



Phân tích thiết kế phần mềm **LINQ**

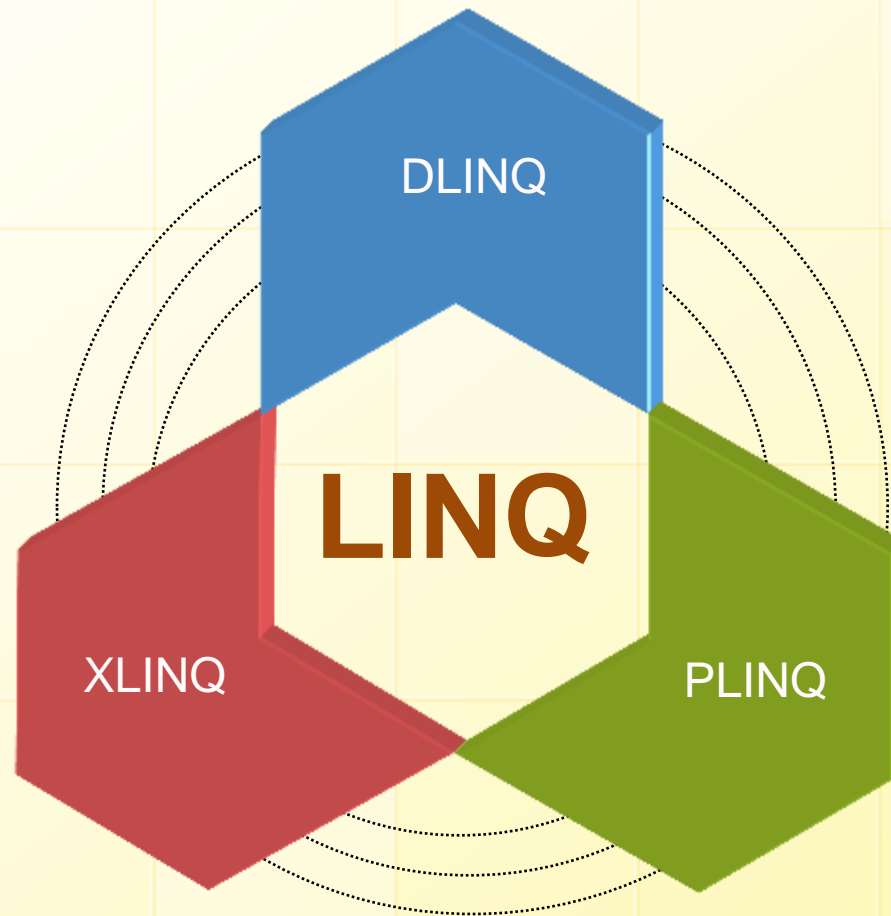
Ngô Ngọc Đăng Khoa

INTRODUCTION

Intro

- LINQ đọc là **LINK**, không phải **LIN-QUEUE**
- LINQ: **L**anguage **I**ntegrated **Q**uery
- LINQ cho phép developer thực hiện truy vấn trên nhiều dạng dữ liệu trong .NET
 - .NET Objects (List, Queue, Array, ...)
 - Database (**DLINQ**)
 - XML (**XLINQ**)
 - Parallel LINQ (**PLINQ**)

Intro



Intro

- Có thể bổ sung provider để mở rộng các nguồn dữ liệu mà LINQ có thể truy vấn

LINQ to NCover

LINQ to NHibernate

LINQ to Opf3

LINQ to Parallel (PLINQ)

LINQ to RDF Files

LINQ to Sharepoint

LINQ to SimpleDB

LINQ to Streams

LINQ to WebQueries

LINQ to WMI

LINQ to Excel

LINQ to Expressions

LINQ to Flickr

LINQ to Geo

LINQ to LLBLGen Pro

LINQ to Lucene

LINQ to Metaweb

LINQ to MySQL

C#

VB

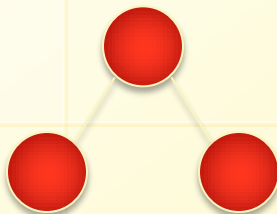
Others...

.NET Language Integrated Query

Standard
Query
Operators

DLinq
(ADO.NET)

XLinq
(System.Xml)



Objects



Database

```
<book>  
  <title/>  
  <author/>  
  <year/>  
  <price/>  
</book>
```

XML

LINQ TO OBJECTS

1st Example

```
List<int> list = new List<int>() {1, 2, 3};  
var query = from n in list  
             where n < 3  
             select n;  
  
foreach (var n in query)  
    Console.WriteLine(n);
```


.NET 3.0+ Features

Implicitly typed local variables

C# 3.0 Implicitly Typed Declaration	C# 2.0 Explicitly Typed Declaration
<pre>var Quantity = 12; var Quantity = 12; var UnitPrice = 55; var OrderDate = DateTime.Now; var ShippedDate = DateTime.Now; var Discount = 0.15; var numbers = new int[] { 0, 1, 2, 3, 4 };</pre>	<pre>int Quantity = 12; int Quantity = 12; int UnitPrice = 55; DateTime OrderDate = DateTime.Now; DateTime ShippedDate = DateTime.Now; double Discount = 0.15; int[] numbers = { 0, 1, 2, 3, 4 };</pre>

```
var query =  
    from n in list  
    where n < 3  
    select n;
```

```
IEnumerable<int> query =  
    from n in list  
    where n < 3  
    select n;
```

.NET 3.0+ Features

Collection Initializers

```
List<int> list = new List<int>();  
list.Add(1);  
list.Add(2);  
list.Add(3);
```



```
List<int> list = new List<int>() {1, 2, 3};
```

.NET 3.0+ Features

Dictionary Initializers

```
Dictionary<int, string> dic =  
    new Dictionary<int, string>();  
dic.Add(1, "value1");  
dic.Add(2, "value2");  
dic.Add(3, "value3");
```



```
Dictionary<int, string> dic =  
    new Dictionary<int, string> {  
        {1, "value1"}, {2, "value2"}  
    };
```

Query Syntax

```
from n in list  
where n < 3  
select n;
```



```
foreach (int n in list)  
{  
    if (n < 3) //xử lý n  
}
```

2nd Example


Truy vấn trên đối tượng

```
public class Customer
{
    public string CustomerID { get; set; }
    public string ContactName { get; set; }
    public string City { get; set; }
}
```

.NET 3.0+ Features

Automatic Properties

```
string _data;  
public string Data  
{  
    get { return _data; }  
    set { _data = value; }  
}
```



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

2nd Example (cont)

```
static List<Customer> GetCustomers()
{
    return new List<Customer> {
        new Customer { CustomerID = "ALFKI", ContactName =
"Maria Anders", City = "Berlin" },
        new Customer { CustomerID = "ANATR", ContactName = "Ana
Trujillo", City = "Mexico D.F." },
        new Customer { CustomerID = "ANTON", ContactName =
"Antonino Moreno", City = "Mexico D.F." }
    };
}
```


