1. Tổng quát

Trước khi tìm hiểu các Các phương thức mở rộng trong LINQ, các bạn nên hiểu rõ 2 khái niệm sau đây trong cú pháp truy vấn LINQ:

- **Element** : Đại diện cho phần tử trong một tập hợp (Collection, Table, ...)
- **Sequence** : Đại diện cho một hợp (Collection, Table, ...)

Trong tài liệu này tôi sẽ thường xuyên dùng element và sequence để cho cách diễn giải ngắn gọn hơn và cũng giúp cho các bạn dễ dàng nhớ đến 2 khái niệm này

Trong cú pháp cơ bản sau, chúng ta dể dàng nhận ra: **Id** là element và **source** là sequence

```
from id in source
  [where condition
  orderby ordering, ordering, ... ]
  select expr
```

2. Các phương thức mở rộng (Extension Methods):

Trong mục này, tôi sẽ lấy CSDL QLPHIM.mdf và BANHANG.mdf làm nguồn dữ liệu cho các ví dụ. Các nguồn là mảng hoặc Collection, nếu đã được khai báo trong các ví dụ trước thì tôi sẽ không khai báo lại nữa, xem như đã có rồi.

Phần khai báo chung cho các ví dụ về LINQ to Entities là như sau:

```
QLPhimDbContext db = new QLPhimDbContext();
DbSet<Phim> dsPhim;
BANHANGEntities context = new BANHANGEntities();
ObjectQuery<DONDH> dsDH;
ObjectQuery<CTDONDH> dsCTDH;
ObjectQuery<SANPHAM> dsSP;
ObjectQuery<LOAISP> dsLoai;
```

Trong sự kiện **Load** của Form, khởi tạo các đối tượng trên:

```
dsPhim = db.Phims;
dsDH = context.DONDHs;
dsCTDH = context.CTDONDHs;
dsSP = context.SANPHAMs;
dsLoai = context.LOAISPs;
```

2.1. Filtering (lọc)

- **Where:** Lọc dữ liệu trong tập hợp dựa trên điều kiện cụ thể

```
int[] dsSo = { 5, 4, 1, 3, 9, 4, 6, 5, 2, 1 };
List<string> dsChuoi = new List<string>{"Thiện", "căn", "ở", "tại", "lòng", "ta",
"chữ", "Tâm", "kia", "mới", "bằng", "ba", "chữ","Tài" };
//lọc các số chẵn trong dsSo
IEnumerable<int> query = dsSo.Where(x => x % 2 == 0);
//Lọc các chuỗi có chiều dài >=4
IEnumerable<string> query=dsChuoi.Where(x => x.Length>= 4);
//lọc các phim có tên bắt đầu là chữ T
var query = dsPhim.Where(x=> x.TenPhim.StartsWith("T"));
```

- **Distinct:** Trả về các element riêng biệt (không trùng) từ một sequence

```
//Lọc các số duy nhất trong dsSo
IEnumerable<int> query = dsSo.Distinct();
//Đếm có bao nhiêu chuỗi không trùng nhau
int so = dsChuoi.Distinct().Count();
//Đếm có bao nhiêu thể loại trong bảng Phim, thay vì đếm trong bảng TheLoai
int so = dsPhim.Select(x => x.TheLoaiID).Distinct().Count();
```

- **Take:** Trả về con số element (số dòng) cụ thể liên tục liền kề tính từ vị trí bắt bắt đầu của sequence

```
//lấy 6 chuỗi đầu tiên trong dschuoi
IEnumerable<string> query = dsChuoi.Take(6);
//lấy 3 phim đầu tiên trong bảng Phim
IEnumerable<Phim> query = dsPhim.Take(3);
```

- **TakeWhile**: Trả về con số element của tập hợp thỏa mãn điều kiện cụ thể
  Lưu ý: p.thức này thường dùng với hành động sắp xếp trước đó

```
//lấy những chuỗi đầu tiên có chứa ký tự n
IEnumerable<string> query= dsChuoi.TakeWhile(x => x.Contains('n'));
//sắp xếp theo Đơn gía, lấy phim có đơn giá <=20000
var query = dsPhim.OrderBy(x=> x.DonGia)
                  .Select(x=> new { x.MaPhim, x.TenPhim, x.DonGia} )
                  .ToList().TakeWhile(x => x.DonGia<=10000);
```

- **Skip**: Bỏ qua số element xác định và trả về số element còn lại trong sequence
  Lưu ý: trong LINQ to Entities, Skip thường kết hợp với Take để phân trang dữ liệu (Data Paging)

```
//Bỏ qua 6 element chuỗi đầu tiên
```

```
var query = dsChuoi.Skip(6);
//Bỏ qua 6 element chuỗi đầu tiên và chỉ lấy 3 element tiếp theo
var query = dsChuoi.Skip(6).Take(3);
//Phân trang dữ liệu: Lấy 3 phim tính từ phim thứ 4 trở đi
var query = dsPhim.Select(x => new { x.MaPhim, x.TenPhim, x.DonGia })
                                                    .Skip(3).Take(3);
```

- **SkipWhile**: Bỏ qua số element thỏa mãn điều kiện xác định và trả về số element còn lại trong sequence

  Lưu ý: p.thức này thường dùng với hành động sắp xếp trước đó

```
//Sắp dsSo giảm dần, sau đó chỉ lấy các element < 5
var query = dsSo.OrderByDescending(x => x).SkipWhile(x => x >= 5);
//Hiển thị 5 đơn hàng gần nhất sau ngày đặt hàng 21/04/2011
var query = dsDH.OrderBy(x => x.NgayDat).ToList()
                .SkipWhile(x => x.NgayDat <= DateTime.Parse("4/21/2011"))
                .Take(5).ToList();
```

2.2. Projecting (phép chiếu)

- **Select**: Chọn các thuộc tính của element để tạo ra một kiểu mới

```
//Liệt kê các element trong dsSo và tạo ra kiểu string
IEnumerable<string> query = dsSo.Select(x => "Số : " + x.ToString());
//kết hợp với chỉ số của element tạo ra kiểu vô danh
var query = dsChuoi.Select((x, i) => new { chiso = i, giatri = x });
//Liệt kê các sản phẩm có giá khuyến mãi
var query = dsSP.Where(x => x.GiaKM > 0)
       .Select(x => new {x.SanphamID, x.TenSanpham, x.LOAISP.TenLoai, x.GiaKM });
```

- **SelectMany**: Chọn các thuộc tính của element để tạo ra kiểu IEnumerable<T> và chuyển các kết quả sequence thành **một sequence**

```
ConNguoi[] dsNhanvien =
 { new ConNguoi{ Hoten="Nhung", Dienthoai = new List<string>{ "0909047213",
                                                              "38221568" } },
   new ConNguoi{ Hoten="Đoan",  Dienthoai = new List<string>{ "0909047213",
                                                              "38221568" } },
   new ConNguoi{ Hoten="Hoa", Dienthoai = new List<string>{ "0909047213",
                                                              "38221568" } }
 };
IEnumerable<string> query = dsNhanvien.SelectMany(x => x.Dienthoai);
```

```
//hoặc
var query=dsNhanvien.SelectMany(x => x.Dienthoai.Select(y=> new{x.Hoten, y}));

//Lớp hỗ trợ ví dụ
class ConNguoi
{
    public string Hoten { get; set; }
    public List<string> Dienthoai { get; set; }
}
//Liệt kê các sản phẩm có thể hiện Tên loại
var query = dsLoai
  .SelectMany(x=> x.SANPHAMs.Select(y=> new {x.TenLoai, y.TenSanpham, y.DonGia}));
```

