

LẬP TRÌNH CSDL VỚI ENTITY FRAMEWORK

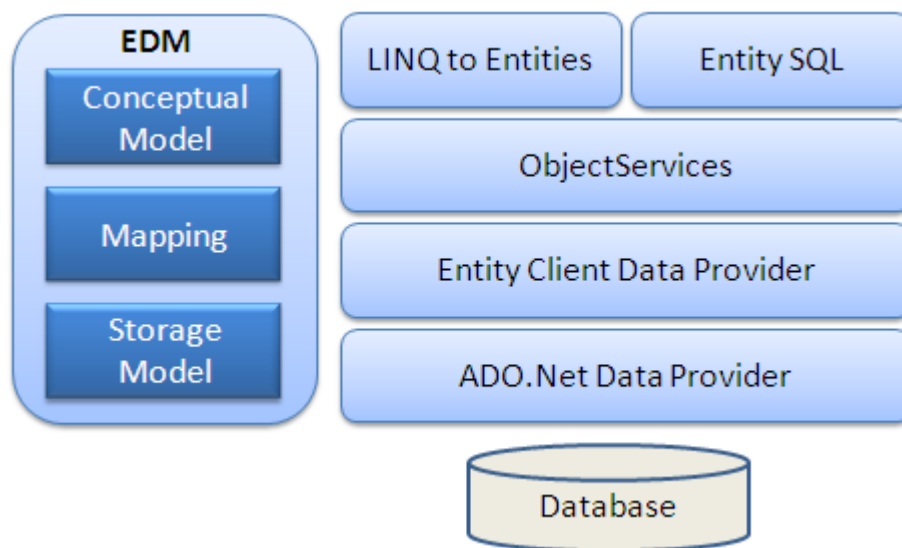
Chú ý: tutorial dựa trên việc đã có kiến thức về cú pháp LinQ.

1. Entity Framework là gì

Microsoft ADO.NET Entity Framework là một khuôn khổ đối tượng Object/ Bảng đồ quan hệ Relational Mapping (**ORM**) cho phép các nhà phát triển dễ dàng làm việc với dữ liệu quan hệ như là các đối tượng, loại bỏ đi sự khó khăn trong việc truy cập dữ liệu trước đây. Bằng cách sử dụng Entity Framework, truy vấn LINQ, thì việc lấy và thao tác dữ liệu như các đối tượng trở nên mạnh mẽ hơn. Entity Framework ORM cung cấp các dịch vụ như change tracking, identity resolution, lazy loading, và truy vấn dữ liệu tập trung vào business logic của ứng dụng.

Đơn giản có thể định nghĩa **Entity Framework** như sau: Entity Framework là một khung Object/Relational Mapping (O/RM). Nó là một kỹ thuật mới cung cấp cho các nhà phát triển ADO.NET một cơ chế tự động để truy cập và lưu trữ dữ liệu trong cơ sở dữ liệu và làm việc với các kết quả ngoài DataReader và DataSet. Các bạn có thể đọc thêm về O/RM [tại đây](#).

Kiến trúc của Entity Framework gồm các thành phần sau:



Trong đó:

EDM (Entity Data Model): bao gồm ba phần chính Conceptual model, Mapping and Storage model.

Conceptual Model - (khái niệm mô hình): là các lớp và mối quan hệ tương ứng với cơ sở dữ liệu. Điều này sẽ được đọc lập từ cơ sở dữ liệu.

Storage Model - (mô hình lưu trữ): mô hình thiết kế cơ sở dữ liệu bao gồm Table, View, Store procedure, Relationship, Key,...

Mapping - (bảng đồ): gồm thông tin các khái niệm mô hình do developer ánh xạ tới mô hình lưu trữ hay cơ sở dữ liệu.

LINQ to Entities: là ngôn ngữ truy vấn được sử dụng để truy vấn với mô hình đối tượng Object model. Giá trị trả về tùy thuộc theo developer, theo Model.

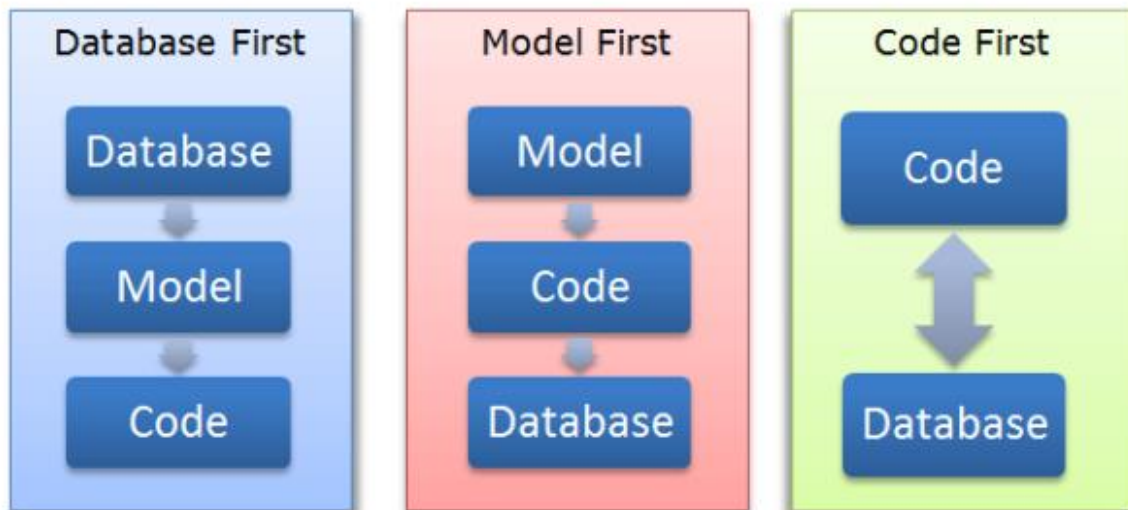
Entity SQL: là ngôn ngữ truy vấn giống như LINQ to Entities. Nhưng nó hơi phức tạp hơn.

Object Service: phục vụ cho việc truy cập, trả giá trị dữ liệu từ cơ sở dữ liệu. Object Service cung cấp đầy đủ dịch vụ để quá trình chuyển đổi dữ liệu từ thực thể đến cấu trúc đối tượng dễ dàng hơn.

Entity Client Data Provider: giúp chuyển đổi L2E hoặc truy vấn Entity SQL vào truy vấn SQL trong cơ sở dữ liệu. Nó sẽ giao tiếp với ADO.NET data provider hoặc lấy dữ liệu từ cơ sở dữ liệu.

ADO.Net Data Provider: lớp này giao tiếp với cơ sở dữ liệu sử dụng theo chuẩn ADO.NET.

Có 3 cách sử dụng Entity Framework: [Code First](#), [Models First](#), [Database First](#).



- **Database First**: Trong trường hợp bạn muốn làm việc với database đã có sẵn. Sử dụng công cụ thiết kế có sẵn trong Visual Studio kết nối database sẽ giúp ta tạo ra các model bao gồm lớp và thuộc tính. Các thông tin về cấu trúc dữ liệu của bạn (Store Schema), mô hình dữ liệu (Conceptual Model), và mối quan hệ giữa chúng được lưu trữ dưới dạng XML trong một tập tin .edmx. **Entity Framework** cung cấp một giao diện hiển thị và chỉnh sửa các tập tin .edmx.

- **Model First:** Trong trường hợp này, chúng ta sử dụng các thiết kế Entity Framework trong Visual Studio tạo ra model trong một tập tin .edmx. Dùng công cụ để generated code từ bản thiết kế model ra database.