.NET 3.0+ Features

Object Initializers

```
class MyClass
{
    public string Prop1 { get; set; }
    public string Prop2 { get; set; }
}
```

```
MyClass c = new MyClass();
c.Prop1 = "Value1";
c.Prop2 = "Value2";
```

```
MyClass c = new MyClass {
    Prop1 = "Value1",
    Prop2 = "Value2"
};
```



2nd Example (cont)

```
var query = from c in GetCustomers()  
            where c.City == "Mexico D.F."  
            //where c.City.StartsWith("A")  
            select new {  
                City = c.City,  
                ContactName = c.ContactName  
            };
```

```
foreach (var c in query)  
{  
    //Xuất thông tin c  
}
```



IEnumerable<> (2 items)	
City	ContactName
Mexico D.F.	Ana Trujillo
Mexico D.F.	Antonino Moreno

.NET 3.0+ Features

Anonymous Type

```
var dude = new { Name = "Bob", Age = 25 };
```



```
internal class AnonymousGeneratedTypeName  
{  
    public string Name { get; set; }  
    public int Age { get; set; }  
}
```

```
AnonymousGeneratedTypeName dude =  
    new AnonymousGeneratedTypeName {Name = "Bob", Age = 25};
```

Query Syntax – let

```
var list =  
    new List<int> { 1,2,3,4,5,6,7,8,9 };
```

```
var query = from n in list  
            where n > 3 && n < 8  
            let g = n * 2  
            let newList = new List<int> {1,2,3}  
            from l in newList  
            select new { l, r = g * l };
```

Query Syntax – let

IEnumerable<> (12 items)	
l	r
1	8
2	16
3	24
1	10
2	20
3	30
1	12
2	24
3	36
1	14
2	28
3	42
24	264

Query Syntax – let

```
var query =  
    from l in File.ReadAllLines(path)  
    let parts = l.Split(';')  
    where parts[0] == server  
    select new {  
        Server = parts[0], Url = parts[1]  
    };
```

Query Syntax – join

Có ý nghĩa như phép kết bảng trong cơ sở dữ liệu quan hệ

```
var query =  
    from c in Categories  
    join p in Products on c.CategoryID equals  
        p.CategoryID  
    select new {c.CategoryName, p.ProductName};
```

Query Syntax – orderby

```
var query =  
    from m in typeof(string).GetMethods()  
    where m.IsStatic == true  
    orderby m.Name [descending]  
    select m.Name;
```

Query Syntax – group... by...

```
var query =  
    from m in typeof(string).GetMethods()  
    where m.IsStatic == true  
    orderby m.Name [descending]  
    group m by m.Name;
```

- **group** đã bao hàm ý nghĩa **select** nên không cần **select** nữa

Query Syntax – group... by...

Group nhiều thuộc tính

```
var query =  
    from p in Products  
    join c in Categories on p.CategoryID  
        equals c.CategoryID  
    group p by c.CategoryID, c.CategoryName;
```



Query Syntax – group... by...

Group nhiều thuộc tính

```
var query =  
    from p in Products  
    join c in Categories on p.CategoryID  
        equals c.CategoryID  
    group p by new {  
        c.CategoryID, c.CategoryName  
    };
```

Query Syntax – group... by... into...

```
var query =  
    from m in typeof(string).GetMethods()  
    where m.IsStatic == true  
    orderby m.Name [descending]  
    group m by m.Name into gr  
    select new {  
        Key = gr.Key, S1g = gr.Count()  
    };
```

Query Syntax – group... by... into...

IEnumerable<> (13 items)	
Key	Slg
Compare	10
CompareOrdinal	2
Concat	11
Copy	1
Equals	2
Format	5
Intern	1
IsInterned	1
IsNullOrEmpty	1
IsNullOrWhiteSpace	1
Join	5
op_Equality	1
op_Inequality	1
	42

Query Syntax

Các **from** có thể được viết lồng nhau

```
var query =  
    from list in lists  
    from num in list  
    select num;
```

Lambda Syntax

- Bản chất của LINQ là các lệnh truy vấn được viết dưới dạng **lambda syntax**
- Query syntax dễ đọc, dễ hiểu hơn so với lambda syntax
- Khi thực thi, query syntax sẽ được compiler chuyển về lambda syntax
- Dùng **lambda syntax** mới có thể tận dụng được hết sức mạnh của LINQ

Lambda Syntax

- Các truy vấn LINQ được viết bằng **query syntax** hoàn toàn có thể được biểu diễn dưới dạng **lambda syntax**
 - Không có chiều ngược lại
- Nên kết hợp **query syntax** & **lambda syntax**.

Lambda Expression

- Có ý nghĩa như con trỏ hàm trong C++
- .NET 2.0 giới thiệu ***Anonymous Methods*** nhằm cài đặt thuận tiện hơn
- ***Lambda Expression*** là phiên bản cải tiến của Anonymous Methods
- Cấu trúc ngắn gọn

argument-list => **expression**

Example – Delegate

```
delegate int MyFunc(int a, int b);

static int Func(int a, int b)
{
    int x = a * a;
    int y = b * b;
    return x + y;
}

static void Main(string[] args)
{
    MyFunc f = Func;
    int s = f.Invoke(2, 3);

    Console.WriteLine(s);
}
```

Example – Anonymous Method

```
delegate int MyFunc(int a, int b);

static void Main(string[] args)
{
    MyFunc f = delegate(int a, int b)
    {
        int x = a * a;
        int y = b * b;
        return x + y;
    };
    int s = f.Invoke(2, 3);

    Console.WriteLine(s);
}
```

Example – Lambda Expression

```
delegate int MyFunc(int a, int b);

static void Main(string[] args)
{
    MyFunc f = (a, b) =>
    {
        int x = a * a;
        int y = b * b;
        return x + y;
    };
    int s = f.Invoke(2, 3);

    Console.WriteLine(s);
}
```

Lambda Expression

```
public static int Add(int a, int b)
{
    return a + b;
}
```



```
(a, b) => a + b //Func<int, int, int>
```

Lambda Expression

```
List<int> numbers=GetNumbers();

//find the first number in the list that is below 10
int match=numbers.Find(n=> n<10);

//print all the numbers in the list to the console
numbers.ForEach(n=> Console.WriteLine(n));

//convert all the numbers in the list to floating-point values
List<float> floatNumbers=numbers.ConvertAll<float>(n=> (float)n);

//sort the numbers in reverse order
numbers.Sort((x, y) => y-x);

//filter out all odd numbers
numbers.RemoveAll(n=> n%2!=0);
```

Lambda Expression

```
public delegate TResult Func<TResult>();  
public delegate TResult Func<T, TResult>(T a);  
public delegate TResult Func<T1, T2, TResult>(T1 a, T2 b);  
public delegate TResult Func<T1, T2, T3, TResult>(T1 a, T2 b, T3 c);  
public delegate TResult Func<T1, T2, T3, T4, TResult>(T1 a, T2 b, T3 c,  
                                                    T4 d);
```

```
public delegate void Action();  
public delegate void Action<T>(T a);  
public delegate void Action<T1, T2>(T1 a, T2 b);  
public delegate void Action<T1, T2, T3>(T1 a, T2 b, T3 c);  
public delegate void Action<T1, T2, T3, T4>(T1 a, T2 b, T3 c, T4 d);
```

QUERY OPERATORS

List of Operators

Type		Name		Type		Name	
Partitioning	Take, Skip			Set	Distinct		
	TakeWhile				Concat, Union		
	SkipWhile				Intersect, Except		
Join	Join, GroupJoin			Conversion	AsEnumerable		
Ordering	OrderBy				ToArray, ToList		
	OrderByDescending				ToDictionary, ToLookup		
	ThenBy, Reverse				OfType, Cast		
Projection	Select, SelectMany			Element	First, FirstOrDefault		
Grouping	GroupBy				Last, LastOrDefault		
Restriction	Where				Single, SingleOrDefault		
Equality	SequenceEqual				ElementAt		
					ElementAtOrDefault		

List of Operators (cont)

Type	Name	Type	Name
Aggregate	Count, LongCount	Generation	Any, All
	Sum, Min, Max		Contains
	Average, Aggregate		Range, Repeat, Empty

Restriction Operators

Where: giữ lại các phần tử thoả điều kiện

Query Syntax

```
var query = from n in list
             where n < 3
             select n;
```

Lambda Syntax

```
var query = list.Where(n => n < 3);
```

Projection Operators

Select

Query Syntax

```
var query = from c in GetCustomers()  
            where c.City.StartsWith("A")  
            select new { c.City, c.ContactName };
```

Lambda Syntax

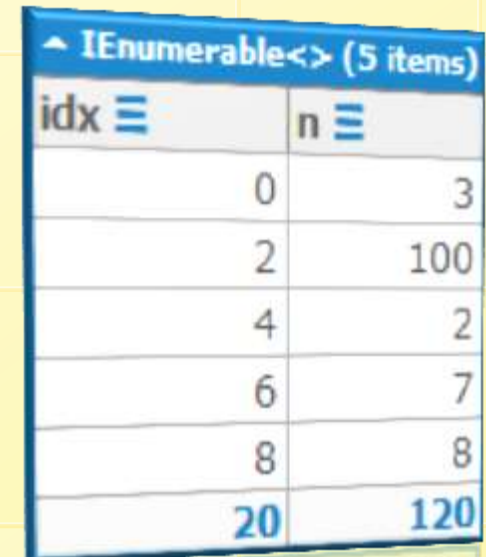
```
var query =  
    GetCustomers()  
    .Where(c => c.City.StartsWith("A"))  
    .Select(c => new { c.City, c.ContactName });  
//.Select(c => c);
```

Projection Operators

Select (có index)

Lambda Syntax

```
int[] numbers = { 3, 9, 100, 4, 2, 6, 7, 1, 8 };  
var query = numbers  
    .Select((n, idx) => new {idx, n})  
    .Where(item => item.idx % 2 == 0);
```



idx	n
0	3
2	100
4	2
6	7
8	8
20	120

Projection Operators

SelectMany: dùng “phẳng hoá” tập hợp

```
public class NewItem
{
    public string Category { get; set; }
    public string Name { get; set; }
    public List<string> Ingredients { get; set; }
    public double Price { get; set; }
}
```

SelectMany

```
private static List<NewItem> GetNewItemList()
{
    List<NewItem> itemList = new List<NewItem> {
        new NewItem
        {
            Category="Icecreams", Name="Chocolate Fudge Icecream",
            Ingredients = new List<string> {"cream", "milk", "mono and diglycerides"},
            Price=10.5
        },
        new NewItem
        {
            Category="Icecreams", Name="Vanilla Icecream",
            Ingredients= new List<string> {"vanilla extract", "guar gum", "cream"},
            Price=9.80
        },
        new NewItem
        {
            Category="Icecreams", Name="Banana Split Icecream",
            Ingredients= new List<string> {"Banana", "guar gum", "cream"},
            Price=7.5
        }
    };
    return itemList;
}
```

SelectMany

```
var itemss = GetNewItemssList();  
var ingredients = itemss.Select(ing => ing.Ingredients);
```

IEnumerable<List<String>> (3 items)	
List<String> (3 items)	cream
	milk
	mono and diglycerides
List<String> (3 items)	vanilla extract
	guar gum
	cream
List<String> (3 items)	Banana
	guar gum
	cream

SelectMany

```
var itemss = GetNewItemssList();  
var ingredients = itemss.SelectMany(ing => ing.Ingredients);
```

▲ IEnumerable<String> (9 items)

cream

milk

mono and diglycerides

vanilla extract

guar gum

cream

Banana

guar gum

cream

Join

Query Syntax:

```
var query =  
    from c in Categories  
    join p in Products on c.CategoryID equals p.CategoryID  
    select new {c.CategoryName, p.ProductName};
```

Lambda Syntax:

```
var query = Categories.Join(  
    Products,  
    c => c.CategoryID,  
    p => p.CategoryID,  
    select new {c.CategoryName, p.ProductName}  
);
```

Join – Multiple Fields

Query Syntax:

```
var query =  
    from s in ShoppingMalls  
    join h in Houses on  
        new { s.CouncilCode, s.PostCode }  
    equals new { h.CouncilCode, h.PostCode }  
    select s;
```

Join – Multiple Fields

Lambda Syntax:

```
var query =  
    ShoppingMalls.Join(  
        Houses,  
        s => new { s.CouncilCode, s.PostCode },  
        h => new { h.CouncilCode, h.PostCode },  
        (s,h) => s  
    );
```

Ordering Operators

OrderBy, OrderByDescending

```
var query2 = from m in list  
              orderby m.Name descending  
              select m;
```

```
var query2 = list.OrderByDescending(m => m.Name);
```

Ordering Operators

ThenBy

```
var query4 = from m in list  
              orderby m.Name, m.Instrument  
              select m;
```

```
var query4a = list.OrderBy(m => m.Name).ThenBy(m => m.Instrument);
```

Ordering Operators

Reverse: đảo dãy

```
List<int> list = new List<int> { 1, 2, 3 };  
list.Reverse();
```

{ 3, 2, 1 }

GroupBy

Query Syntax:

```
var query =  
    from p in Products  
    join c in Categories on p.CategoryID  
        equals c.CategoryID  
    group p.ProductName by c.CategoryName;
```

GroupBy

Lambda Syntax:

```
var query = Products.Join(  
    Categories,  
    p => p.CategoryID,  
    c => c.CategoryID,  
    (p, c) => new { p.ProductName, c.CategoryName }  
)  
.GroupBy(i=>i.CategoryName, i=>i.ProductName);
```


GroupBy

▲ IOrderedQueryable<IGrouping<String,String>> (8 items)	
Key=Beverages	
▲ IGrouping<String,String> (12 items)	
Chai	
Chang	
Guaraná Fantástica	
Sasquatch Ale	
Steeleye Stout	
Côte de Blaye	
Chartreuse verte	
Ipoh Coffee	
Laughing Lumberjack Lager	
Outback Lager	
Rhönbräu Klosterbier	
Lakkalikööri	
Key=Condiments	
▲ IGrouping<String,String> (12 items)	
Aniseed Syrup	
Chef Anton's Cajun Seasoning	

GroupBy

Query Syntax:


```
var query =  
    from p in Products  
    join c in Categories on p.CategoryID  
        equals c.CategoryID  
    group p by new { c.CategoryID, c.CategoryName }  
        into grpRow  
    select new {  
        grpRow.Key.CategoryID,  
        grpRow.Key.CategoryName,  
        I = grpRow.Count()  
    };
```

GroupBy

Lambda Syntax:

```
var query = Products.Join(  
    Categories,  
    p => p.CategoryID,  
    c => c.CategoryID,  
    (p, c) => new { p, c.CategoryID, c.CategoryName }  
)  
.GroupBy(c => new { c.CategoryID, c.CategoryName })  
.Select(grpRow => new {  
    grpRow.Key.CategoryID,  
    grpRow.Key.CategoryName,  
    I = grpRow.Count()  
}  
);
```

GroupBy

▲ IOrderedQueryable<> (8 items)		
CategoryID	CategoryName	I 
1	Beverages	12
2	Condiments	12
3	Confections	13
4	Dairy Products	10
5	Grains/Cereals	7
6	Meat/Poultry	6
7	Produce	5
8	Seafood	12
		77

Generation Operators

Range: tạo 1 dãy số nguyên liên tiếp

```
public static IEnumerable<int> Range(int start, int count);
```

```
var query = from x in Enumerable.Range(1, 2)
            from y in Enumerable.Range(1, 3)
            select new { x, y };
```

```
{ x = 1, y = 1 }
{ x = 1, y = 2 }
{ x = 1, y = 3 }
{ x = 2, y = 1 }
{ x = 2, y = 2 }
{ x = 2, y = 3 }
```

Generation Operators

Repeat: tạo 1 dãy số chỉ chứa duy nhất 1 giá trị

```
var list = Enumerable.Repeat(108, 12);
```

Empty: tạo 1 dãy số có 0 phần tử

```
var list = Enumerable.Empty<double>();  
Console.WriteLine(list);
```



Generation Operators

Any

- Dùng để kiểm tra dãy có rỗng hay không?

```
var listA = Enumerable.Empty<double>();  
var listB = Enumerable.Range(1, 10);  
  
Console.WriteLine("Are there any items in ListA: {0}, ListB: {1}",  
    listA.Any(), listB.Any());
```

Generation Operators

Any

- Dùng để kiểm tra dãy có chứa phần tử nào thoả điều kiện X hay không?

```
var listA = Enumerable.Empty<double>();  
var listB = Enumerable.Range(1, 10);  
  
Console.WriteLine("Does listB contain the number {0}: {1}",  
    8, listB.Any(i => i == 8));
```


Generation Operators

All

- Dùng để kiểm tra dãy có phải tất cả phần tử của dãy đều thoả điều kiện X hay không?

```
var list = Enumerable.Range(1, 10);

if (list.All(i => i < 11))
{
    Console.WriteLine("Condition met");
}
else
{
    Console.WriteLine("Condition not met");
}
```

Partitioning Operators

Take: lấy n phần tử đầu tiên trong dãy

```
var query1 = (from r in romans
               where r.Gender == 'm'
               select r.Name).Take(2);
```

Skip: bỏ qua n phần tử đầu tiên trong dãy, lấy từ phần tử thứ (n+1)

```
query1 = (from r in romans
           where r.Gender == 'm'
           select r.Name).Skip(2);
```

Partitioning Operators

TakeWhile: lấy các phần tử đầu cho tới khi thoả điều kiện

```
var query = Enumerable  
    .Range(1, 100)  
    .Where(x => x % 3 == 0)  
    .TakeWhile(x => x % 11 != 0);
```

3, 6, 9,
12, 15, 18,
21, 24, 27, 30

SkipWhile: bỏ các phần tử đầu cho tới khi thoả điều kiện

Hot Tip

- Ta có thể kết hợp **Take/ TakeWhile** & **Skip/ SkipWhile** để thực hiện tính năng phân trang dữ liệu.
 - Nguồn dữ liệu có nhiều records
 - Thực hiện phân trang, mỗi trang 10 records
 - Lấy ra các dữ liệu thuộc trang 2

```
var query = dataSrc.Skip(10).Take(10);
```

Element Operators

First: lấy phần tử đầu tiên trong dãy, “thả” **InvalidOperationException** khi dãy rỗng

```
var query = (from r in romans
              where r.Gender == 'm'
              select r.Name).First();
```

```
var firstLambda = (from r in romans
                    where r.Gender == 'm'
                    select r.Name).First(n => n.Length > 4);
```

Element Operators

FirstOrDefault: tương tự như First nhưng trả về **null** & ko "thả" exception khi dãy rỗng

```
var query = (from r in romans
              where r.Gender == 'z'
              select r.Name).FirstOrDefault();
```

Last, LastOrDefault

Element Operators

Single:

- Trả về duy nhất 1 item trong dãy có duy nhất 1 phần tử.
- “Thả” exception khi dãy có nhiều hơn 1 phần tử
- Dùng **Single** để ép dãy có 1 phần tử về đối tượng cụ thể

Single

```
string name = (from r in romans  
               where r.Name.Length == 4  
               select r.Name).Single();
```

```
string name = (from r in romans  
               where r.Gender == 'm'  
               select r.Name).Single(r => r.Length == 4);
```


First vs. Single vs. Take(1)

- **First** trả về phần tử đầu tiên trong dãy có ≥ 1 phần tử
- **Single** ép dãy có duy nhất 1 phần tử thành kiểu đối tượng cụ thể. Khi dùng **Single** ta đã hàm ý việc kiểm tra xem dãy có chứa nhiều hơn 1 phần tử hay không?
- **Take(1)** trả về 1 dãy có 1 phần tử lấy từ dãy gốc

Element Operators

ElementAt: lấy phần tử thứ i trong dãy

```
query = (from r in romans  
         where r.Gender == 'm'  
         select r.Name).ElementAt(2);
```

DefaultIfEmpty

Khi kết quả truy vấn là dãy 0 phần tử, **DefaultIfEmpty** sẽ tạo ra *1 phần tử mặc định* cho dãy

```
var query = (from r in romans
              where r.Gender == 'z'
              select r).DefaultIfEmpty();
```



NULL

(Kết quả là dãy có 1 phần tử, phần tử đó = null)

DefaultIfEmpty

Tự định nghĩa phần tử mặc định

```
var query = (from r in romans
              where r.Gender == 'z'
              select r)
              .DefaultIfEmpty(new Roman
                              {
                                  Id = -1,
                                  Gender = 'N',
                                  Name = "Empty Roman"
                              });
```

```
{ Id = -1; Gender = N; Name = Empty Roman }
```

Set Operators

Union: kết hợp 2 dãy cùng kiểu dữ liệu lại & loại bỏ các phần tử trùng

```
var listA = Enumerable.Range(1, 3);  
var listB = new List<int> { 3, 4, 5, 6 };  
  
var listC = listA.Union(listB);
```

{ 1, 2, 3, 4, 5, 6 }

Set Operators

Concat: kết hợp 2 dãy cùng kiểu dữ liệu lại & không loại bỏ các phần tử trùng

```
var listA = Enumerable.Range(1, 3);  
var listB = new List<int> { 3, 4, 5, 6 };  
  
var listC = listA.Union(listB);
```

{ 1, 2, 3, 3, 4, 5, 6 }

Set Operators

Distinct: loại bỏ các phần tử trùng trong dãy

```
var listA = new List<int> { 1, 2, 3, 3, 2, 1 };  
var listB = listA.Distinct();
```

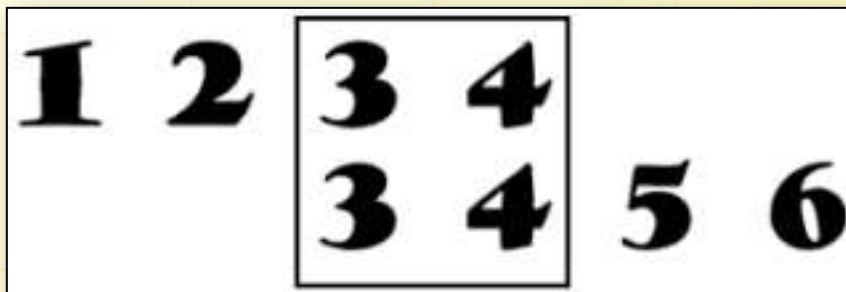
Distinct

1 2 3 3 2 1 → 1 2 3

Set Operators

Intersect: lấy phần giao của 2 dãy có cùng kiểu dữ liệu

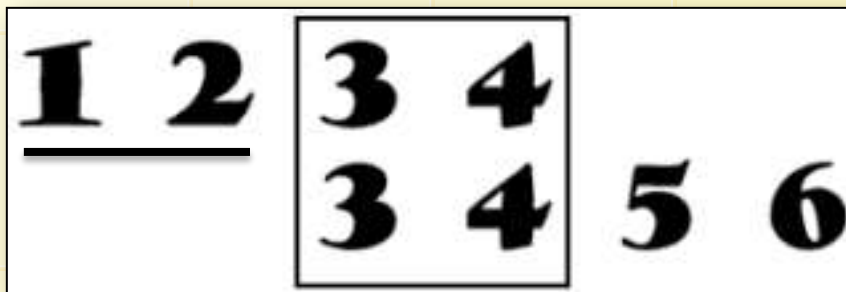
```
var listA = Enumerable.Range(1, 4);  
var listB = new List<int> { 3, 4, 5, 6 };  
var listC = listA.Intersect(listB);
```



Set Operators

Except: lấy các phần tử thuộc dãy 1 & không chứa phần giao của 2 dãy

```
var listA = Enumerable.Range(1, 6);  
var listB = new List<int> { 3, 4 };  
  
var listC = listA.Except(listB);
```



Aggregate Operators

Count: trả về số lượng phần tử có trong dãy

```
var list = Enumerable.Range(5, 12);  
Console.WriteLine(list.Count());
```

Có thể chỉ định điều kiện **Count**

```
var list = Enumerable.Range(1, 25);  
  
Console.WriteLine("Total Count: {0}, Count the even numbers: {1}",  
    list.Count(),  
    list.Count(n => n % 2 == 0));
```

Aggregate Operators

Min, Max: trả về phần tử nhỏ nhất, lớn nhất trong dãy

```
var list = Enumerable.Range(6, 10);  
Console.WriteLine("Min: {0}, Max: {1}", list.Min(), list.Max());
```

Có thể chỉ định thuộc tính để lấy min, max

```
double maxPrice =  
    Products.Max(p => p.UnitPrice)
```

Aggregate Operators

Average: tính giá trị trung bình của dãy

```
var list = Enumerable.Range(0, 5);  
Console.WriteLine("Average: {0}", list.Average());
```

Có thể chỉ định thuộc tính để tính trung bình

```
List<Item> items = GetItems();  
  
double averageValue = items.Average(v => v.Length + v.Width);  
Console.WriteLine("AverageValue: {0}", AverageValue);
```

Aggregate Operators

Sum: tính tổng của dãy

```
var list = Enumerable.Range(5, 3);  
Console.WriteLine("List sum = {0}", list.Sum());
```

Có thể chỉ định thuộc tính để tính tổng

```
var items = GetItems();  
Console.WriteLine("Sum the lengths of the items: {0}",  
    items.Sum(x => x.Length + x.Width));
```

Conversion Operators

ToList, ToArray: chuyển kết quả truy vấn sang List, Array

```
List<int> list = (from n in Enumerable.Range(1, 3)
                  where n < 3
                  select n).ToList();
```

```
int[] data = (from num in list
               where num < 3
               select num).ToArray();
```


Conversion Operators

ToDictionary: chuyển kết quả truy vấn sang Dictionary, khi dùng hàm này cần chỉ định thuộc tính **KEY** cho Dictionary

```
var query = from r in romans
             select new { r.Name, r.Gender, r.Id };

var romanDictionary = query.ToDictionary(r => r.Name);

Console.WriteLine(romanDictionary["Augustus"]);
Console.WriteLine(romanDictionary["Livia Drusilla"]);
```

Conversion Operators

OfType: lấy ra các phần tử thuộc kiểu dữ liệu nào đó trong dãy

```
ArrayList list = new ArrayList { 1, "That", 2, "This" };  
  
IEnumerable<string> elist = list.OfType<string>();  
  
var query = from num in elist  
            select num;
```

“That”, “This”

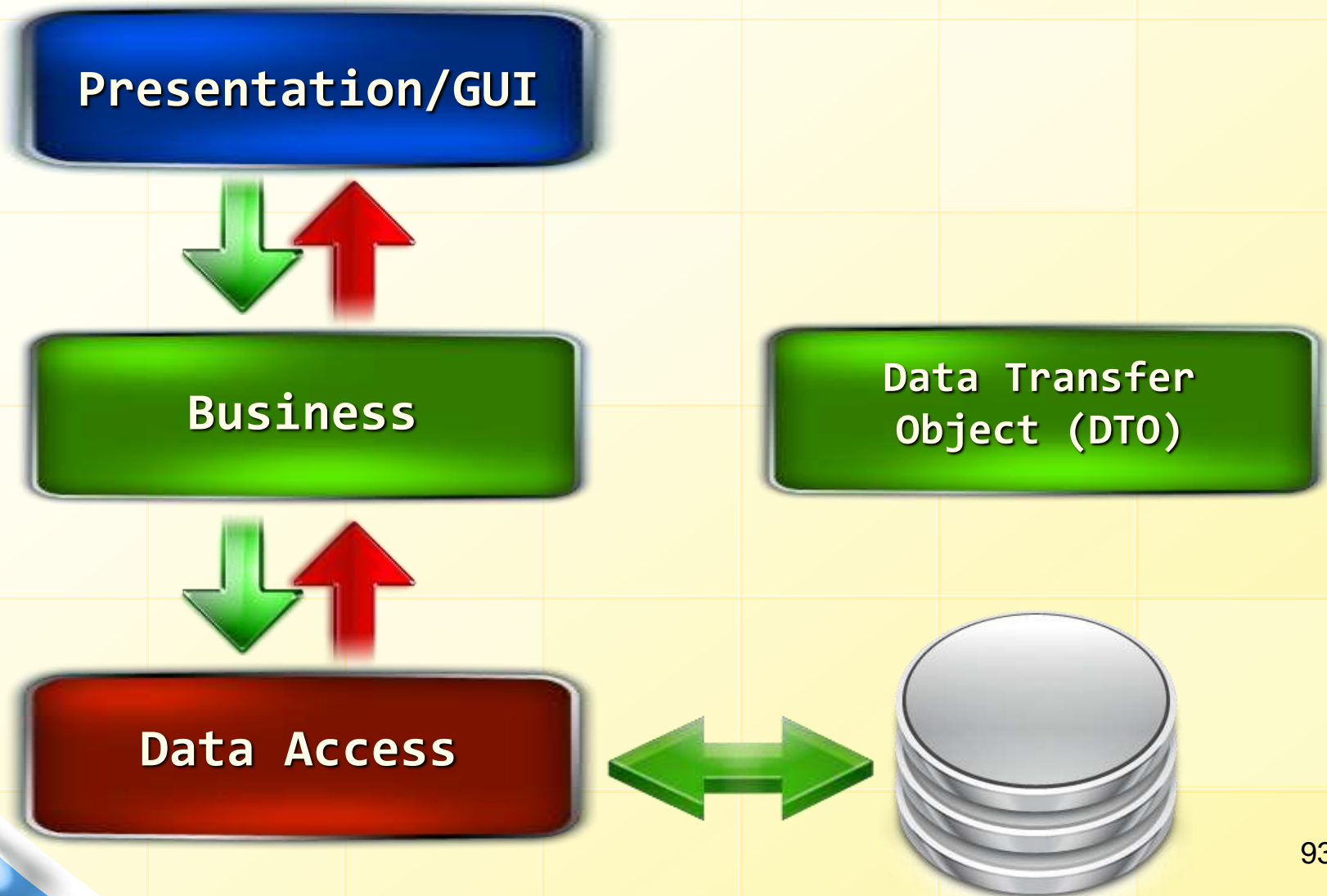
LINQ TO SQL

(DLINQ)

Persistence

- Ứng dụng thường có nhu cầu lưu lại dữ liệu.
- Dữ liệu có thể là file text, xml, **cơ sở dữ liệu quan hệ**, ...
- Trong phần mềm hướng đối tượng, dữ liệu cần lưu là các objects
 - Lưu trữ tình trạng hiện tại
 - Có khả năng tái tạo lại tình trạng đã được lưu

Persistence



Persistence

- Các hướng tiếp cận trong .NET
 - DataSets
 - Hand-coding
 - ORM (**DLINQ**, NHibernate, ...)

ORM

- Lập trình hướng đối tượng là hướng tiếp cận tốt để xây dựng ứng dụng phức tạp
- ORM là cầu nối giúp dễ dàng chuyển đổi các đối tượng xuống CSDL quan hệ và ngược lại
- ORM hỗ trợ các tính năng: caching, transaction, concurrency control

ORM

- Developer chỉ cần quan tâm tới việc **ánh xạ các đối tượng sang CSDL**
- **LINQ to SQL (DLINQ)** là 1 công cụ ORM


Entity Class

- Ánh xạ **class** sang table thông qua các **attribute**
 - Class \Leftrightarrow Table
 - Property \Leftrightarrow Field

```
using System.Data.Linq;  
using System.Data.Linq.Mapping;  
  
[Table(Name="Customers")]  
public class Customer  
{  
    [Column(IsPrimaryKey=true)]  
    public string CustomerID;  
  
    [Column]  
    public string City;  
}
```

DataContext

- Là đối tượng chủ chốt trong DLINQ
- Quản lý tất cả các thao tác CRUD xuống CSDL



```
// DataContext takes a connection string
DataContext db = new DataContext("c:\\northwind\\northwnd.mdf");

// Get a typed table to run queries
Table<Customer> Customers = db.GetTable<Customer>();

// Query for customers from London
IQueryable<Customer> CustomerQuery = from c in Customers
                                     where c.City == "London"
                                     select c;
```


Relationships

Ánh xạ **quan hệ 1-n** trong CSDL quan hệ

- Sử dụng attribute **Association** ở cả 2 class
- Class [1] định nghĩa **OtherKey**
- Class [n] định nghĩa **ThisKey**

Relationships

```
[Table(Name = "Customers")]
public class Customer
{
    [Column(IsPrimaryKey = true)]
    public string CustomerID;

    EntitySet<Order> _orders;

    [Association(Storage = "_orders", OtherKey = "CustomerID")]
    public EntitySet<Order> Orders
    {
        get { return _orders; }
        set { _orders.Assign(value); }
    }
}
```

Relationships

```
[Table(Name="Orders")]
public class Order
{
    [Column(IsPrimaryKey = true)]
    public int OrderID;

    [Column]
    public string CustomerID;

    EntityRef<Customer> _customer;

    [Association(Storage = "_customer", ThisKey = "CustomerID")]
    public Customer Customer
    {
        get { return _customer.Entity; }
        set { _customer.Entity = value; }
    }
}
```

Hot Tip

- Có thể ánh xạ thông qua các *attribute* hoặc file viết file ánh xạ dạng xml (*.dbml*)

Mapping (command-line)

- Sử dụng file công cụ **sqlmetal** để generate file ánh xạ (.dbml)
 - *Program Files\Microsoft SDKs\Windows\v6.0A\bin\SqlMetal.exe*
- Cách sử dụng **sqlmetal**



Mapping (Visual Studio 2008)



