type string has value Hello World
Dynamic parameter of unknown type has value 12.5
Returned dynamic values:
3
Hello World
12.5
```