- **Code First:** Riêng với cách này có 2 lựa chọn, làm việc với database có sẵn hoặc sẽ tạo mới. Nhưng dù làm với cách nào, chúng ta cũng dùng codebehind để xử lý là chính. Không dùng các tool, giao diện trực quan giống như 2 cách bên trên.

Việc lựa chọn workflow phù hợp cho chương trình của bạn phụ thuộc vào câu trả lời cho hai câu hỏi sau:

Câu hỏi 1: Bạn tạo sẵn cơ sở dữ liệu với các bảng hay chưa?

Câu hỏi 2: Bạn muốn tạo các Model theo cách viết code hay dựa vào giao diện thiết kế? Trong quy trình xây dựng ứng dụng, thông thường cơ sở dữ liệu sẽ được xây dựng trước việc này có thể do nhân viên phân tích thiết kế chuyên về cơ sở dữ liệu làm hay với các ứng dụng đơn giản hơn thì lập trình viên thực hiện. Sau đó tùy theo yêu cầu phát sinh trong quá trình thực hiện hay từ khách hàng mà ta điều chỉnh cơ sở dữ liệu này. Vậy câu trả lời cho câu hỏi 1 thông thường là có.

Dù cách nào thì cuối cùng cũng phải có Code để thao tác trong mã lệnh và Database để lưu trữ dữ liệu. Model chỉ là một thành phần trung gian.

Theo kinh nghiệm của nhiều lập trình viên đi trước, Code First là phương án mềm dẻo nhất (hơi mất công ngồi gõ code, tuy nhiên nếu dùng tốt code snippets thì đỡ nhiều).

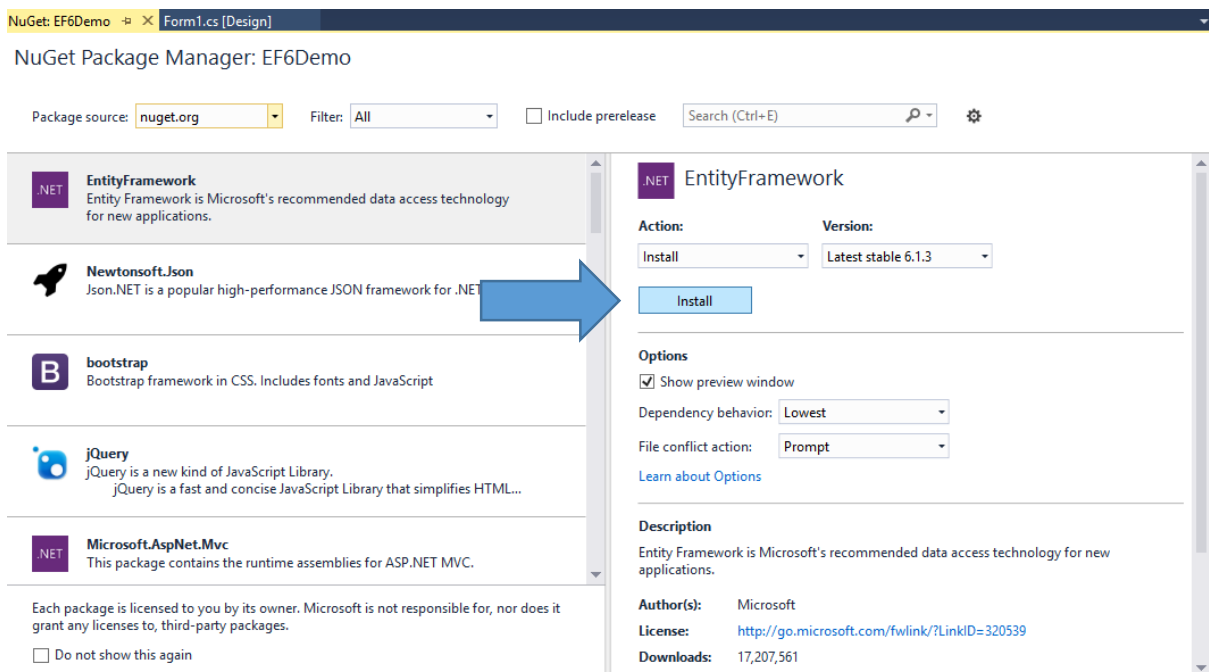
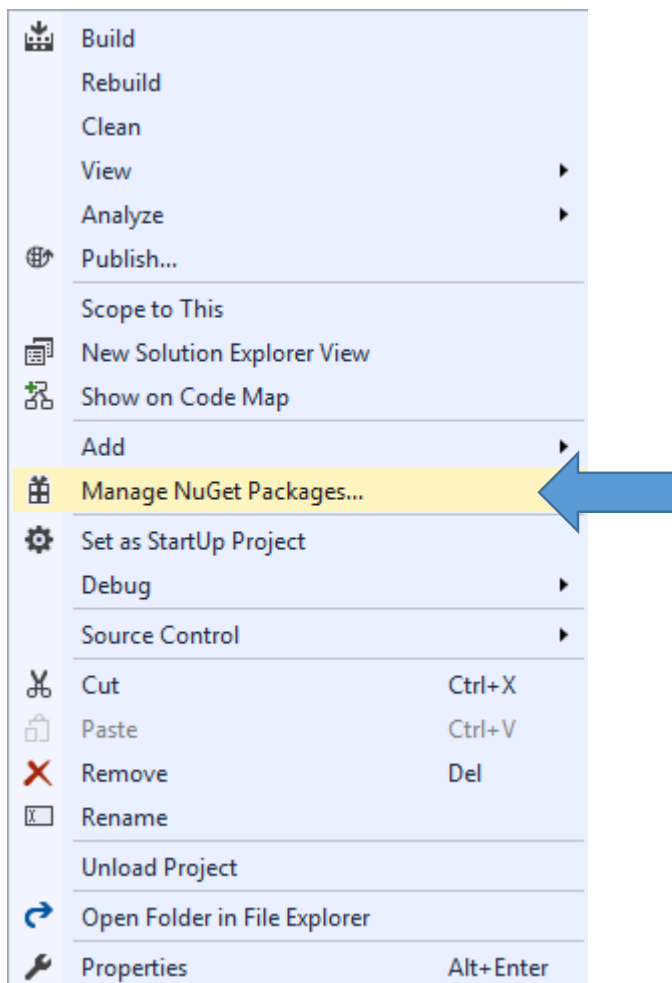
2. Làm việc với Entity Framework Database First

Trong ví dụ này, chúng ta sẽ sử dụng phương thức Database First với cơ sở dữ liệu Northwind. Trước tiên cần phải tạo Entity Data Model. Đây là lớp chứa các mô hình dữ liệu ở mức khái niệm và là cơ sở hạ tầng để đọc và lưu dữ liệu từ cơ sở dữ liệu.

