# Opérations de requête LINQ de base (C#)

**Visual Studio 2012**

Cette rubrique présente brièvement les expressions de requête LINQ et quelques-uns des types d'opérations classiques que vous effectuez dans une requête. Vous trouverez des informations plus détaillées dans les rubriques suivantes :

**Remarque**

Si vous êtes déjà familiarisé avec un langage de requête comme SQL ou XQuery, vous pouvez ignorer la majorité de cette rubrique. Consultez le paragraphe « Clause from » de la section suivante pour en savoir plus sur l'ordre des clauses dans les expressions de requête LINQ.

## Obtention d'une source de données

Dans une requête LINQ, la première étape consiste à spécifier la source de données. En C# comme dans la plupart des langages de programmation, une variable doit être déclarée avant de pouvoir être utilisée. Dans une requête LINQ, la clause from apparaît en premier pour introduire la source de données (customers) et la variable de portée (cust).

C#

```
//queryAllCustomers is an IEnumerable<Customer>
var queryAllCustomers = from cust in customers
                        select cust;
```

La variable de portée est similaire à la variable d'itération dans une boucle foreach, à la différence qu'aucune itération réelle ne se produit dans une expression de requête. Lorsque la requête est exécutée, la variable de portée servira de référence à chaque élément consécutif dans customers. Étant donné que le compilateur peut déduire le type de cust, vous n'avez pas à le spécifier explicitement. Des variables de portée supplémentaires peuvent être introduites par une clause let. Pour plus d'informations, consultez let, clause (Référence C#).

**Remarque**

Pour les sources de données non génériques telles que ArrayList, la variable de portée doit être explicitement typée. Pour plus d'informations, consultez Comment : interroger un ArrayList avec LINQ et from, clause (Référence C#).

## Filtrage

L'opération de requête la plus courante est probablement l'application d'un filtre sous forme d'expression booléenne. Du fait du filtre, la requête retourne uniquement les éléments pour lesquels l'expression a la valeur true. Le résultat est produit à l'aide de la clause where. Le filtre activé spécifie les éléments à exclure de la séquence source. Dans l'exemple suivant, seuls les customers qui ont une adresse à Londres sont retournés.

C#

```
var queryLondonCustomers = from cust in customers
                           where cust.City == "London"
                           select cust;
```

Vous pouvez utiliser les opérateurs logiques C# AND et OR habituels pour appliquer autant d'expressions de filtre que nécessaire dans la clause where. Par exemple, pour retourner uniquement les clients de « Londres » ET AND dont le nom est « Devon », vous écrirez le code suivant :

C#

```
where cust.City=="London" && cust.Name == "Devon"
```

Pour retourner les clients de Londres ou Paris, vous écrirez le code suivant :

C#

```
where cust.City == "London" || cust.City == "Paris"
```

Pour plus d'informations, consultez where, clause (Référence C#).

## Ordering

Souvent, il est utile de trier les données retournées. La clause orderby triera les éléments de la séquence retournée d'après le comparateur par défaut pour le type qui est trié. Par exemple, la requête suivante peut être étendue pour trier les résultats selon la propriété Name. Étant donné que Name est une chaîne, le comparateur par défaut effectue un tri alphabétique de A à Z.

C#

```
var queryLondonCustomers3 =
    from cust in customers
    where cust.City == "London"
    orderby cust.Name ascending
    select cust;
```

Pour classer les résultats dans l'ordre inverse, de Z à A, utilisez la clause orderby…descending.

Pour plus d'informations, consultez orderby, clause (Référence C#).

## Regroupement

La clause group vous permet de grouper vos résultats selon une clé que vous spécifiez. Par exemple, vous pouvez spécifier un regroupement des résultats par City de sorte que tous les clients de Londres ou Paris soient dans des groupes individuels. Dans ce cas, cust.City est la clé.

C#

```
// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>
  var queryCustomersByCity =
      from cust in customers
      group cust by cust.City;

  // customerGroup is an IGrouping<string, Customer>
  foreach (var customerGroup in queryCustomersByCity)
  {
      Console.WriteLine(customerGroup.Key);
      foreach (Customer customer in customerGroup)
      {
          Console.WriteLine("    {0}", customer.Name);
      }
  }
```

Lorsque vous terminez une requête avec une clause group, vos résultats prennent la forme d'une liste de listes. Chaque élément de la liste est un objet qui a un membre Key et une liste d'éléments groupés sous cette clé. Lorsque vous itérez sur une requête qui produit une séquence de groupes, vous devez utiliser une boucle foreach imbriquée. La boucle externe itère sur chaque groupe et la boucle interne itère au sein des membres de chaque groupe.

Si vous devez faire référence aux résultats d'une opération de groupe, vous pouvez utiliser le mot clé into pour créer un identificateur qui peut être interrogé ultérieurement. La requête suivante retourne uniquement les groupes qui contiennent plus de deux clients :

C#

```
// custQuery is an IEnumerable<IGrouping<string, Customer>>
var custQuery =
    from cust in customers
    group cust by cust.City into custGroup
    where custGroup.Count() > 2
    orderby custGroup.Key
    select custGroup;
```

Pour plus d'informations, consultez group, clause (Référence C#).

## Jointure

Les opérations de jointure créent des associations entre les séquences qui ne sont pas modelées explicitement dans les sources de données. Par exemple, vous pouvez effectuer une jointure pour rechercher tous les clients et distributeurs qui ont le même emplacement. Dans LINQ, la clause join fonctionne toujours par rapport à des collections d'objets plutôt que directement par rapport à des tables de base de données.

C#

```
var innerJoinQuery =
    from cust in customers
    join dist in distributors on cust.City equals dist.City
    select new { CustomerName = cust.Name, DistributorName = dist.Name };
```

Dans LINQ, il n'est pas nécessaire d'utiliser join aussi souvent que vous le faites dans SQL, car les clés étrangères dans LINQ sont représentées dans le modèle objet comme des propriétés qui stockent une collection d'éléments. Par exemple, un objet Customer contient une collection d'objets Order. Au lieu d'effectuer une jointure, vous accédez aux commandes en utilisant la notation par point :

```
from order in Customer.Orders...
```

Pour plus d'informations, consultez join, clause (Référence C#).

## Sélection (projections)

La clause select produit les résultats de la requête et spécifie la « forme » ou le type de chaque élément retourné. Par exemple, vous pouvez spécifier si vos résultats se composeront d'objets Customer complets, d'un seul membre, d'un sous-ensemble de membres ou d'un type de résultat complètement différent basé sur un calcul ou une création d'objet. Lorsque la clause select produit autre chose qu'une copie de l'élément source, l'opération est appelée projection. L'utilisation de projections pour transformer des données est une fonction puissante des expressions de requête LINQ. Pour plus d'informations, consultez Transformations de données avec LINQ (C#) et select, clause (Référence C#).