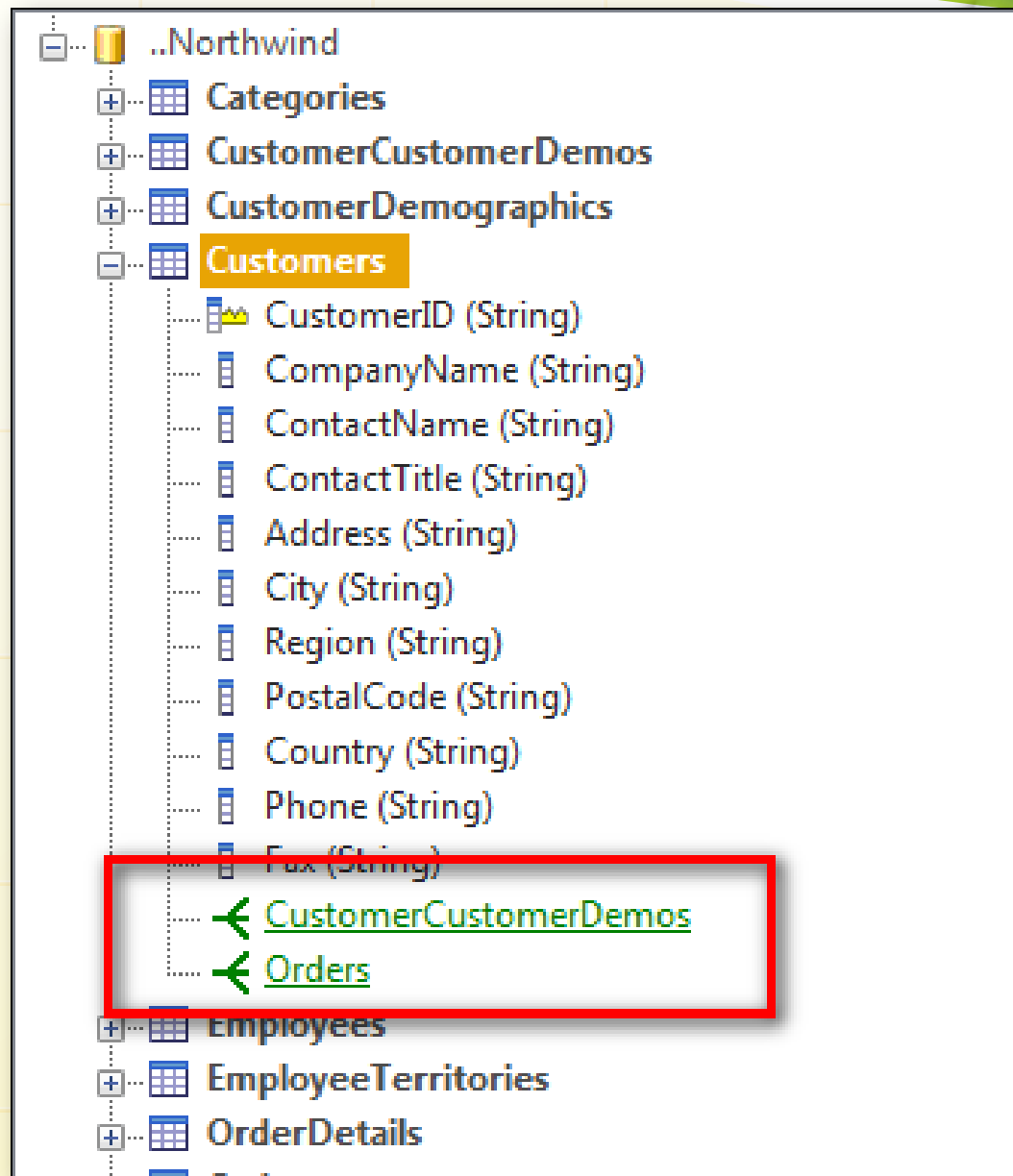
Hot Tip

- Có thể thay đổi chuỗi kết nối tới CSDL lúc runtime
 - Viết hàm partial **OnCreated** cho lớp **DataContext**

```
partial class NorthwindDataContext
{
    partial void OnCreated()
    {
        this.Connection.ConnectionString
            = "Server=.; Database=Northwind; Integrated Security=SSPI";
    }
}
```

Mapping (Visual Studio 2008)

- Nếu khi thực hiện thao tác ánh xạ, CSDL đã có cài đặt khoá ngoại thì *Visual Studio* tự động add các *entityRef* & *entitySet* vào các *Entity*
 - Không cần thực hiện **JOIN** khi cần truy vấn thông tin trên nhiều table.



Querying Database

- Khai báo ***dataContext***
- Đối tượng DataContext có các thuộc tính ứng với các table dưới CSDL
 - db.Customers
 - db.Categories
 - ...
- Các thuộc tính này chính là nguồn dữ liệu cho các truy vấn LINQ

Querying Database

```
var OrdersQuery = from o in db.Orders  
                   where o.ShipVia == 1  
                   select o;
```

```
NorthwindDataContext db = new NorthwindDataContext();  
dataGridView1.DataSource = db.Products;
```

- Truy vấn chỉ được thực khi nào thực sự dùng đến
 - Duyệt kết quả truy vấn
 - Gán lên control

Hot Tip

- Nếu cần dùng đến kết quả truy vấn >1 lần, nên cache kết quả truy vấn lại → *ToList/ToArray*

```
var CustomerQuery = from c in db.Customers
                    where c.Country == "Spain"
                    select c;

// Some more work ...
// GetEnumerator() causes query translation, SQL query execution
// and object materialization.
foreach(Customer c in CustomerQuery) {...}
// The query is executed again
foreach(Customer c in CustomerQuery) {...}
```

```
List<Customer> CustomerList = CustomerQuery.ToList();
```

Compiled Queries

- Nhu cầu: dùng 1 câu query LINQ nhiều lần nhưng khác tham số
- Vd:
 - Hiển thị danh sách học sinh của lớp
 - Hiển thị danh sách hoá đơn của khách hàng
- Giải pháp:
 - Viết nhiều câu query → tốn kém chi phí chuyển đổi truy vấn LINQ sang truy vấn SQL
 - **Sử dụng Compiled Query**: thích hợp cho web

Compiled Queries

```
// Define a compiled query
var OrdByCustId = CompiledQuery.Compile(
    (NorthwindDataContext context, string custId) =>
        from o in context.Orders
        where o.CustomerID == custId
        select o);

// Execute the compiled query
string cust = "276AROUT";
var q = OrdByCustId(db, cust);
```

Compiled Queries

```
static class Queries
{
    public static Func<NorthwindDataContext, string, IQueryable<Order>>
        OrdByCustId = CompiledQuery.Compile(
            (NorthwindDataContext context, string custId) =>
                from o in context.Orders
                where o.CustomerID == custId
                select o);

    // More compiled queries for common query patterns
}
```


Modifying & Saving Entities

- Thay đổi dữ liệu trực tiếp lên các Entities
- Các hàm thay đổi dữ liệu
 - **InsertOnSubmit**: thêm 1 entity
 - **DeleteOnSubmit**: xóa 1 entity
 - **DeleteAllOnSubmit**: xóa tất cả entities thoả điều kiện
- Gọi hàm **DataContext.SubmitChanges()** để lưu các thay đổi xuống CSDL

Modifying & Saving Entities

```
NorthwindDataContext db = new NorthwindDataContext();
```

```
Product p = new Product();
```

```
//Set properties for new Product
```

```
//...
```

```
//Add p to the Products collection
```

```
db.Products.InsertOnSubmit(p);
```

```
db.SubmitChanges();
```

Modifying & Saving Entities

```
NorthwindDataContext db = new NorthwindDataContext(connectionString);
db.Log = Console.Out;

// Retrieve single customer with given ID - AROUT
Customer cust = (from c in db.Customers
                  where c.CustomerID == "AROUT"
                  select c).Single();

Order ord = new Order();
// Set properties for the new order
...
// Add order to the customer's Orders collection
cust.Orders.Add(ord);

db.SubmitChanges();
```

Modifying & Saving Entities

```
// Set the value to newly inserted OrderID
int retrievedID = ...;

// Retrieve single order with given ID
Order ord = (from o in db.Orders
             where o.OrderID == retrievedID
             select o).Single();

// Mark the order for deletion
db.Orders.DeleteOnSubmit(ord);

db.SubmitChanges();
```

Modifying & Saving Entities

```
var cust = (from c in db.Customers
             where c.CustomerID == "AROUT"
             select c).Single();

// Change a property of retrieved customer
cust.ContactName = "Horatio Hornblower";

// Persist the change
db.SubmitChanges();
```

Manage relationship

- Có thể thay đổi khoá ngoại bằng cách
 - Add/Remove entity ra khỏi *entitySet*
 - Thay đổi *entityRef*

```
Customer cust1 = db.Customers.Single(c => c.CustomerID == id1);
Customer cust2 = db.Customers.Single(c => c.CustomerID == id2);


// Pick an order
Order o = cust1.Orders[0];

// Remove from first, add to the second
cust1.Orders.Remove(o);
cust2.Orders.Add(o);
```

Manage relationship

```
Customer cust1 = db.Customers.Single(c => c.CustomerID == id1);

// Pick an order
Order o = cust1.Orders[0];

// Set reference to null. db.DeleteOnSubmit() not called
o.Customer = null; 

// Prints 'true'
Console.WriteLine(cust1.Orders.Contains(o));

// Updates Order
db.SubmitChanges();
```


Submitting changes

- Mỗi khi gọi **submitChanges**, toàn bộ thay đổi sẽ được lưu xuống CSDL
- Sau khi lưu thành công, toàn bộ thay đổi sẽ bị "bỏ quên", **dataContext** lúc này không còn chứa bất kỳ thông tin nào về những thay đổi nữa.
- Không có **rollback** khi lưu thất bại → developer phải tự mình sửa lỗi & **submitChanges** lại

Transaction

```
using System.Transactions; // Add reference to System.Transactions.dll

using (TransactionScope ts = new TransactionScope())
{
    NorthwindDataContext db = new NorthwindDataContext();
    List<Order> orders =
        db.Orders.Where(o => o.Customer.CustomerID == "ALFKI").ToList();
    // Modify order entities here
    db.SubmitChanges();
    ts.Complete();
}
```


Transaction

```
using System.Data.SqlClient;

DbTransaction myTxn = db.Connection.BeginTransaction();
// Execute some SqlCommands with the transaction
db.Transaction = myTxn;
try
{
    db.SubmitChanges();
}
catch(ChangeConflictException e)
{
    // handle conflicts and roll back transaction
}
// Do additional work with the transaction
myTxn.Commit();
```

Attaching Multitier Entities

- Ứng dụng có thể được chia làm nhiều Tiers
- Hành động ĐỌC & GHI thường không được dùng chung 1 đối tượng `dataContext`

Attaching Multitier Entities

- Cần attach đối tượng được thay đổi ở tier khác vào context mới

```
public bool Update(tt_customer customer)
{
    context = new TimeTrakkerContext();

    tt_customer Tcust = context.tt_customers.Single(c => c.Pk == customer.Pk);
    context.tt_customers.Attach(customer, Tcust);
    context.SubmitChanges();

    return true;
}
```

Attaching Multitier Entities

```
NorthwindDataContext db1 = new NorthwindDataContext();
Customer c1 = db1.Customers.Single(c => c.CustomerID == "AROUT");

// Customer entity changed on another tier - e.g. through a browser
// Back on the mid-tier, a new context needs to be used
NorthwindDataContext db2 = new NorthwindDataContext();
// Create a new entity for applying changes
Customer c2 = new Customer();
c2.CustomerID = originalID;

// Set other properties needed for optimistic concurrency check
c2.CompanyName = originalCompanyName;
...
// Tell DataContext to track this object for an update
db2.Customers.Attach(c2);
// Now apply the changes
c2.ContactName = "Horatio Hornblower";
// DataContext has original/current values to update the customer
db2.SubmitChanges();
```

Using Store Proc in DLINQ

- Dùng **sqlmetal /sprocs** để generate hàm ánh xạ từ CSDL sang hàm trên C#
- Dùng VS2008 designer



ISingleResult

```
CREATE PROCEDURE OrdersByCustomer    @CustomerID nchar(5)
AS
SELECT *
FROM Orders
WHERE CustomerID = @CustomerID
```

```
NorthwindDataContext db = new NorthwindDataContext();
db.Log = Console.Out;

ISingleResult<Order> OrdersQuery = db.OrdersByCustomer("BOLID");
```


IMultipleResults

```
CREATE PROCEDURE SuppliersAndCustomers @City nvarchar(15)
AS

SELECT *
FROM Suppliers
WHERE City = @City

SELECT *
FROM Customers
WHERE City = @City
```

IMultipleResults

```
public partial class NorthwindDataContext
{
    [Function(Name = "dbo.SuppliersAndCustomers")]
    [ResultType(typeof(Supplier))]
    [ResultType(typeof(Customer))]
    public IMultipleResults SuppliersAndCustomers(
        [Parameter(Name = "City", DbType = "NVarChar(15)")] string city)
    {
        IExecuteResult result = this.ExecuteMethodCall(this,
            ((MethodInfo)(MethodInfo.GetCurrentMethod())), city);
        return ((IMultipleResults)(result.ReturnValue));
    }
}
```

```
using(IMultipleResults results = db.SuppliersAndCustomers("London"))
{
    List<Supplier> suppliers = results.GetResult<Supplier>().ToList();
    List<Customer> customers = results.GetResult<Customer>().ToList();
}
```



```
CREATE FUNCTION OrdersByShipper(@shipper integer)
RETURNS TABLE
AS
RETURN (SELECT *
        FROM Orders ord
        WHERE ord.ShipVia = @shipper)
```

```
[Function(Name="dbo.OrdersByShipper", IsComposable=true)]
public IQueryable<Order> OrdersByShipper([Parameter(DbType="Int")]
    System.Nullable<int> shipper)
{
    return this.CreateMethodCallQuery<Order>(this,
        ((MethodInfo)(MethodInfo.GetCurrentMethod()))), shipper);
}
```

```
IQueryable<Order> orders = from o in db.OrdersByShipper(1)
                           where o.Customer.City == "London"
                           select o;
```

Store Proc for CUD

- Tạo các proc cho phép Insert/ Delete/ Update
- Cấu hình các table trong file dbml để LINQ dùng các proc khi **submitChanges** thay cho việc tự generate các lệnh Insert/ Delete/ Update





Thank You!
Questions & Answers