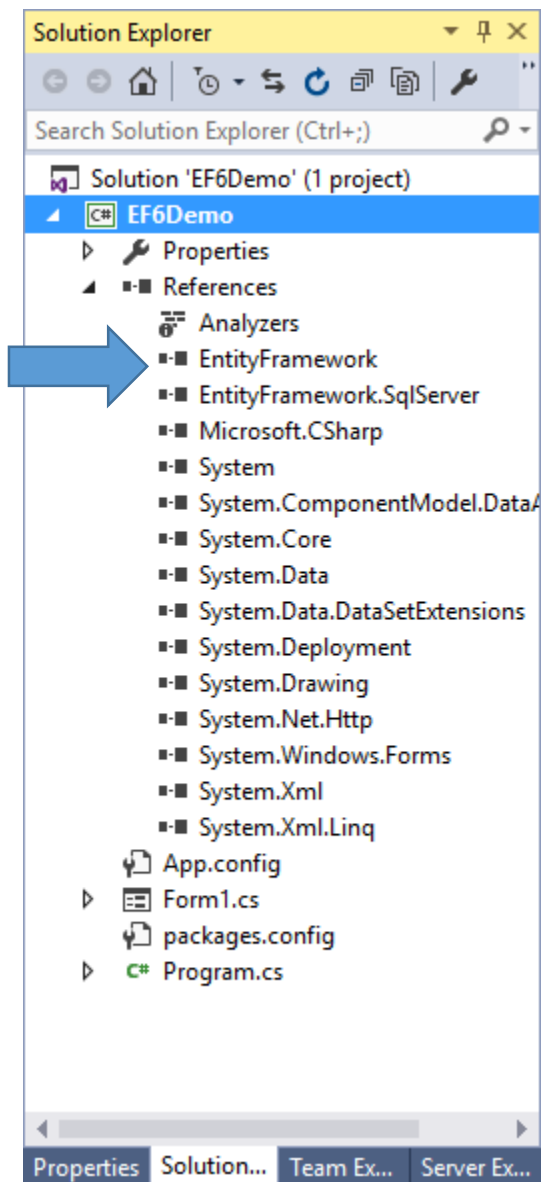
Các bước tạo Entity Data Model như sau:

Trước tiên, cần phải bổ sung thư viện Entity Framework vào project.

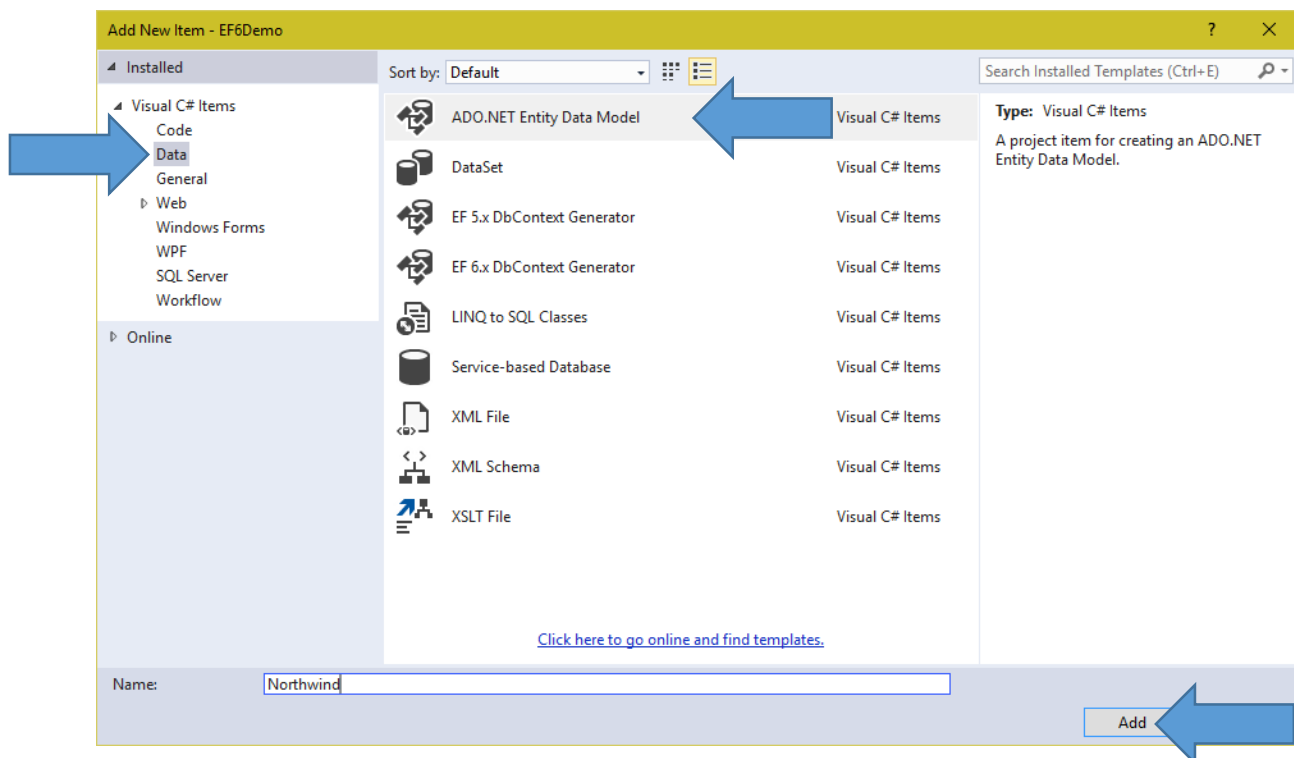
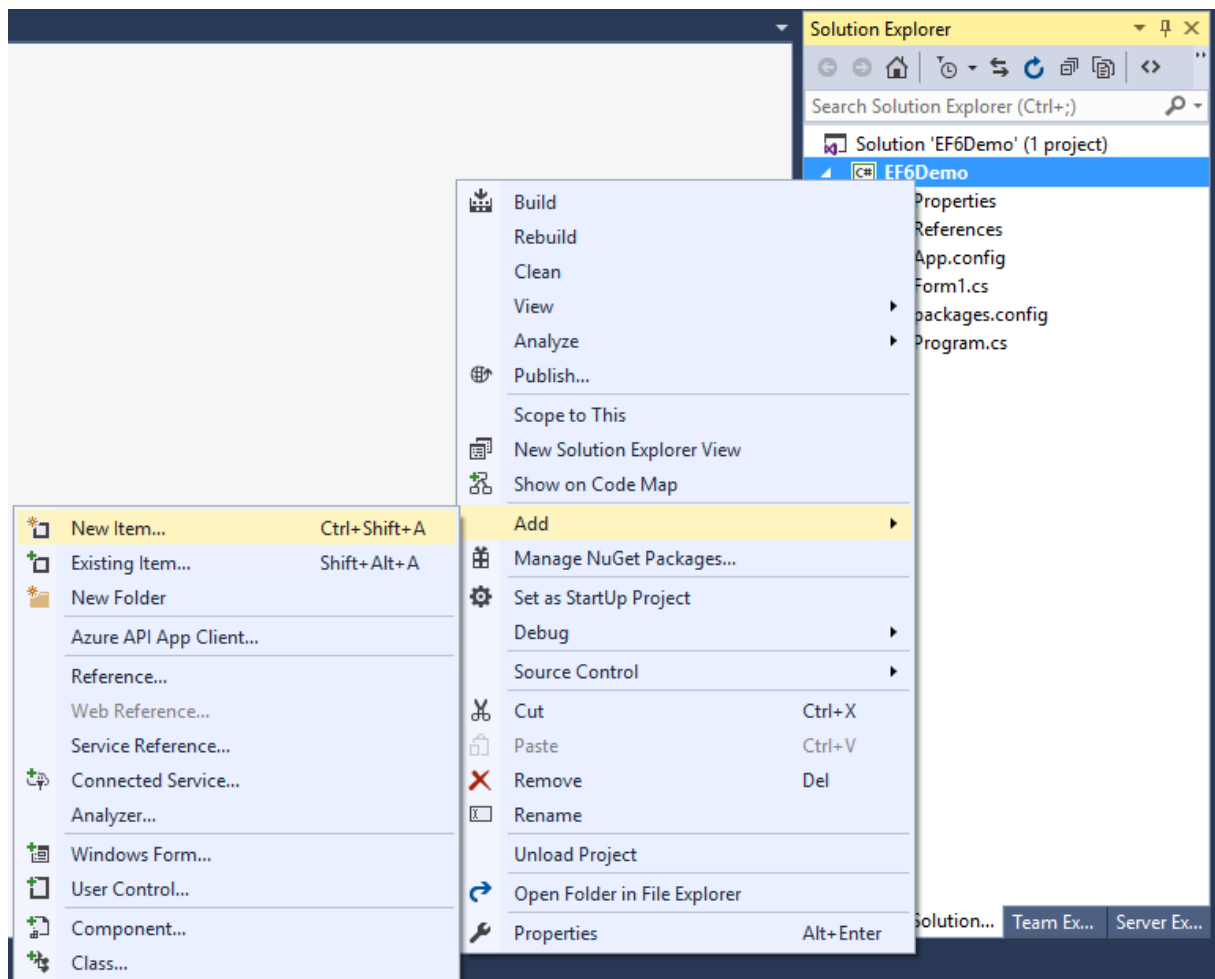
Right-click vào project và chọn Manage Nuget Packages, search Entity Framework và cài đặt (install):



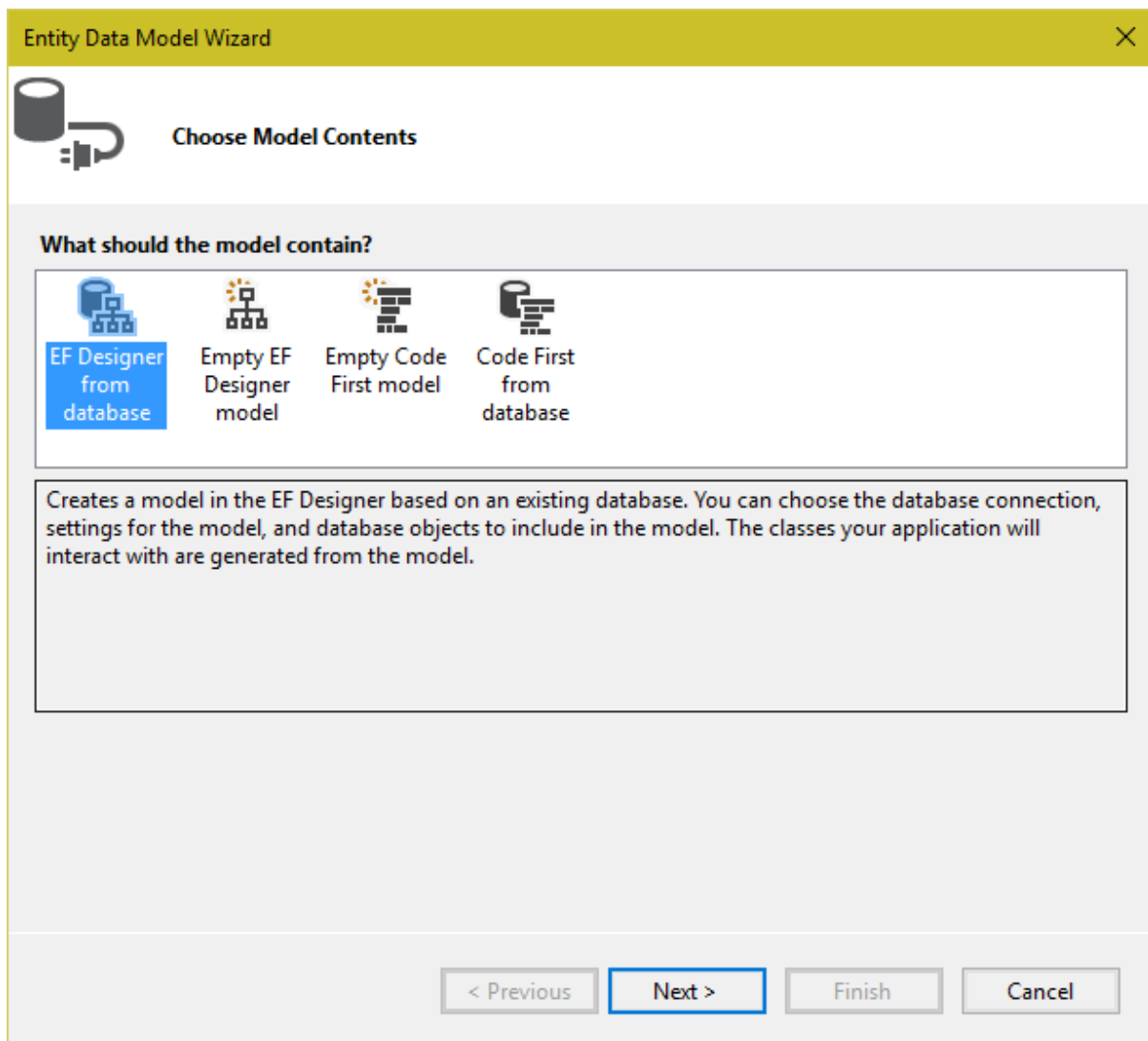
Kiểm tra lại phần References của project:



Tạo Entity Data Model, right-click vào Project Explorer và Add| New Item. Cửa sổ Add New Item hiện lên, chọn Data| ADO.NET Entity Data Model, đặt tên và chọn Add.



Cửa sổ Entity Data Model Wizard hiện lên, chọn EF Designer from database:.



Chọn New Connection, cung cấp thông tin kết nối đến SQL Server như Server Name, Username/ Password để đăng nhập vào SQL Server và chọn cơ sở dữ liệu cần làm việc:

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient) Change...

Server name:
. Refresh

Log on to the server

☐ Use Windows Authentication

☒ Use SQL Server Authentication

User name: sa

Password:

☒ Save my password

Connect to a database

☒ Select or enter a database name:

▼

☐ KQXS_DB

☐ MarathonSkills2015

☐ master

☐ model

☐ msdb

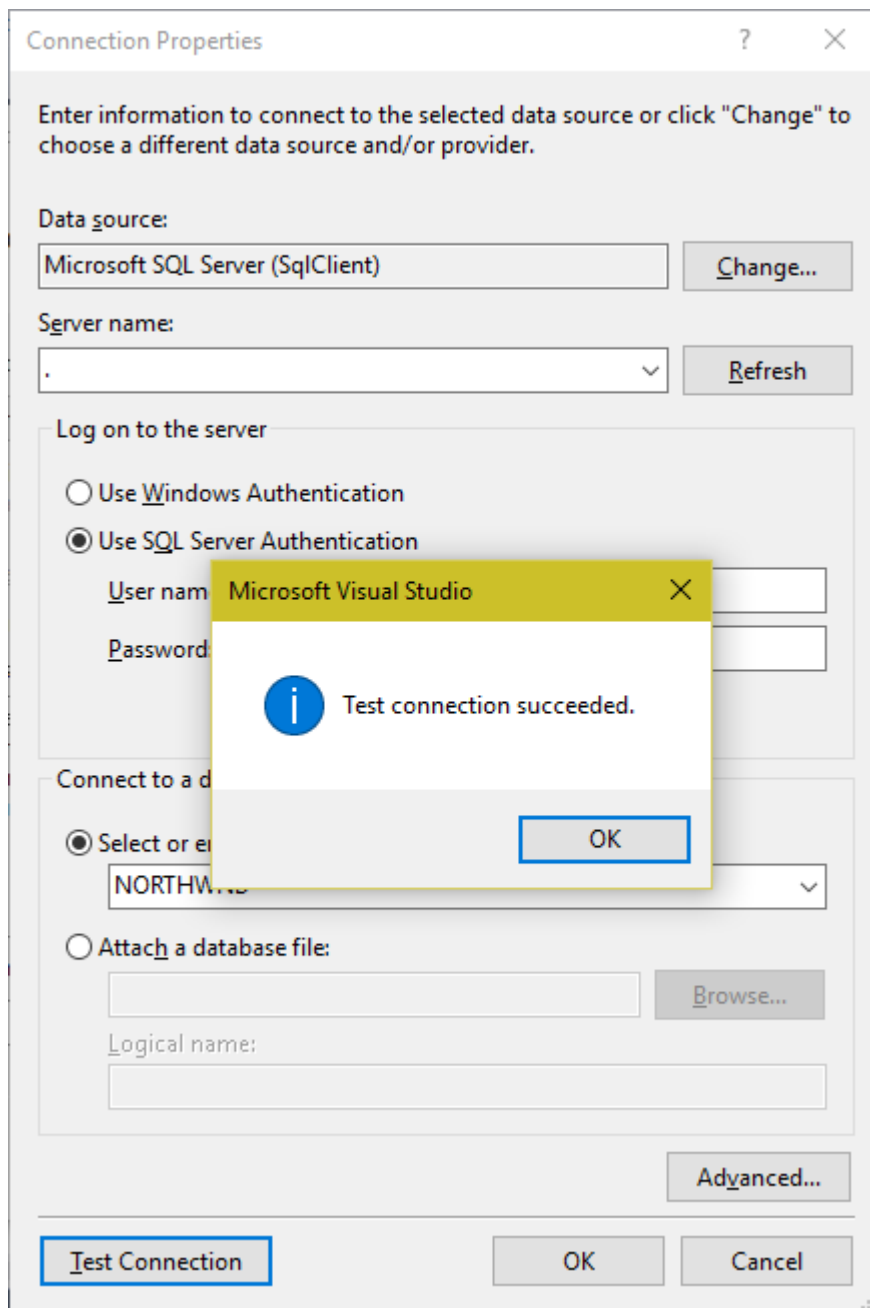
☐ NORTHWND

☐ tempdb

Advanced...


Test Connection OK Cancel

Kiểm tra kết nối đã thành công hay chưa:



Lưu lại thông tin kết nối trong file App.config, nhấn Next:

Entity Data Model Wizard

 Choose Your Data Connection

Which data connection should your application use to connect to the database?

desktop-fifehu.NORTHWND.dbo

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☒ Yes, include the sensitive data in the connection string.

Connection string:

metadata=res://*/Northwind.csdl|res://*/Northwind.ssdl|res://*/Northwind.msl;provider=System.Data.SqlClient;provider connection string="data source=.;initial catalog=NORTHWND;persist security info=True;user id=sa;password=*****;MultipleActiveResultSets=True;App=EntityFramework"

☒ Save connection settings in App.Config as:

NORTHWNDEntities

< Previous

Next >

Finish

Cancel

Chọn các bảng cần làm việc, sau đó nhấn Finish:

Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☒ Tables
 - ☒ dbo
 - ☒ Categories
 - ☒ CustomerCustomerDemo
 - ☒ CustomerDemographics
 - ☒ Customers
 - ☒ Employees
 - ☒ EmployeeTerritories
 - ☒ Order Details
 - ☒ Orders
 - ☒ Products
 - ☒ Region

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

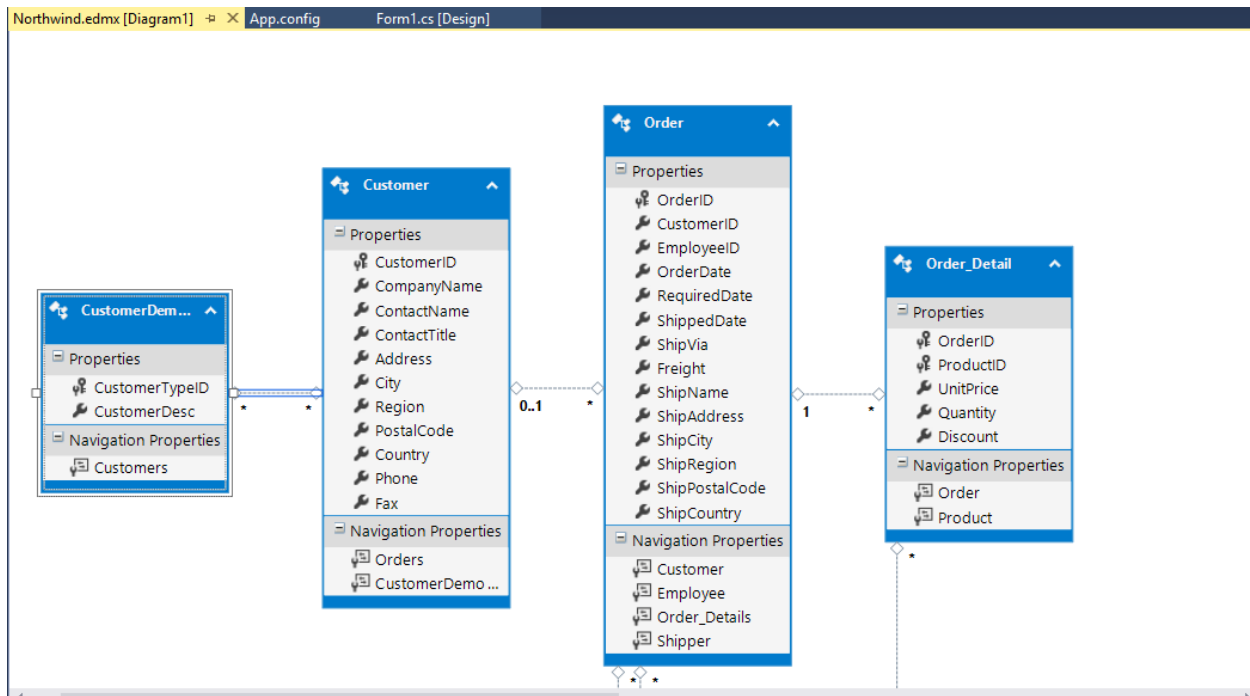
☒ Import selected stored procedures and functions into the entity model

Model Namespace:

NORTHWNDModel

< Previous Next > **Finish** Cancel

Đối tượng Entity Data Model sẽ được tạo ra. Lúc này nhìn vào cửa sổ Project Explorer các bạn sẽ thấy một file .edmx được thêm vào. Mở file này chúng ta sẽ thấy cơ sở dữ liệu đã được ánh xạ thành các class và mối quan hệ giữa các entity. Với file .edmx cho thấy thiết kế mô hình dữ liệu, có thể chỉnh sửa các mô hình và tạo lại các mô hình bất cứ lúc nào.



Bây giờ chúng ta có thể sử dụng data model trong ứng dụng của mình.

Thiết kế giao diện của Form1 như sau:

The Form1 window displays the following controls:

- Category ID: Add
- Category Name: Update
- Category Description: Delete
- A large empty rectangular area at the bottom, intended for a DataGridView.

Form này chúng ta sẽ lấy thông tin từ một bảng của cơ sở dữ liệu, hiển thị nó lên DataGridView cũng như thêm mới, cập nhật, xóa các bản ghi trong bảng đó.

Trước tiên là viết hàm để lấy dữ liệu từ cơ sở dữ liệu thông qua đối tượng DataContext. Đây là đối tượng cho phép chúng ta thao tác với cơ sở dữ liệu thông qua các phương thức Add(), Update(), Delete() v.v...

