



## Module 14

# Utiliser LINQ Requêter des Données

# Sommaire

- Utiliser les Méthodes d'Extensions LINQ et Opérateurs de Requête
- Créer des Requêtes et des Expressions LINQ Dynamiquement

# Leçon 1: Utiliser les Méthodes d'Extensions LINQ et Opérateurs de Requête

- Quel est l'objectif de LINQ?
- Requête des Données et Construire un Jeu de Résultats
- Filtrer des Données
- Ordonnées les Données
- Grouper les Données et Réaliser des Calculs d'Agrégation
- Joindre des Données Issues de Différents Jeux de Données
- Utiliser les Opérateurs de Requête LINQ
- Évaluation Différée ou pas des Requêtes

# Quel est l'objectif de LINQ?

## Architecture LINQ

**C#**

**Autres Langages .NET**

**LINQ**

### LINQ-enabled data source

LINQ-enabled ADO.NET

LINQ to  
Objects

LINQ to  
DataSets

LINQ to  
SQL

LINQ to  
Entities

LINQ to  
XML

# Requêter des Données et Construire un Jeu de Résultats

La Possibilité de sélectionner des lignes de données est une exigence fondamentale des applications modernes

```
IEnumerable<Customer> customers = new[]  
{  
    new Customer{ FirstName = "...", LastName="...", Age = 41},  
    ...  
};
```

```
List<string> customerLastNames = new List<string>();  
foreach (Customer customer in customers)  
{  
    customerLastNames.Add(customer.LastName);  
}
```

L'approche traditionnelle qui utilise **foreach**

...

```
IEnumerable<string> customerLastNames =  
    customers.Select(cust => cust.LastName);
```

L'approche LINQ approach utilise la méthode d'extension **Select**

La méthode d'extension **Select** est disponible dans toute classe qui implémente les interfaces **IQueryable<T>** ou **IEnumerable<T>**

# Filtrer des Données

Utiliser la méthode d'extension **Where** pour restreindre les éléments retournés

```
var customerLastNames =  
    customers.Where(cust => cust.Age > 25).  
    Select(cust => cust.LastName);
```

Obtenir les noms de famille des clients de plus de 25 ans

Utilisez la méthode d'extension **Where** pour filtrer les données, et utilisez la méthode d'extension **Select** pour renvoyer les données



**Important:** Utilisez les méthodes dans le bon ordre, ordre sinon vous pourriez obtenir des résultats erronés ou des erreurs de compilation

# Ordonnées les Données

Ordonnez les données avec les méthodes d'extension **OrderBy**, **OrderByDescending**, **ThenBy**, and **ThenByDescending**

Utilisez les méthodes d'extension **OrderBy** et **OrderByDescending** pour réaliser un tri simple

```
var sortedCustomers =  
    customers.OrderBy(cust => cust.FirstName);
```

Ordonner les clients  
par leur prénom

Utiliser les méthodes d'extension **ThenBy** et **ThenByDescending** pour trier sur plusieurs clés

```
var sortedCustomers =  
    customers.OrderBy(cust => cust.FirstName).  
    ThenBy(cust => cust.Age);
```

Ordonner les clients  
par leur prénom et  
leur âge

# Grouper les Données et Réaliser des Calculs d'Agrégation

Calculer un résultat agrégé à partir d'une collection dénombrable

```
Console.WriteLine(  
    "Count:{0}\t\tAverage age:{1}\t\tLowest:{2}\t\tHighest:{3}",  
    customers.Count(),  
    customers.Average(cust => cust.Age),  
    customers.Min(cust => cust.Age),  
    customers.Max(cust => cust.Age));
```

Grouper les données avec la méthode d'extensions **GroupBy**

```
var customersGroupedByAgeRange = customers.GroupBy( ... );
```