2.3. Joining (kết nối)

- **Join**: Kết hợp các element của 2 sequence dựa trên khóa tương xứng.

```
string[] nhomsp = new string[]{"Rượu bia", "Ngũ cốc","Rau cải","Sữa","Hải sản"};
List<Sanpham> dssp = new List<Sanpham> {
      new Sanpham { Tensp = "Xà lách", Nhomsp="Rau cải" },
      new Sanpham { Tensp = "Cua biển", Nhomsp="Hải sản" },
      new Sanpham { Tensp = "Đậu xanh", Nhomsp="Ngũ cốc" } ,
      new Sanpham { Tensp = "Tôm càng", Nhomsp="Hải sản" }
};
//Tìm các element giống nhau từ 2 tập hợp, tạo ra một tập kết quả tương đương.
//còn gọi là liên kết tương đương
var query = nhomsp                          //outer collection
           .Join(dssp,                      //inner collection
                 keyo => keyo,              //outer key selector
                 keyi => keyi.Nhomsp,       //inner key selector
                 (o, i) => new { o, i });//result selector
//Class hỗ trợ
class Sanpham
{
       public string Tensp { get; set; }
       public string Nhomsp { get; set; }
}
//Ghi chú: Trong LINQ to Entities, Join thường sử dụng với bài toán thống kê, xem
trong Average
//Ví nếu chúng ta viết câu câu truy vấn sau:
var query = dsLoai.Join(dsSP, o => o.LoaispID, i => i.LoaispID,
```

```
                        (o, i) => new { o.TenLoai, i.TenSanpham, i.DonGia });
//thì tương đương với câu truy vấn này: (sẽ hay hơn)
var query = dsSP.Select(x => new {x.LOAISP.TenLoai, x.TenSanpham, x.DonGia });
//Hoặc dùng SelectMany như trong ví dụ mục trên
```

- **GroupJoin**: Kết hợp các element của 2 sequence dựa trên khóa tương đương và nhóm lại thành một kết quả.

  Tương đương với cú pháp Query syntax là **join ... into**

```
var query = nhomsp                          //outer collection
            .GroupJoin(dssp,                //inner collection
                  keyo => keyo,             //outer key selector
                  keyi => keyi.Nhomsp,      //inner key selector
 (o, i) => new { Tenhom=o, tensp= i.Select(x => x.Tensp) });//result selector
StringBuilder stb = new StringBuilder();
foreach (var n in query)
{
     stb.AppendLine(string.Format("{0}", n.Tenhom));
     foreach (string t in n.tensp)
           stb.AppendLine(string.Format("    {0}", t));
}
```

```
//Nhóm danh sách Sản phẩm và danh sách Loại sản phẩm thành một kết quả
var query = dsLoai.GroupJoin(dsSP, o => o.LoaispID, i => i.LoaispID,
(o, i) => new { o.TenLoai, spham = i.Select(y => new { y.TenSanpham, y.DonGia})} );
```

2.4. Ordering (sắp xếp)

- **OrderBy**: Sắp xếp các element theo thứ tự tăng dần theo khóa khai báo

```
var query = dsChuoi.OrderBy(x => x);
//không phân biệt chữ Hoa/thường
var query = dsChuoi.OrderBy(x => x, StringComparer.CurrentCultureIgnoreCase);
```

- **OrderByDescending**: Sắp xếp các element theo thứ tự giảm dần theo khóa khai báo
- **ThenBy**: Sắp xếp các element theo thứ tự tăng dần theo khóa khai báo, sau khi sắp xếp theo khóa trước đó

```
var query = dssp.OrderBy(x => x.Nhomsp).ThenBy(x => x.Tensp);
```

- **ThenByDescending**: Sắp xếp các element theo thứ tự giảm dần theo khóa khai báo, sau khi sắp xếp theo khóa trước đó
- **Reverse**: Đảo ngược thứ tự sắp xếp các element trong sequence

```
var query = dsSo.OrderBy(x => x).Reverse();
```

2.5. Grouping (Nhóm)

- **GroupBy**: Nhóm các element theo một bộ khóa cụ thể

```
var query = dssp.GroupBy(x => x.Nhomsp)
                .Select(x => new {x.Key, sosp = x.Count()});
//Cho biết số lượng sản phẩm của từng Loại, gồm: Mã loại sản phẩm, tên loại, Số lượng
var query = dsSP.GroupBy(x => new { x.LoaispID, x.LOAISP.TenLoai })
            .OrderBy(x => x.Key.LoaispID)
            .Select(x => new {x.Key.LoaispID, x.Key.TenLoai, soluong=x.Count() });
```

2.6. Set (tập hợp)

- **Concat:** Nối 2 sequence lại. Phương thức này tương đương với UNION ALL trong SQL

```
//Nối 2 chuỗi dsChuoi1 với dschuoi
List<string> dsChuoi1 = new List<string>()
{
     "Đã mang lấy nghiệp vào thân",
     "Cũng đừng trách lẫn trời gần trời xa."
};
var query = dsChuoi1.Concat(dsChuoi);
//Lấy những sản phẩm có khuyến mãi và những sản phẩm đã được đặt hàng
var query = dsCTDH.Select(x => new { x.SANPHAM.TenSanpham })
            .Concat(dsSP.Where(x => x.GiaKM > 0)
            .Select(x => new { x.TenSanpham }) ).OrderBy(x=> x.TenSanpham);
```

- **Union** : Hội kết quả của 2 sequence và loại bỏ các element trùng lắp.
  P.thức này tương đương với UNION trong SQL

```
//trả về kết quả chứa các số duy nhất của hai mảng
int[] dsSo2 = { 0, 2, 4, 5, 6, 8, 9 };
var query = dsSo.Union(dsSo2);
//Lấy những sản phẩm có khuyến mãi và những sản phẩm đã được đặt hàng, loại bỏ được các sản phẩm trùng lắp
var query = dsCTDH.Select(x => new { x.SANPHAM.TenSanpham })
            .Union(dsSP.Where(x => x.GiaKM > 0)
            .Select(x => new { x.TenSanpham })).OrderBy(x=> x.TenSanpham);
```

- **Intersect**: Hội kết quả của 2 sequence và chỉ lấy các element trùng lắp.

P.thức này tương đương với WHERE ... IN ... trong SQL

```
//trả về kết quả chứa các số giống nhau của hai mảng
int[] dsSo2 = { 0, 2, 4, 5, 6, 8, 9 };
var query = dsSo.Intersect(dsSo2);
//Lấy những sản phẩm vừa có khuyến mãi vừa được đặt hàng
var query = dsCTDH.Select(x => new { x.SANPHAM.TenSanpham })
            .Intersect(dsSP.Where(x => x.GiaKM > 0)
            .Select(x => new { x.TenSanpham })).OrderBy(x => x.TenSanpham);
```

- **Except**: Hội kết quả của 2 sequence và chỉ lấy các element nào của sequence thứ nhất mà không có trong sequence thứ 2.

  P.thức này tương đương với EXCEPT hoặc WHERE ... NOT IN ... trong SQL

```
//trả về kết quả chứa các số của mảng dsSo2 mà không có trong dsSo
int[] dsSo2 = { 0, 2, 4, 5, 6, 8, 9 };
var query = dsSo2.Except(dsSo);
//Lấy những sản phẩm được đặt hàng nhưng không có khuyến mãi
var query = dsCTDH.Select(x => new { x.SANPHAM.TenSanpham })
            .Except(dsSP.Where(x => x.GiaKM > 0)
            .Select(x => new { x.TenSanpham })).OrderBy(x => x.TenSanpham);
```

2.7. Conversion (import)

- **OfType**: Lọc các element kiểu IEnumerable thỏa điều kiện kiểu xác định.

  Nói cách khác là chuyển kiểu IEnumerable sang IEnumerable<T> và sẽ không chuyển các element khác kiểu T

```
//Lọc các element có kiểu string
ArrayList ds = new ArrayList()
{
    "Mango","Orange","Apple",3.0, "Banana", true, 100
};
IEnumerable<string> query = ds.OfType<string>();
```

- **Cast**: Chuyển kiểu các element kiểu IEnumerable sang kiểu xác định, tuy nhiên nó sẽ báo lỗi nếu không chuyển được vì có element sai kiểu

```
//nếu dùng lại ví dụ trên
IEnumerable<string> query = ds.Cast<string>(); //Lỗi run-time
//nhưng nếu khai báo lại nguồn dữ liệu như sau thì sẽ không còn lỗi
ArrayList ds = new ArrayList()
```

```
{
     "Mango","Orange","Apple", "Banana"
};
```

2.8. Conversion (Export)

- **ToArray**: Tạo ra một mảng từ một IEnumerable<T>

```
//tạo một mảng từ kiểu List<Sanpham>
var query = dssp.ToArray();   //Vừa tạo, vừa thi hành truy vấn
//tạo một mảng từ kiểu ObjectQuery<LOAISP>
var query = dsLoai.ToArray();
```

- **ToList**: Tạo ra một List<T> từ một IEnumerable<T>.

```
Sanpham[] dssp =
{
     new Sanpham { Tensp = "Xà lách", Nhomsp="Rau cải" },
     new Sanpham { Tensp = "Cua biển", Nhomsp="Hải sản" },
     new Sanpham { Tensp = "Đậu xanh", Nhomsp="Ngũ cốc" } ,
     new Sanpham { Tensp = "Tôm càng", Nhomsp="Hải sản" }
};
//tạo một List<Sanpham> từ kiểu mảng Sanpham
var query = dssp.ToList();
//tạo một List<string> từ kiểu mảng Sanpham
var query = dssp.Select(x => x.Tensp).ToList();
//tạo một List<LOAISP> từ kiểu ObjectQuery<LOAISP>
var query = dsLoai.ToList();
```

- **ToDictionary**: Tạo ra một Dictionary<TKey, TValue> từ một IEnumerable<T> dựa theo một bộ khóa xác định

```
//tạo một Dictionary<string, int> từ kiểu mảng
var spApple = (new[]
{     new  {Ten="iPhone",Dg=17}, new  {Ten="iPad",Dg=10},
      new  {Ten="iPod",Dg=2}, new  {Ten="Mac laptop",Dg=25},
}).ToDictionary(x => x.Ten);     //lấy Ten làm khóa, do đó nếu có giá trị trùng
nhau thì sẽ bị lỗi run-time
//tạo một Dictionary<int, string> từ kiểu ObjectQuery<LOAISP>
var query = dsLoai.Select(x => new {x.LoaispID, x.TenLoai }).ToDictionary(x=>
x.LoaispID);
```

- **ToLookup**: Tạo ra một Lookup<TKey, TElement> từ một IEnumerable<T> dựa theo một bộ khóa xác định.

  Điểm khác nhau so với ToDictionary là cho phép trùng Key

```
List<Sanpham> dssp = new List<Sanpham>
{
      new Sanpham { Tensp = "Xà lách", Nhomsp="Rau cải" },
      new Sanpham { Tensp = "Cua biển", Nhomsp="Hải sản" },
      new Sanpham { Tensp = "Đậu xanh", Nhomsp="Ngũ cốc" } ,
      new Sanpham { Tensp = "Tôm càng", Nhomsp="Hải sản" }
};
//câu lệnh sẽ bị lỗi lúc run-time vì trùng khóa
var query = dssp.ToDictionary(x => x.Nhomsp);
//Dòng lệnh này sẽ không bị lỗi
var query = dssp.ToLookup(x => x.Nhomsp);
```

Ví dụ khác: Nhóm theo ký tự đầu tiên của thuộc tính Nhomsp

```
//Cũng có thể dùng ILookup và nhóm các element lại:
ILookup<char, string> query = dssp.ToLookup(k =>  Convert.ToChar(
                                    k.Nhomsp.Substring(0, 1)), v => v.Tensp);
StringBuilder stb = new StringBuilder();
//Khi thi hành truy vấn, dùng kiểu IGrouping<char, string>
foreach (IGrouping<char, string> q in query)
{
   stb.AppendLine(q.Key.ToString());
   foreach(string str in q )
        stb.AppendLine("   " +str);
}
```

- **AsEnumerable**: Trả về input có kiểu IEnumerable<T>.

```
IEnumerable<string> query = dsChuoi.AsEnumerable().Where(x => x == "chữ");
```

- **AsQueryable** : Chuyển kiểu IEnumerable<T> sang kiểu IQueryable<T>

```
IQueryable<string> query = dsChuoi.AsQueryable().Where(x => x == "chữ");
```

2.9. Element (Phần tử)

- **First**: Trả về element đầu tiên trong sequence.

  Tương đương SELECT TOP 1...ORDER BY...

```
//Trả về chuỗi đầu tiên có ký tự bắt đầu bằng t trong dsChuoi
string query = dsChuoi.First(x => x.StartsWith("t"));

//Sản phẩm đầu tiên có đơn giá cao nhất
SANPHAM sp = dsSP.OrderByDescending(x => x.DonGia).First();
```

- **FirstOrDefault**: Trả về element đầu tiên trong sequence hoặc giá trị mặc định nếu sequence không chứa element nào

```
//Trả về null nếu không có element nào thỏa điều kiện
string query = dsChuoi.FirstOrDefault(x => x.StartsWith("z"));
//Trả về 0 nếu không có element nào có gái trị bằng 10
int query = dsSo.FirstOrDefault(x => x==10);
//Sản phẩm đầu tiên có đơn giá bằng 0
SANPHAM sp = dsSP.FirstOrDefault(x=> x.DonGia==0);
```

- **Last**: Trả về element cuối cùng trong sequence.
  Tương đương SELECT TOP 1...ORDER BY... DESC
- **LastOrDefalut**: Trả về element cuối cùng trong sequence hoặc giá trị mặc định nếu sequence không chứa element nào
- **Single**: Trả về chỉ duy nhất một element trong sequence và sẽ ném ngoại lệ nếu có hơn 1 element

```
//lỗi vì có nhiều element thỏa điều kiện
string query = dsChuoi.Single(x => x.StartsWith("t"));
//Lỗi vì dschuoi có nhiều element
string query = dsChuoi.Single();
//OK vì chỉ có 1 element thỏa điều kiện
string query = dsChuoi.Single(x => x.StartsWith("k"));
//Ok vì SanphamID là khóa chính
SANPHAM sp = dsSP.Single(x => x.SanphamID == 1);
```

- **SingleOrDefault**: Trả về chỉ duy nhất một element trong sequence hoặc giá trị mặc định nếu sequence rỗng. Nó sẽ ném ngoại lệ nếu có hơn một element

```
//Trả về null nếu không có element nào thỏa điều kiện
string query = dsChuoi.SingleOrDefault(x => x.StartsWith("z"));
//Trả về 0 nếu không có element nào có gái trị bằng 10
int query = dsSo.SingleOrDefault(x => x==10);
```

- **ElementAt**: Trả về element ở chỉ số xác định trong sequence

```
//lấy element thứ 2 trong dschuoi
string query = dsChuoi.ElementAt(1);
//SANPHAM sp = dsSP.ElementAt(1); //No OK
SANPHAM sp = dsSP.ToList().ElementAt(1); //OK vì hỗ trợ cho collection
```

- **ElementAtOrDefault**: Trả về element ở chỉ số xác định trong sequence hoặc giá trị mặc định nếu chỉ số nằm ngoài sequence

```
//Trả về 0 nếu không vì chỉ số 50 vượt quá số element trong sequence
int query = dsSo.ElementAtOrDefault(50);
```

- **DefaultIfEmpty**: Trả về các element của một sequence xác định, hoặc giá trị mặc định trong một collection nếu sequence rỗng

```
Pet defaultPet = new Pet { Name = "Default Pet", Age = 0 };
//Tạo một danh sách có phần tử
List<Pet> pets1 = new List<Pet>{ new Pet { Name="Barley", Age=8 },
     new Pet { Name="Boots", Age=4 }, new Pet { Name="Whiskers", Age=1 } };
var query = pets1.DefaultIfEmpty();
StringBuilder stb = new StringBuilder();
foreach (var s in query)
     stb.AppendLine(s.Name);
//Tạo một danh sách rỗng
List<Pet> pets2 = new List<Pet>();
//Sequence rỗng thì nó sẽ lấy defaultPet
var query2 = pets2.DefaultIfEmpty(defaultPet);
foreach (var s in query2)
    stb.AppendLine(s.Name);
```

Ví dụ trong LINQ to Entities:

```
//Những loại sản phẩm chưa có sản phẩm (tương đương Left/Right Join trong SQL)
var query = from loai in dsLoai
            join sp in dsSP on loai.LoaispID equals sp.LoaispID into kq
            from p in kq.DefaultIfEmpty()  //nhờ DefaultIfEmpty, kq sẽ chứa tất cả
loại sản phẩm, kể cả loại chưa có sản phẩm
            where p == null
            select new { loai.TenLoai, tensp = p == null ? "" : p.TenSanpham };
//ta có thể dùng GroupJoin thay thế join có .DefaultIfEmpty() như cách trên:
var query = dsLoai.GroupJoin(dsSP, o => o.LoaispID, i => i.LoaispID,
                          (o, i) => new { o.LoaispID, o.TenLoai, i })
                  .Where(x => x.i.Count() == 0)
                  .Select(x => new { x.LoaispID, x.TenLoai });
```

2.10. Aggregation (Thống kê)

- **Aggregate**: Tính toán tích lũy trong sequence (thống kê tùy ý)

```
//Bài toán sum
```

```
int kq = dsSo.Aggregate( (Sohientai, Soketiep) => Sohientai+Soketiep );
//Dùng tham số Seed để khởi tạo giá trị tích lũy ban đầu là 2
int kq = dsSo.Aggregate(2, (Sohientai, Soketiep) => (Sohientai + Soketiep) );
//Đảo ngược lại các element trong dsChuoi
string str = dsChuoi.Aggregate((chuoihientai, chuoiketiep) => chuoiketiep+ " " +
chuoihientai);
```

- **Average**: Tính trung bình của một sequence

```
//Tính trung bình các số trong dsSo
double tb = dsSo.Average();
//Tính đơn giá trung bình sản phẩm Apple
SpApple[] dssp = {new  SpApple {Ten="iPhone",Dg=17}, new SpApple
                    {Ten="iPad",Dg=10}, new SpApple {Ten="iPod",Dg=2}, new SpApple
                    {Ten="Mac laptop",Dg=25}
                  };
double tb = dssp.Average(x => x.Dg);
//Lớp hỗ trợ
class SpApple
{
    public string Ten { get; set; }
    public int Dg { get; set; }
}
//Đơn giá trung bình, lớn nhất, nhỏ nhất và số lượng sản phẩm của từng Loại sản
phẩm
var query = dsSP
           .GroupBy(x => new { x.LoaispID, x.LOAISP.TenLoai })
           .Select(x => new
            {
                x.Key.LoaispID,
                x.Key.TenLoai,
                dgmax = x.Max(d => d.DonGia),
                dgmin = x.Min(d => d.DonGia),
                dgtb = x.Average(d => d.DonGia),
                sosp = x.Count()
             }).OrderBy(x => x.LoaispID);
```

- **Count**: Trả về số lượng element trong sequence

```
List<Sanpham> dssp = new List<Sanpham> {
                new Sanpham { Tensp = "Xà lách", Nhomsp="Rau cải" },
                new Sanpham { Tensp = "Cua biển", Nhomsp="Hải sản" },
```

```
                new Sanpham { Tensp = "Đậu xanh", Nhomsp="Ngũ cốc" } ,
                new Sanpham { Tensp = "Tôm càng", Nhomsp="Hải sản" } };
//Đếm số lượng sản phẩm theo từng Nhomsp
var query = dssp.GroupBy(x => x.Nhomsp)
                .Select(x => new {x.Key, sosp= x.Count() });
//Đếm số lượng sản phẩm của Nhomsp là "Hải sản"
int dem = dssp.Count(x => x.Nhomsp == "Hải sản");
```

- **LongCount:** Trả về số lượng element trong sequence, kiểu dữ liệu là Int64

```
long dem = dssp.LongCount(x => x.Nhomsp == "Hải sản");
```

- **Sum**: Tính tổng của các element kiểu số trong một sequence

```
//Tính tổng trị giá các sản phẩm Apple
int tong = dssp.Sum(x => x.Dg);
//Tính tổng trong dsSo
int tong = dsSo.Sum();
//Tổng số lượng đặt hàng của từng sản phẩm
var query = dsCTDH.GroupBy(x => new { x.SanphamID, x.SANPHAM.TenSanpham})
                  .Select(x => new { x.Key.SanphamID, x.Key.TenSanpham, tongsl =
                  x.Sum(y => y.SoLuong) });
```

- **Max**: Trả về giá trị lớn nhất trong sequence
- **Min**: Trả về giá trị nhỏ nhất trong sequence

```
//Tìm số lớn nhất trong dsSo
int max = dsSo.Max();
//Tìm đơn giá nhỏ nhất trong danh sách sản phẩm Apple
int min = dssp.Min(x => x.Dg);
//Tìm giá trị nhỏ nhất trong dsChuoi
string str = dsChuoi.Min();
```

2.11. Quantifiers (định lượng, lượng hóa)

- **All** : Xác định tất cả các element của sequence có thỏa điều kiện hay không. Trả về kiểu bool

```
//Tất cả các element trong dsSo có lớn 5 không?
bool kq = dsSo.All(x => x > 5);
//Tất cả các sản phẩm trong nhóm sp có chứa chữ 'u'
List<Sanpham> dssp = new List<Sanpham> {
                new Sanpham { Tensp = "Xà lách", Nhomsp="Rau cải" },
                new Sanpham { Tensp = "Cua biển", Nhomsp="Hải sản" },
                new Sanpham { Tensp = "Đậu xanh", Nhomsp="Ngũ cốc" } ,
                new Sanpham { Tensp = "Tôm càng", Nhomsp="Hải sản" }
```

```
            };
var query = dssp.GroupBy(x => x.Nhomsp)
                .Where(x => x.All( y=> y.Tensp.Contains('u')));
```

- **Any**: Xác định sequence có chứa element nào thỏa điều kiện không. Trả về kiểu bool

```
//Xem dsSo có element nào không?
bool kq = dsSo.Any();   //true
//Trong nhóm sản phẩm tồn tại ít nhất 1 s.phẩm chứa chữ 'u'
var query = dssp.GroupBy(x => x.Nhomsp)
                .Where(x => x.Any(y => y.Tensp.Contains('u')));
```

- **Contains**: Xác định sequence có chứa element nào không dựa trên giá trị truyền vào. Trả về kiểu bool

(xem thêm ví dụ trên)

```
bool kq = dsChuoi.Contains("Tâm");  //True
```

- **SequenceEqual**: Xác định 2 sequence có bằng nhau không. Trả về kiểu bool

```
string[] dsA = { "cherry", "apple", "blueberry" };
string[] dsB = { "apple", "blueberry", "cherry" };
bool kq = dsA.SequenceEqual(dsB);   //False
```

2.12. Generation (phát sinh)

- **Empty**: Trả về một IEnumerable<T> rỗng có tham số được định kiểu

```
//Phát sinh một IEnumerable<string> rỗng
IEnumerable<string> empty = Enumerable.Empty<string>();
```

- **Range**: Phát sinh một sequence chứa các số nguyên trong một miền giá trị xác định

```
//Phát sinh một IEnumerable<int> có 3 số nguyên, bắt đầu từ 4
IEnumerable<int> squares = Enumerable.Range(4, 3).Select(x => x * x);
//Nếu in kết quả ra màn hình, ta thấy là: 16 25 36
```

- **Repeat**: Phát sinh một sequence chứa một giá trị được lặp lại

```
//Phát sinh một IEnumerable<string> chứa 10 chuỗi "Tôi thích lập trình"
IEnumerable<string> sts = Enumerable.Repeat("Tôi thích lập trình", 10);
```