Tạo đối tượng DataContext để thao tác dữ liệu:

```
public partial class Form1 : Form
{
    readonly NORTHWNDEntities db = new NORTHWNDEntities();
    1 reference
    public Form1()
    {
        InitializeComponent();
    }
}
```

Để chỉ định nguồn dữ liệu cho DataGridView chúng ta đặt thuộc tính DataSource cho nó. View code của Form1 và thêm hàm sau:

```
private void LoadGrid()
{
    var cat = from category in db.Categories
              select new
              {
                  category.CategoryID,
                  category.CategoryName,
                  category.Description,
                  category.Picture
              };

    gridCategory.DataSource = cat.ToList();
}
```

Ở sự kiện Form_Load() gọi hàm này ra để lấy dữ liệu lên:

```
private void Form1_Load(object sender, EventArgs e)
{
    LoadGrid();
}
```

Thêm mới một sản phẩm, chúng ta viết sự kiện cho nút Add:

Trước tiên cần kiểm tra xem các ô textbox đã có dữ liệu hay chưa, nếu chưa có thì cần phải nhập đủ dữ liệu vào các ô:

```
private new string Validate()
{
    if (string.IsNullOrEmpty(txtCatId.Text.Trim()))
        return "Id không được trống";
    if (string.IsNullOrEmpty(txtCatName.Text.Trim()))
        return "Tên không được trống";
    if (string.IsNullOrEmpty(txtCatDescription.Text.Trim()))
        return "";
    return "";
}
```

Trước khi thêm, gọi hàm Validate() để kiểm tra. Nếu không có lỗi thì tạo mới một đối tượng Category và thêm vào cơ sở dữ liệu bằng phương thức Add(), lưu lại thay đổi bằng phương thức SaveChanges():

```
private void btnAddCategory_Click(object sender, EventArgs e)
{
    string result = Validate();
    if (!string.IsNullOrEmpty(result))
    {
        MessageBox.Show(result, "Infomation", MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }
    try
    {
        {
            var category = new Category()
            {
                CategoryID = Convert.ToInt32(txtCatId.Text),
                CategoryName = txtCatName.Text,
                Description = txtCatDescription.Text,
                Picture = null
            };
            if (db.Categories.Find(category.CategoryID) == null)
            {
                db.Categories.Add(category);
                db.SaveChanges();
            }
            else MessageBox.Show("Mã nhóm đã tồn tại");
        }
        catch (Exception ex) { };
        LoadGrid();
    }
}
```

Với thuộc tính Primary Key, cần phải kiểm tra xem nó đã tồn tại trong cơ sở dữ liệu hay chưa, nếu chưa thì mới thêm vào. Tuy nhiên với những bảng khi thiết kế trường Primary Key đã quy định cho nó thuộc tính Auto Increment thì các bạn không cần quan tâm đến vấn đề kiểm tra mã nữa, vì nó sẽ được tự động tăng.

Viết code cho sự kiện CellClick của DataGridView, khi click vào một ô của DataGridView thì sẽ load dữ liệu lên Textbox ở trên:

```
private void gridProduct_CellClick(object sender, DataGridViewCellEventArgs e)
{
    int id = e.RowIndex;
    if (id < 0)
        return;
    txtCatId.Text = Convert.ToString(gridCategory[0, id].Value);
    txtCatName.Text = Convert.ToString(gridCategory[1, id].Value);
    txtCatDescription.Text = Convert.ToString(gridCategory[2, id].Value);
}
```

Thuộc tính e.RowIndex sẽ cho biết chỉ số của dòng trong DataGridView đã được click. Từ chỉ số này chúng ta lấy được các thông tin của dòng đó như CategoryID, CategoryName, v.v... và đặt thuộc tính Text cho các ô Textbox.

Cập nhật dữ liệu, viết hàm cho sự kiện Click của nút Update:

```

private void btnUpdateCategory_Click(object sender, EventArgs e)
{
    string result = Validate();
    if (!string.IsNullOrEmpty(result))
    {
        MessageBox.Show(result, "Infomation", MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }
    try
    {
        int id = Convert.ToInt32(txtCatId.Text);
        if (id < 0)
            return;
        var category = db.Categories.Find(id);
        if (category != null)
        {
            category.CategoryID = Convert.ToInt32(txtCatId.Text);
            category.CategoryName = txtCatName.Text;
            category.Description = txtCatDescription.Text;
            db.SaveChanges();
        }
        else MessageBox.Show("Không cập nhật được!");
    }
    catch (Exception ex) { };
    LoadGrid();
}

```

Dùng phương thức Find() của đối tượng DataContext để tìm bản ghi có khóa chính ở ô Textbox để cập nhật thông tin.

Tương tự như vậy viết code cho nút Delete khi Click() vào nút này, cũng dùng phương thức Find() để tìm đối tượng cần xóa và sử dụng phương thức Remove() để xóa bản ghi trong cơ sở dữ liệu:

```

private void btnDeleteCategory_Click(object sender, EventArgs e)
{
    string result = Validate();
    if (!string.IsNullOrEmpty(result))
    {
        MessageBox.Show(result, "Infomation", MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }
    try
    {
        int id = Convert.ToInt32(txtCatId.Text);
        if (id < 0)
            return;
        var category = db.Categories.Find(id);
        if (category != null)
        {
            db.Categories.Remove(category);
            db.SaveChanges();
        }
    }
    catch (Exception ex) { };
    LoadGrid();
}

```

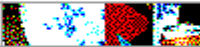

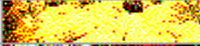

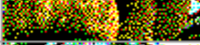
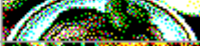

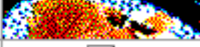

Form 1 khi thực thi:

Form1

Category ID:

Category Name:

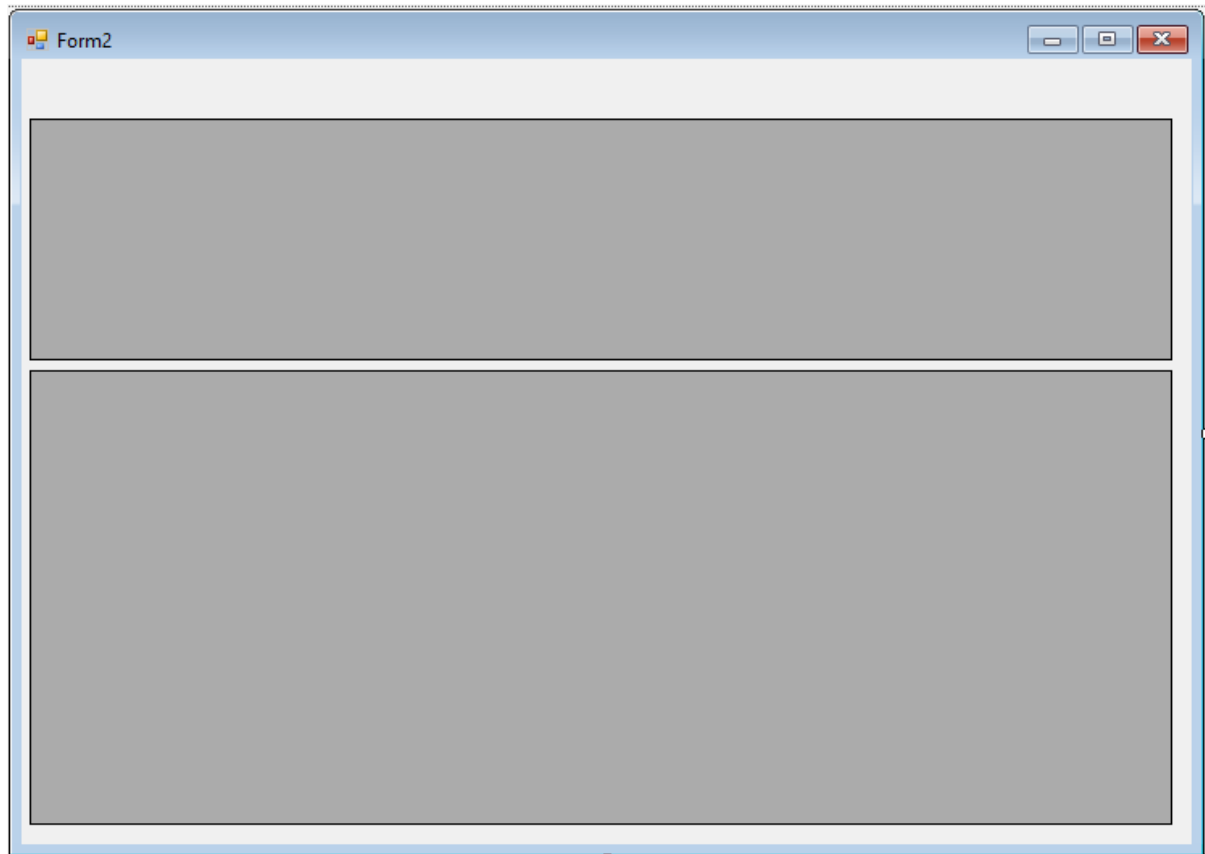
Category Description:

	CategoryID	CategoryName	Description	Picture
▶	1	Beverages	Soft drinks, coffe...	
	2	Condiments	Sweet and savor...	
	3	Confections	Desserts, candie...	
	4	Dairy Products	Cheeses	
	5	Grains/Cereals	Breads, crackers,...	
	6	Meat/Poultry ww	Prepared meats	
	7	Produce	Dried fruit	
	8	Seafood	Seaweed and fish	
	12	Confections	Desserts, candie...	

Trên đây là các phương thức đơn giản để thao tác với một bảng trong cơ sở dữ liệu. Tiếp theo chúng ta sẽ làm việc với Form dạng Master-Detail.

Thêm một form mới Form2 vào project. Thiết kế giao diện cho form này gồm có 2 DataGridView. DataGridView ở trên sẽ hiển thị thông tin về các danh mục sản phẩm, DataGridView ở dưới sẽ hiển thị các loại sản phẩm trong danh mục sản phẩm đó.

Giao diện của Form2 như sau:



Tương tự như ở Form1, tạo đối tượng DataContext để thao tác dữ liệu:

```
public partial class Form2 : Form
{
    NORTHWNDEntities db = new NORTHWNDEntities();
    Oreferences
    public Form2()
    {
        InitializeComponent();
    }
}
```

Viết hàm để đọc dữ liệu lên DataGridView Master:

```
private void LoadMaster()
{
    var cat = from category in db.Categories
              select new
              {
                  category.CategoryID,
                  category.CategoryName,
                  category.Description,
                  category.Picture
              };
    gridMaster.DataSource = cat.ToList();
}

private void Form2_Load(object sender, EventArgs e)
{
    LoadMaster();
}
```

Ở DataGridView Detail, cần lọc dữ liệu theo CategoryID được click ở DataGridView Master. Hàm đọc dữ liệu cho DataGridView Detail như sau:

```
private void LoadDetail(int id)
{
    var sp = from category in db.Categories
              join product in db.Products on category.CategoryID equals product.CategoryID
              join supplier in db.Suppliers on product.SupplierID equals supplier.SupplierID
              where category.CategoryID == id
              select new
              {
                  product.ProductID,
                  product.ProductName,
                  category.CategoryName,
                  supplier.CompanyName,
                  product.QuantityPerUnit,
                  product.UnitPrice,
                  product.UnitsInStock,
                  product.UnitsOnOrder,
                  product.ReorderLevel
              };
    gridDetail.DataSource = sp.ToList();
}
```

Khi click vào một ô ở DataGridView Master:

```
private void gridMaster_CellClick(object sender, DataGridViewCellEventArgs e)
{
    int id = e.RowIndex;
    if (id < 0)
        return;
    LoadDetail(id+1);
}
```

Form 2 khi thực thi:

Form2

CategoryID	CategoryName	Description	Picture
5	Grains/Cereals	Breads, crackers,...	
6	Meat/Poultry ww	Prepared meats	
7	Produce	Dried fruit	
8	Seafood	Seaweed and fish	
12	Confections	Desserts, candie...	

ProductID	ProductName	CategoryName	CompanyName	QuantityPerUnit	UnitPrice	UnitsInStock
7	Uncle Bob's Orga...	Produce	Grandma Kelly's ...	12 - 1 lb pkgs.	30.0000	15
14	Tofu	Produce	Mayumi's	40 - 100 g pkgs.	23.2500	35
28	Rössle Sauerkraut	Produce	Plutzer Lebensmit...	25 - 825 g cans	45.6000	26
51	Manjimup Dried A...	Produce	G'day, Mate	50 - 300 g pkgs.	53.0000	20
74	Longlife Tofu	Produce	Tokyo Traders	5 kg pkg.	10.0000	4

Một ví dụ nữa về thao tác lọc dữ liệu và hiển thị biểu đồ trong Form, thêm vào Form3 và thiết kế giao diện như sau:

Form3

Minimum Unit Price Category

Category	Price
1	48
2	30
3	60
4	15
5	55
6	68
7	75
8	0

Với ô Textbox để nhập vào giá thành của sản phẩm, một Combobox để lọc sản phẩm theo nhóm. Biểu đồ sẽ hiển thị sản phẩm với mức giá đã được lọc. DataGridView hiển thị đầy đủ các thông tin của sản phẩm.

Tương tự như 2 form trước, chúng ta cũng tạo đối tượng DataContext:

```
public partial class Form3 : Form
{
    NORTHWNDEntities db = new NORTHWNDEntities();
    0 references
    public Form3()
    {
        InitializeComponent();
    }
}
```