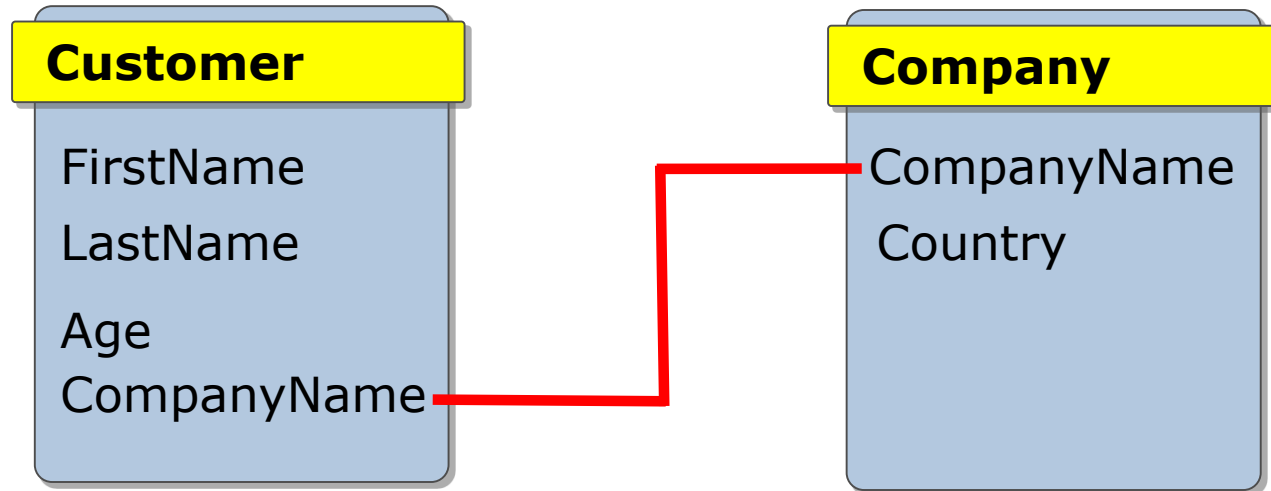
Éliminer les valeurs dupliquées

```
Console.WriteLine("{0}",  
    customers.Select(cust => cust.Age).Distinct().Count());
```



# Joindre des Données Issues de Différents Jeux de Données

LINQ fournit la méthode d'extension Join pour joindre les données stockées dans des sources différentes et ainsi exécuter des requêtes composites



```
var customersAndCompanies = customers.Join(  
    companies,  
    custs => custs.CompanyName,  
    comps => comps.CompanyName,  
    (custs, comps) =>  
        new { custs.FirstName, custs.LastName, comps.Country});
```

# Utiliser les Opérateurs de Requête LINQ

les Opérateurs de Requête LINQ fournissent une syntaxe abrégée qui ressemble aux clauses SQL

```
IEnumerable<string> customerLastNames =  
    customers.Select(cust => cust.LastName);
```

*Approche par les  
méthodes d'extensions*

```
IEnumerable<string> customerLastNames =  
    from cust in customers  
    select cust.LastName;
```

*Approche par les  
opérateurs de requête*

Il y a un opérateur de requête équivalent pour chaque méthode d'extensions LINQ

```
var customersOver25 = from cust in customers  
    where cust.Age > 25 select cust;  
...  
var sortedCustomers = from cust in customers  
    orderby cust.FirstName select cust;  
...  
var customersGroupedByAge = from cust in customers  
    group cust by cust.Age;
```

# Évaluation Différée ou pas des Requêtes

Les requêtes LINQ ne sont pas exécutées au niveau de leur déclaration. Elles sont évaluées au moment de leur appel.

Cette stratégie est nommée "évaluation différée".

```
var usCompanies = from a in companies
                   where String.Equals(a.Country, "United States")
                   select a.CompanyName;
```

*Données pas encore  
retrouvées*

```
foreach (string name in usCompanies)
{
    Console.WriteLine(name);
}
```

*Données retrouvées*

Vous pouvez forcer l'évaluation de vos requêtes en utilisant les méthodes **ToList** et **ToArray** dans la déclaration de la requête

Une évaluation précoce créer une copie statique des données

## Lesson 2: Créer des Requêtes et des Expressions LINQ Dynamiques

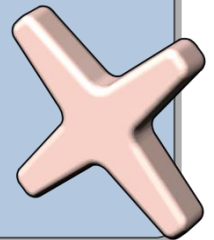
- Qu'est-ce qu'une Requête LINQ Dynamique?
- Qu'est-ce qu'un arbre d'Expression?
- Les Types d'Expressions
- Obtenir des Informations sur le Type au Run Time
- Compiler et executer une requête LINQ Dynamic

# Créer des Requêtes et des Expressions LINQ Dynamiquement

Les requêtes statiques sont très performantes, mais nécessitent de connaître la requête à la compilation

Requêtes Statiques:

- Vous force toujours à écrire toutes les requêtes possibles
- Besoin parfois de beaucoup de code inutile qu'on ne peut jamais exécuter
- Cachent parfois des bogues à l'aide des instructions conditionnelles complexes



Requêtes dynamiques sont encore plus puissants que les requêtes statiques, car elles vous permettent de construire une requête en cours d'exécution

Les Requêtes Dynamiques vous permettent de:

- Construire une requête unique basée sur les paramètres fournis
- Produire une requête selon les exigences de l'utilisateur
- Ecrire uniquement le code nécessaire
- Réduire la quantité de bogues potentiels



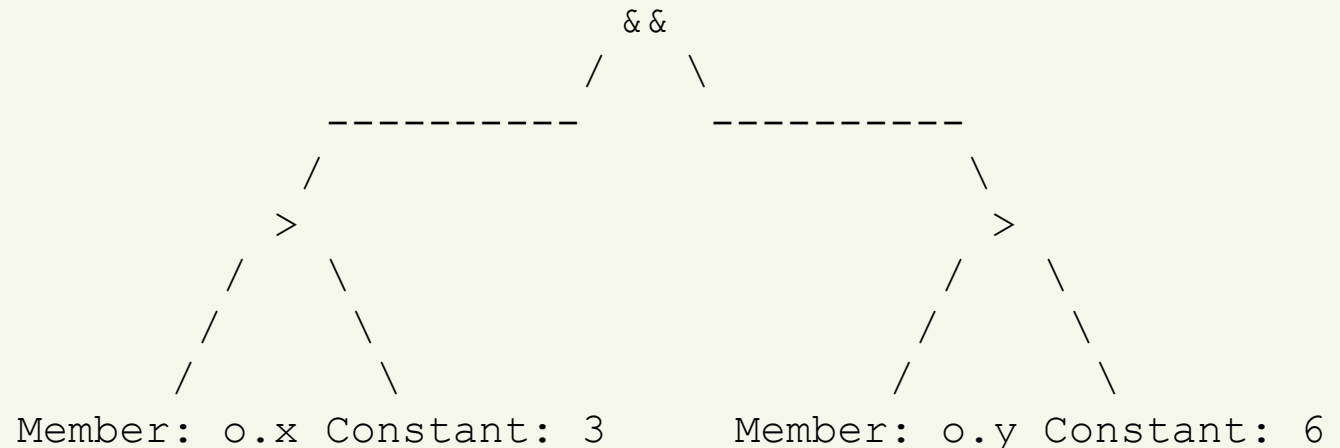
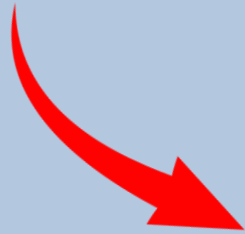
# Qu'est-ce qu'un arbre d'Expression?

Les arbres d'expression contiennent généralement plusieurs expressions, qui, une fois évaluée forme une requête dynamique

Les arbres d'expression fournissent une syntaxe flexible pour développer des expressions lambda

Lorsque vous créez un arbre d'expression, chaque expression doit être divisée en éléments individuels

`x > 3 && y > 6`



# Les Types d'Expressions

## La classe **Expression**

Fournit des méthodes statiques pour créer d'autres objets expression

## La classe **BinaryExpression**

Représente une expression avec un opérateur binaire

## La classe **ConstantExpression**

Représente une référence constante dans une expression

## La classe **MemberExpression**

Représente une référence aux membres, telle qu'une propriété dans un type

## La classe **UnaryExpression**

Représente des opérations unaires dans les expressions

## La classe **Expression<TDelegate>**

Représente une expression lambda



# Obtenir des Informations sur le Type au Run Time

Lorsque vous développez un arbre d'expression, vous ne pouvez pas utiliser les propriétés des objets c# directement


Vous devez utiliser la réflexion pour accès objet types et membres

- Utilisez la classe **Type** pour représenter le type d'un objet
- Utilisez la classe **MemberInfo** pour représenter un membre

```
Type stringType = typeof(string);
```

```
MemberInfo stringLength = stringType.GetProperty("Length");
```

Représente le  
type **string**



Représente le  
membre **Length**





# Compiler et exécuter une requête LINQ Dynamic

Vous pouvez utiliser la méthode d'instance **Compile** qu'expose la classe **Expression<TDelegate>** pour compiler l'arbre d'expression lambda expression dans une expression lambda

```
Expression<Func<int, bool>> expressionTree =  
    Expression.Lambda<Func<int, bool>>(expression, parameter);  
  
Func<int, bool> myDelegate = null;  
myDelegate += expressionTree.Compile();
```

Vous pouvez utiliser la méthode **Compile** comme argument dans un méthode d'extension LINQ

```
var bankAccounts = accounts.Where(expressionTree.Compile());
```

Vous pouvez utiliser la méthode **DynamicInvoke** de l'objet **TDelegate** que la méthode **Compile** renvoie pour appeler l'expression lambda

```
bool response = expressionTree.Compile().DynamicInvoke(1);
```

# Atelier Pratique

- Exercice 1:
- Exercice 2: