# LINQ

# (Language INTegrated Query)

Lionel Seinturier

Université des Sciences et Technologies de Lille

Lionel.Seinturier@univ-lille1.fr

31/10/08





## LINQ (Language INTegrated Query)

- proposer un langage de requêtage SQL *like* intégré au langage de programmation (C#, VB, …)
- unifier les pratiques
- éviter d'imposer aux développeurs la manipulation de deux syntaxes ≠
- éviter d'avoir à transmettre des requêtes sous la forme de `string` (voir ADO.NET)

- *framework* unifié d'accès aux données
  - objets (collections et tableaux)
  - tables SQL
  - fichiers XML





## LINQ for Objects

- nouveaux mots-clés
  - from : variable d'itération
  - in : collection sur laquelle s'effectue l'itération
  - where : condition de sélection de l'élément courant
  - select : valeur à sélectionner

Exemple : interroger un tableau

```
int[] tab = new int[]{ 1, 2, 3, 4, 5, 6 };
IEnumerable<int> pairs =
  from number in tab
  where number % 2 == 0
  select number;

foreach(int i in pairs) { Console.WriteLine(i); }
```





## LINQ for Objects

Exemple : interroger une collection d'objets

```
var people = new List<User>() {
  new User { Name = "Bob", Age = 24 },
  new User { Name = "Anne", Age = 26 } };

var persons =
  from p in people
  where p.Age > 25 && p.Name.Length >= 2
  select p;

foreach(var person in persons) {
  Console.WriteLine( person.Name + " "+ person.Age);
}
```

## LINQ for Objects

Jointures

```
var addresses = new List<Address>() {
  new User { Name = "Bob", Ville = "Lille" },
  new User { Name = "Pat", Ville = "Paris" } };

var persons =
  from p in people
  join a in addresses on p.Name equals a.Name
  select p;

foreach(var person in persons) {
  Console.WriteLine( person.Name + " "+ person.Age);
}
```





## LINQ for Objects

Opérateurs sur les collections et les tableaux

- OfType<T>
- Min, Max, Sum, Average

```
object[] values = { 1, "Tom", 'T', 12.5, 3, true, 20 };
var results = values.OfType<int>();
foreach(int i in results) { Console.WriteLine(i); }

int[] IntegerValues = { 0, 2, 5, 6, 7 };
int max = IntegerValues.Max();
int min = IntegerValues.Min();
int sum = IntegerValues.Sum();
double average = IntegerValues.Average();
```





## LINQ for Objects

Opérateurs sur les collections et les tableaux

- Where(λ)
- Count(λ)
- ToArray, ToList

```
var values = new object[] { 1, "Tom", 'T', 12.5, 3, true, 20 };
var val = values.Where(v => v.ToString().Length >= 3);
foreach(object o in val) { Console.WriteLine(o.ToString()); }

int i = values.Count(v => v.ToString().Length >= 3);

object[] array = values.ToArray();
List<object> list = array.ToList();
```





## LINQ for SQL

- *mapping* O/R (objet/relationnel)
- manipuler les données stockées dans des tables SQL "presque" comme des collections
- annotations pour faire le lien entre
  - les propriétés des classes
  - les colonnes des tables SQL

⇨ similitude avec EJB3

3 annotations principales

- pour classe : `[Table(Name="...")]` (le nom de la table)
- pour propriété : `[Column(Name="...", IsPrimaryKey=bool)]`
- pour propriété : `[Association(ThisKey="...")]`

- définies dans namespace `System.Data.Linq.Mapping`
  - note pour VS 2008 : ajouter `System.Data.Linq` dans les références du projet

## LINQ for SQL

```
using System;
using System.Data.Linq.Mapping;

[Table(Name="Comptes")]
public class Compte {

    [Column(IsPrimaryKey=true)]
    public string Nom { get { return nom; } set { nom=value; } }

    [Column]
    public double Solde { get { return solde; } set { solde = value; } }

    private string nom;
    private double solde;
}
```

- seules les propriétés annotées sont mises en correspondance





## LINQ for SQL

Récupération des valeurs contenues dans la table

```
String dbname = "D:\\...\\Database.mdf";
DataContext db = new DataContext(@dbname);

Table<Compte> comptes = db.GetTable<Compte>();

var all = from c in comptes select c;

foreach (Compte c in all) {
  Console.WriteLine(c.Nom + " " + c.Solde);
}
```

- 3 possibilités pour `new DataContext`
  - le fichier `.mdf` de la base de données
  - une chaîne de connexion (*connection string*)
  - un objet connexion





## LINQ for SQL

Insertion d'un élément dans la table

```
Compte nco = new Compte { Nom = "Chelsea", Solde = 27 };
comptes.InsertOnSubmit(nco);
db.SubmitChanges();
```

Modification d'un élément de la table

```
c.Solde = 24;
db.SubmitChanges();
```

Suppression d'un élément de la table

```
comtpes.Remove(c);
db.SubmitChanges();
```

- globalisation de +sieurs modifications
- appel à `SubmitChanges` pour les prendre en compte





## LINQ for XML

- créer des requêtes sur une structure de données représentant un arbre XML
- ajouter, supprimer, modifier cette structure
- répercuter les modifications sur le fichier XML

Création de l'arbre

- par analyse syntaxique d'une chaîne
- par chargement d'un fichier
- par utilisation des classes XElement, XAttribute

## LINQ for XML

Création d'un arbre XML par analyse syntaxique d'une chaîne

```
XElement contacts = XElement.Parse(
  @"<contacts>
      <contact>
        <name>Patrick Hines</name>
        <phone type="home">206-555-0144</phone>
        <phone type="work">425-555-0145</phone>
        <address>
          <street1>123 Main St</street1>
          <city>Mercer Island</city>
          <state>WA</state>
          <postal>68042</postal>
        </address>
        <netWorth>10</netWorth>
      </contact>
    </contacts>" );
```

Création d'un arbre XML par chargement d'un fichier

```
XElement contactsFromFile = XElement.Load(@"c:\myContactList.xml");
```





## LINQ for XML

Création d'un arbre XML par utilisation des classes XElement, XAttribute

```
XElement contacts =
  new XElement("contacts",
    new XElement("contact",
      new XElement("name", "Patrick Hines"),
      new XElement("phone", "206-555-0144"),
      new XElement("address",
        new XElement("street1", "123 Main St"),
        new XElement("city", "Mercer Island") ) ) );
```

Requêtage

```
foreach(object o in contacts.Nodes()) { Console.WriteLine(o); }

var result =
  from c in contacts.Elements("contact")
  select c.Element("name");
foreach(XElement name in result) { Console.WriteLine(name.Value); }
```





## LINQ for XML

Ajouter, supprimer, modifier des éléments

```
XElement mobilephone = new XElement("mobilephone", "206-333-4546");
contacts.Add(mobilephone);
contacts.Element("mobilephone").Remove();
contacts.Element("mobilephone").ReplaceContent("0123456789");
contacts.SetElement("mobilephone", "0123456789");
```

Répercution des modifications

```
contacts2.Save("C:\Contacts.xml");
```





## LINQ for XML

Accéder aux attributs d'un élément

• propriété Attribute d'un objet de type XElement

```
XElement contacts =
  new XElement("contacts",
    new XElement("contact",
      new XElement("name", "Patrick Hines"),
      new XElement("phone",
        new XAttribute("type", "home"), "206-555-0144"),
      new XElement("phone",
        new XAttribute("type", "pro"), "206-333-4567"),
      new XElement("address",
        new XElement("street1", "123 Main St"),
        new XElement("city", "Mercer Island") ) ) );
```

## LINQ for XML

### Accéder aux attributs d'un élément

```
foreach(var p in contacts.Elements("phone")) {
  if((string)p.Attribute("type") == "home") {
    Console.WriteLine("Tel perso: " + (string)p);
} }
```

### Modifier, supprimer un attribut

```
contacts.Element("phone").SetAttribute("type","home");
contacts.Element("phone").Attribute("type").Remove();
```