Đọc dữ liệu lên DataGridView tương tự như Form1 và Form2. Ở đây viết 2 hàm với 1 hàm có 1 tham số là giá sản phẩm, 1 hàm có 2 tham số là giá sản phẩm và loại sản phẩm. Đồng thời gắn dữ liệu lên điều khiển Chart.

```

private void LoadDataToGrid(decimal price)
{
    try
    {
        var sp = from category in db.Categories
                  join product in db.Products on category.CategoryID equals product.CategoryID
                  join supplier in db.Suppliers on product.SupplierID equals supplier.SupplierID
                  where product.UnitPrice >= price
                  orderby product.UnitPrice
                  select new
                  {
                      product.ProductID,
                      product.ProductName,
                      category.CategoryName,
                      supplier.CompanyName,
                      product.QuantityPerUnit,
                      product.UnitPrice,
                      product.UnitsInStock,
                      product.UnitsOnOrder,
                      product.ReorderLevel
                  };
        chartPrice.DataSource = sp.ToList();
        chartPrice.DataBind();
        gridProduct.DataSource = sp.ToList();
    }
    catch (Exception ex) { };
}

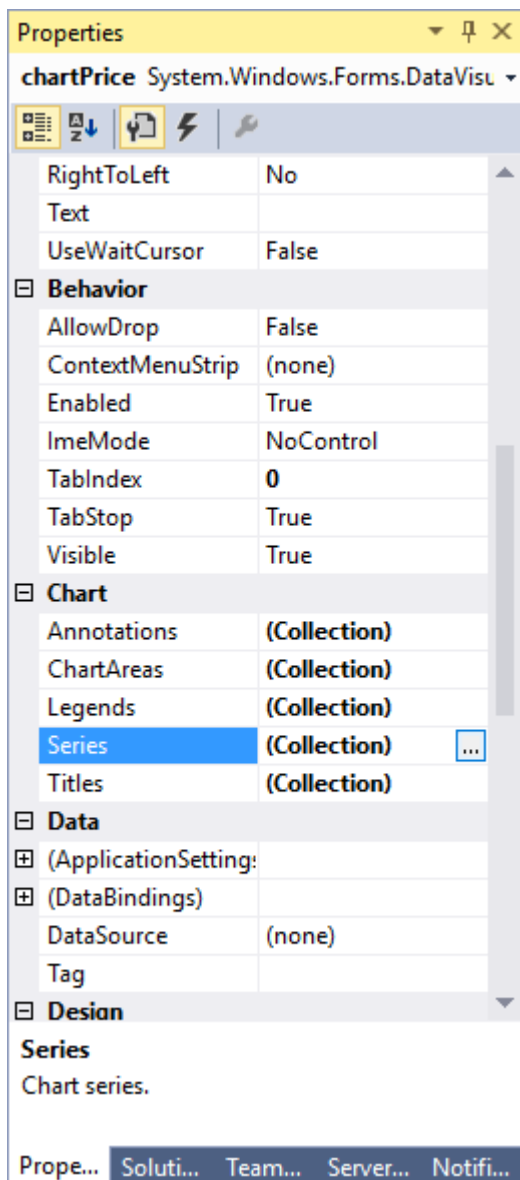
private void LoadDataToGrid(decimal price, int categoryid)
{
    try
    {
        var sp = from category in db.Categories
                  join product in db.Products on category.CategoryID equals product.CategoryID
                  join supplier in db.Suppliers on product.SupplierID equals supplier.SupplierID
                  where product.UnitPrice >= price && category.CategoryID == categoryid
                  orderby product.UnitPrice
                  select new
                  {
                      product.ProductID,
                      product.ProductName,
                      category.CategoryName,
                      supplier.CompanyName,
                      product.QuantityPerUnit,
                      product.UnitPrice,
                      product.UnitsInStock,
                      product.UnitsOnOrder,
                      product.ReorderLevel
                  };
        chartPrice.DataSource = sp.ToList();
        chartPrice.DataBind();
        gridProduct.DataSource = sp.ToList();
    }
    catch (Exception ex) { };
}

```

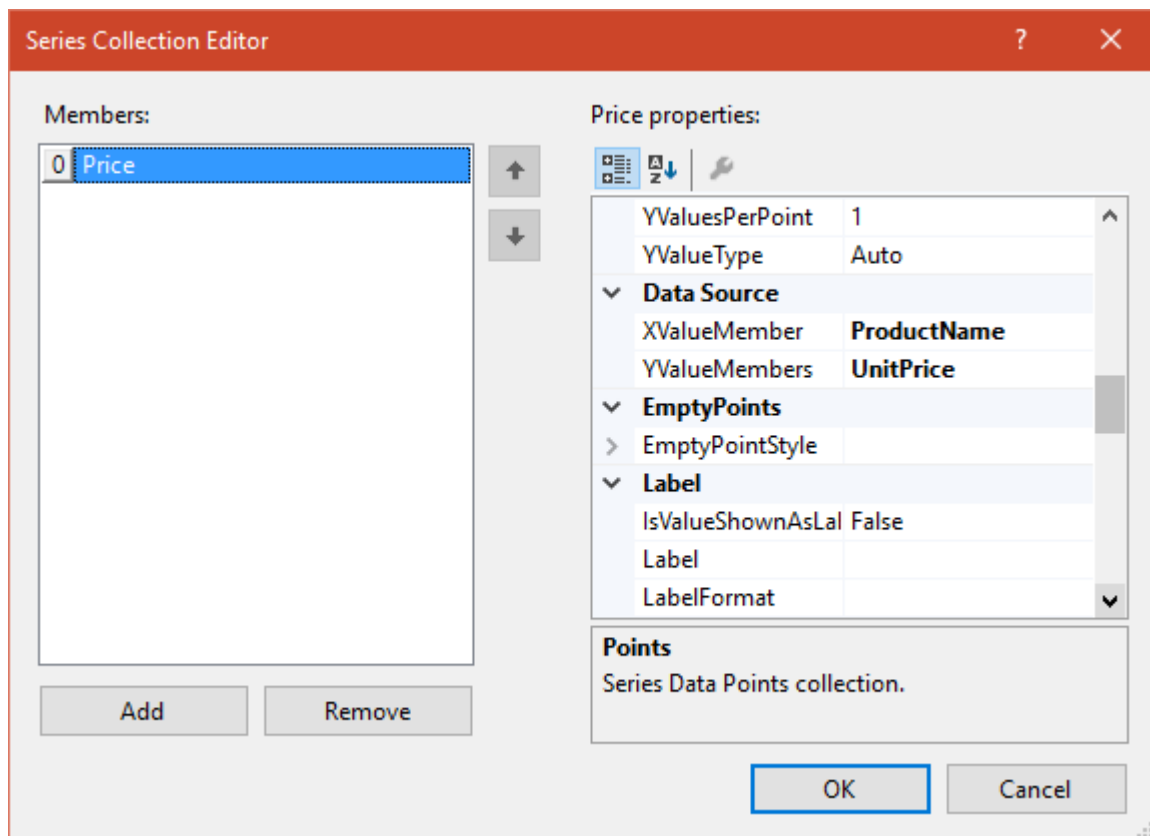
Hàm load dữ liệu lên Combobox:

```
private void LoadDataToCombo()
{
    try
    {
        var combo1 = from cat in db.Categories
                      select cat;
        cbCategory.DataSource = combo1.ToList();
        cbCategory.DisplayMember = "CategoryName";
        cbCategory.ValueMember = "CategoryID";
    }
    catch (Exception ex) { };
}
```

Đặt các thuộc tính cho Chart:



Chọn thuộc tính Series và đặt XvalueMember là ProductName, YvalueMember là UnitPrice (tên các trường trong cơ sở dữ liệu).



Viết code cho sự kiện KeyPress của ô Textbox và sự kiện SelectedIndexChanged của Combobox:

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == 13)
    {
        try
        {
            decimal filter = Convert.ToDecimal(txtPrice.Text);
            LoadDataToGrid(filter);
            e.Handled = true;
        }
        catch (Exception ex) { };
    }
}

private void cbCategory_SelectedIndexChanged(object sender, EventArgs e)
{
    try
    {
        decimal filter = Convert.ToDecimal(txtPrice.Text);
        int catid = Convert.ToInt32(cbCategory.SelectedValue.ToString());
        LoadDataToGrid(filter, catid);
    }
    catch (Exception ex) { };
}
```

Ở sự kiện Form_Load():

```
-----  
private void Form3_Load(object sender, EventArgs e)  
{  
    try  
    {  
        txtPrice.Text = "0";  
        decimal filter = Convert.ToDecimal(txtPrice.Text);  
        LoadDataToGrid(filter);  
        LoadDataToCombo();  
    }  
    catch (Exception ex) { };  
}
```

Form3 khi thực thi:

