Welcome to LINQ

From Begin to Advanced

# **Bắt đầu với LINQ**

LINQ – Truy vấn tích hợp ngôn ngữ , là ngôn ngữ truy vấn mạnh mẽ được giới thiệu với .NET 3.0 và Visual Studio 2008. LINQ được tích hợp trong C# và VB để truy vấn nhiều nguồn dữ liệu khác nhau.

Tài liệu này sẽ giúp bạn học ngôn ngữ LINQ bằng các chủ đề từ cơ bản đến nâng cao. Các hướng dẫn này được chia thành một loạt các chủ đề liên quan, để bạn bắt đầu từ một chủ đề phải được hiểu trước, sau đó dần dần tìm hiểu các tính năng khác của LINQ một cách tuần tự. Hướng dẫn LINQ được đóng gói với các giải thích dễ hiểu, các ví dụ thực tế, các mẹo hữu ích, ghi chú thông tin và các điểm cần nhớ.

Các hướng dẫn này được thiết kế cho người mới bắt đầu và các chuyên gia muốn tìm hiểu LINQ từng bước.

Tiên quyết: Kiến thức cơ bản về .Net Framework 3.5 / 4.5, C #, Visual Studio là bắt buộc.

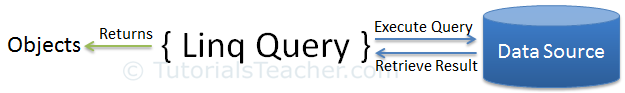
**LINQ là gì?**

LINQ (Truy vấn tích hợp ngôn ngữ) là cú pháp truy vấn thống nhất trong C # và VB.NET để truy xuất dữ liệu từ các nguồn và định dạng khác nhau. Nó được tích hợp trong C # hoặc VB, do đó loại bỏ sự không phù hợp giữa ngôn ngữ lập trình và cơ sở dữ liệu, cũng như cung cấp một giao diện truy vấn duy nhất cho các loại nguồn dữ liệu khác nhau.

Ví dụ: SQL là Ngôn ngữ truy vấn có cấu trúc được sử dụng để lưu và truy xuất dữ liệu từ cơ sở dữ liệu. Theo cùng một cách, LINQ là một cú pháp truy vấn có cấu trúc được xây dựng trong C # và VB.NET để lấy dữ liệu từ các loại nguồn dữ liệu khác nhau, chẳng hạn như: collections, bộ dữ liệu DO.Net, tài liệu XML, dịch vụ web và Máy chủ MS SQL và các cơ sở dữ liệu khác.



Truy vấn LINQ trả về kết quả dưới dạng đối tượng. Nó cho phép bạn sử dụng cách tiếp cận hướng đối tượng trên tập kết quả và không phải lo lắng về việc chuyển đổi các định dạng kết quả khác nhau thành các đối tượng.



Ví dụ sau đây cho thấy một truy vấn LINQ đơn giản nhận được tất cả các chuỗi từ một mảng có chứa 'a'.

// Data source

string[] names = {"Bill", "Steve", "James", "Mohan" };

// LINQ Query

var myLinqQuery = from name in names

where name.Contains('a')

select name;

// Query execution

foreach(var name in myLinqQuery)

Console.Write(name + " ");

Bạn sẽ không nhận được kết quả của truy vấn LINQ cho đến khi bạn thực hiện nó. Ở đây chúng tôi đã sử dụng vòng lặp foreach để thực hiện truy vấn của mình được lưu trữ trong myLinqQuery. Vòng lặp foreach thực hiện truy vấn trên nguồn dữ liệu và nhận kết quả và sau đó lặp lại qua tập kết quả.

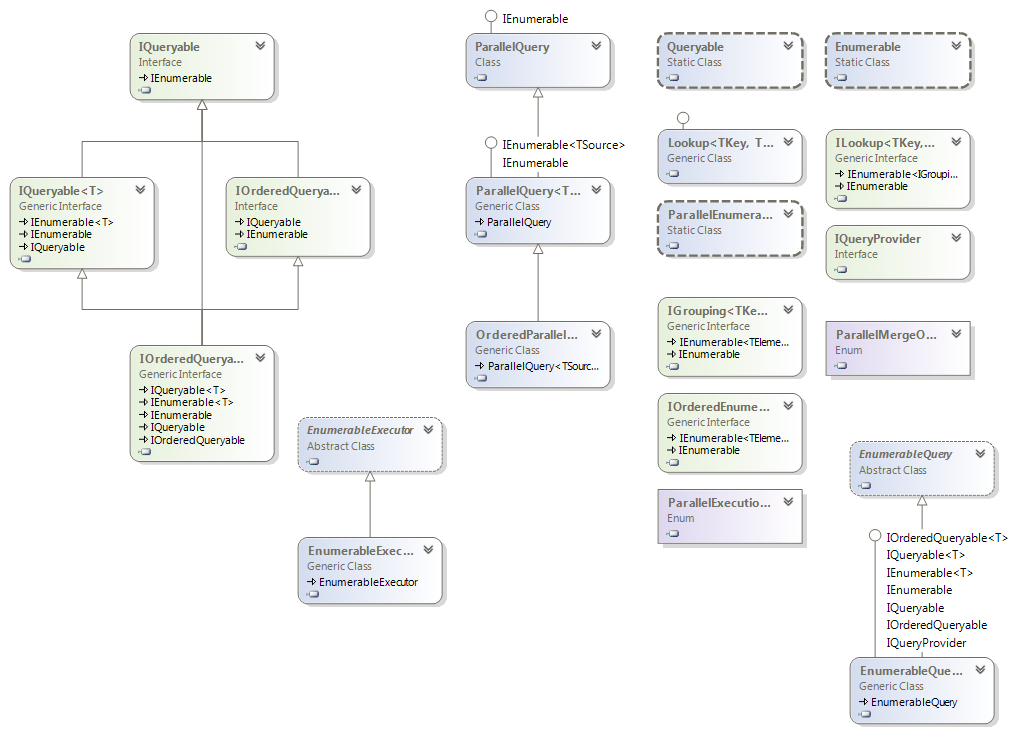
Do đó, mọi truy vấn LINQ phải truy vấn một số loại nguồn dữ liệu cho dù đó có thể là mảng, bộ sưu tập, XML hoặc cơ sở dữ liệu khác. Sau khi viết truy vấn LINQ, nó phải được thực thi để có kết quả.

**Tại sao sử dụng LINQ?**

* Ngôn ngữ quen thuộc: Nhà phát triển không cần phải học một ngôn ngữ truy vấn mới cho từng loại nguồn dữ liệu hoặc định dạng dữ liệu.
* Giảm code: Nó làm giảm số lượng mã được viết so với cách tiếp cận truyền thống hơn.
* Mã có thể đọc được: LINQ làm cho mã dễ đọc hơn để các nhà phát triển khác có thể dễ dàng hiểu và duy trì nó.
* Cách truy vấn chuẩn hóa với nhiều nguồn dữ liệu: Có thể sử dụng cùng một cú pháp LINQ để truy vấn nhiều nguồn dữ liệu.
* An toàn tại thời điểm biên dịch của các truy vấn: LINQ cung cấp kiểm tra kiểu của các đối tượng tại thời gian biên dịch.
* Hỗ trợ IntelliSense: LINQ cung cấp IntelliSense cho các collection chung. IntelliSense là một thuật ngữ chung cho nhiều tính năng chỉnh sửa mã bao gồm: hoàn thành mã, thông tin tham số, thông tin nhanh và danh sách thành viên. Các tính năng của IntelliSense đôi khi được gọi bằng các tên khác như "code completion", "content assist" và "code hinting".
* Định hình dữ liệu: Bạn có thể truy xuất dữ liệu theo các khuôn khác nhau.

**LINQ API?**

Chúng ta có thể viết các truy vấn LINQ cho các lớp thể hiện interface <T> hoặc IQueryable <T> của IEnumerable. Không gian tên System.Linq bao gồm các lớp và interface sau đây yêu cầu cho các truy vấn LINQ.

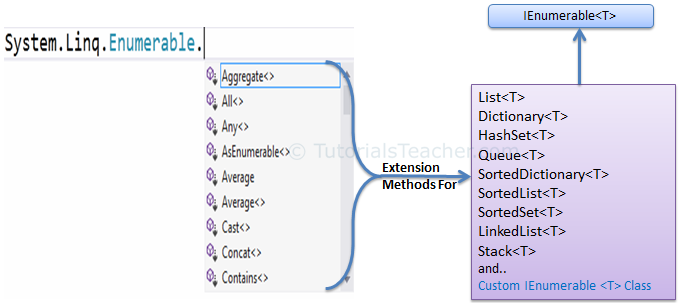


Các truy vấn LINQ sử dụng các phương thức mở rộng cho các lớp thực hiện giao diện IEnumerable hoặc IQueryable. Enumerable và Queryable là hai lớp tĩnh chứa các phương thức mở rộng để viết các truy vấn LINQ.

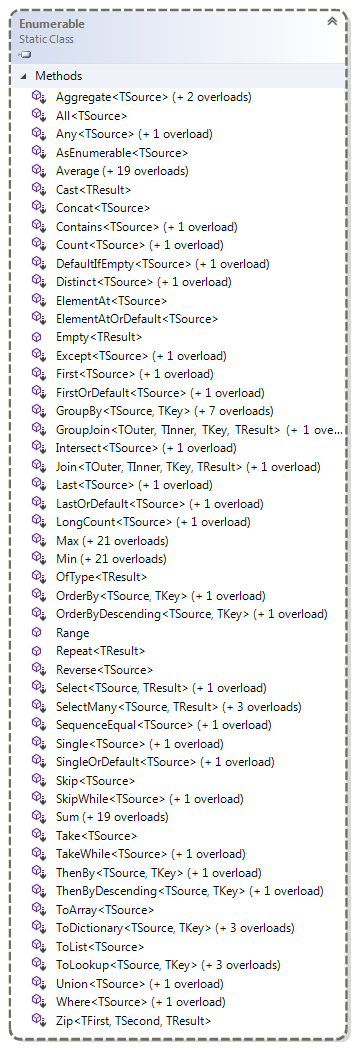
IEnumerable

IEnumerable class bao gồm các phương thức mở rộng cho các lớp hiện thực IEnumerable<T> interface. Ví dụ tất cả các lớp collection dựng sẵn hiện thực IEnumerable<T> interface và vì vậy chúng ta có thể viết các truy vấn LINQ để lấy dữ liệu từ các bộ sưu tập tích hợp.

Hình dưới đây cho thấy các phương thức mở rộng được bao gồm trong lớp Enumerable có thể được sử dụng với generic collections trong C # hoặc VB.Net.



Hình dưới đây cho thấy tất cả các phương thức mở rộng có sẵn trong lớp Enumerable

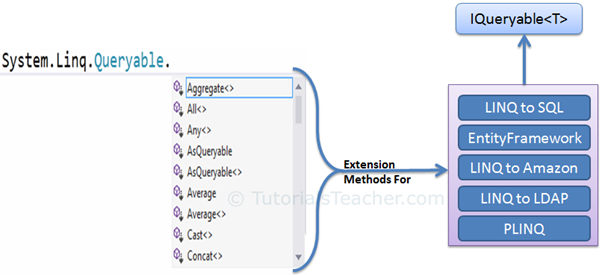


Queryable

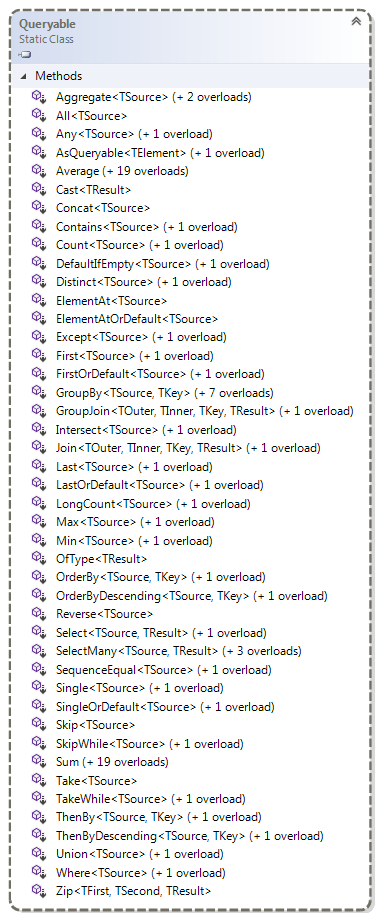
Queryable là lớp bao gồm các phương thức mở rộng cho các lớp hiện thực interface IQueryable<T>. IQueryable<T> interface được sử dụng để cung cấp các khả năng truy vấn đối với một nguồn dữ liệu cụ thể trong đó loại dữ liệu được biết. Ví dụ, API Entity Framework triển khai interface IQueryable<T> để hỗ trợ các truy vấn LINQ với cơ sở dữ liệu lót như MS SQL Server.

Ngoài ra, có các API có sẵn để truy cập dữ liệu của bên thứ ba. Ví dụ: LINQ to Amazon cung cấp khả năng sử dụng LINQ với các dịch vụ web của Amazon để tìm kiếm sách và các mặt hàng khác. Điều này có thể đạt được bằng cách triển khai interface IQueryable<T> cho Amazon.

Hình dưới đây cho thấy các phương thức mở rộng có sẵn trong lớp Truy vấn có thể được sử dụng với các nhà cung cấp dữ liệu gốc hoặc bên thứ ba khác.



Hình dưới đây cho thấy các phương thức mở rộng có sẵn trong lớp Truy vấn.



**Những điểm cần nhớ:**

* Sử dụng không gian tên System.Linq để sử dụng LINQ.
* API LINQ bao gồm hai lớp tĩnh chính Enumerable và Queryable.
* Lớp Enumerable tĩnh bao gồm các phương thức mở rộng cho các lớp hiện thực interface IEnumerable<T>.
* Lớp Queriable tĩnh bao gồm các phương thức mở rộng cho các lớp hiện thực interface IQueryable<T>.

**Cú pháp truy vấn LINQ (Query Syntax)**

Query syntax tương tự như SQL (Ngôn ngữ truy vấn có cấu trúc) cho cơ sở dữ liệu. Nó được định nghĩa trong mã C # hoặc VB.

Cú pháp truy vấn LINQ bắt đầu bằng từ khóa và kết thúc bằng từ khóa select. Sau đây là một truy vấn LINQ mẫu trả về một tập hợp các chuỗi có chứa một từ "Tutorials"

// string collection

IList<string> stringList = new List<string>() {

"C# Tutorials",

"VB.NET Tutorials",

"Learn C++",

"MVC Tutorials" ,

"Java"

};

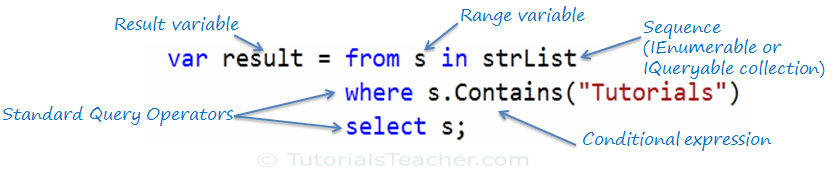
// LINQ Query Syntax

var result = from s in stringList

where s.Contains("Tutorials")

select s;

Hình dưới đây cho thấy cấu trúc của cú pháp truy vấn LINQ.



Cú pháp truy vấn bắt đầu bằng mệnh đề From, theo sau là một biến phạm vi. Mệnh đề From được cấu trúc như sau "**From** rangeV*ariableName* **in** *IEnumerablecollection*". Có nghĩa là từ mỗi đối tượng trong collection. Nó tương tự như một vòng lặp foreach “foreach(Student s in studentList)”

Sau mệnh đề From, bạn có thể sử dụng các toán tử truy vấn tiêu chuẩn khác nhau để lọc, nhóm, nối các phần tử của collection. Có khoảng 50 Toán tử truy vấn tiêu chuẩn có sẵn trong LINQ. Trong hình trên, chúng ta đã sử dụng toán tử "where" theo sau là một điều kiện. Điều kiện này thường được thể hiện bằng cách sử dụng biểu thức lambda.

Cú pháp truy vấn LINQ luôn kết thúc bằng mệnh đề Select hoặc Group.

Mệnh đề Select được sử dụng để định hình dữ liệu. Bạn có thể chọn toàn bộ đối tượng như nó hoặc chỉ một số thuộc tính của nó. Trong ví dụ trên, chúng tôi đã chọn từng thành phần chuỗi kết quả.

Trong ví dụ sau, chúng tôi sử dụng cú pháp truy vấn LINQ để tìm ra các học sinh thiếu niên từ collection (String) của Student.

**C#**

// Student collection

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 13} ,

new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20} ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }

};

// LINQ Query Syntax to find out teenager students

var teenAgerStudent = from s in studentList

where s.Age > 12 && s.Age < 20

select s;

**VB.NET**

// Student collection

Dim studentList = New List(Of Student) From {

New Student() With {.StudentID = 1, .StudentName = "John", .Age = 13},

New Student() With {.StudentID = 2, .StudentName = "Moin", .Age = 21},

New Student() With {.StudentID = 3, .StudentName = "Bill", .Age = 18},

New Student() With {.StudentID = 4, .StudentName = "Ram", .Age = 20},

New Student() With {.StudentID = 5, .StudentName = "Ron", .Age = 15}

}

// LINQ Query Syntax to find out teenager students

Dim teenAgerStudents As IList(Of Student) = (From s In studentList \_

Where s.Age > 12 And s.Age < 20 \_

Select s).ToList()

**Những điểm cần nhớ**

* Cú pháp truy vấn giống như cú pháp SQL (Ngôn ngữ truy vấn cấu trúc).
* Cú pháp truy vấn bắt đầu bằng mệnh đề from và có thể kết thúc bằng mệnh đề Select hoặc GroupBy.
* Sử dụng nhiều yếu tố khác như lọc, nối, nhóm, sắp xếp toán tử để xây dựng kết quả mong muốn
* Implicitly typed variable - var có thể được sử dụng để giữ kết quả của truy vấn LINQ.

**Cú pháp phương thức LINQ (Method Syntax)**

Sử dụng các phương thức mở rộng được bao gồm trong lớp tĩnh Enumerable hoặc Queryable. Tương tự như cách bạn sẽ gọi phương thức mở rộng của bất kỳ lớp nào.

Sau đây là một truy vấn cú pháp phương thức LINQ mẫu trả về một tập hợp các chuỗi có chứa một từ "Tutorials".

// string collection

IList<string> stringList = new List<string>() {

"C# Tutorials",

"VB.NET Tutorials",

"Learn C++",

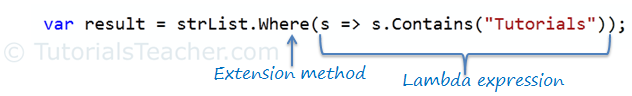
"MVC Tutorials" ,

"Java"

};

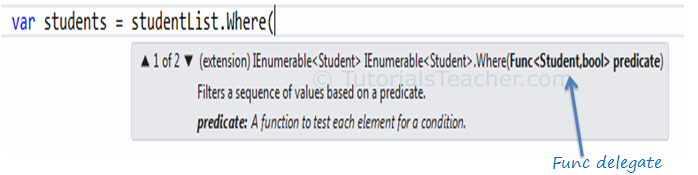
// LINQ Query Syntax

var result = stringList.Where(s => s.Contains("Tutorials"));

Hình dưới đây minh họa cấu trúc của cú pháp phương thức LINQ.

Như bạn có thể thấy trong hình trên, cú pháp phương thức bao gồm các phương thức mở rộng và biểu thức Lambda. Phương thức mở rộng Where() được định nghĩa trong lớp Enumerable.

Nếu bạn kiểm tra chữ ký của phương thức mở rộng Where, bạn sẽ thấy phương thức Where chấp nhận một predicate delegate là Func <Student, bool>. Điều này có nghĩa là bạn có thể truyền bất kỳ hàm delegate nào chấp nhận đối tượng Student làm tham số đầu vào và trả về giá trị Boolean như trong hình dưới đây. Biểu thức lambda hoạt động như delegate được truyền vào trong mệnh đề Where.



Ví dụ sau đây cho thấy cách sử dụng truy vấn cú pháp phương thức LINQ với collection IEnumerable<T>.

**C#**

// Student collection

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 13} ,

new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20} ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }

};

// LINQ Method Syntax to find out teenager students

var teenAgerStudents = studentList.Where(s => s.Age > 12 && s.Age < 20)

.ToList<Student>();

**VB.NET**

// Student collection

Dim studentList = New List(Of Student) From {

New Student() With {.StudentID = 1, .StudentName = "John", .Age = 13},

New Student() With {.StudentID = 2, .StudentName = "Moin", .Age = 21},

New Student() With {.StudentID = 3, .StudentName = "Bill", .Age = 18},

New Student() With {.StudentID = 4, .StudentName = "Ram", .Age = 20},

New Student() With {.StudentID = 5, .StudentName = "Ron", .Age = 15}

}

// LINQ Method Syntax to find out teenager students

Dim teenAgerStudents As IList(Of Student) = studentList.Where(Function(s) s.Age > 12 And s.Age < 20).ToList()

**Những điểm cần nhớ:**

* LINQ Method Syntax giống như gọi phương thức mở rộng.
* LINQ Method Syntax cho phép một loạt các phương thức mở rộng gọi.
* Implicitly typed variable – var có thể được sử dụng để giữ kết quả của truy vấn LINQ.

**Biểu thức Lambda**

***Cấu trúc của một biểu thức Lambda***

C # 3.0 (.NET 3.5) đã giới thiệu biểu thức lambda cùng với LINQ. Biểu thức lambda là một cách ngắn hơn để biểu diễn phương thức ẩn danh bằng cách sử dụng một số cú pháp đặc biệt.

Ví dụ: sau kiểm tra phương pháp ẩn danh nếu học sinh là thiếu niên hay không:

**C#**

delegate(Student s) { return s.Age > 12 && s.Age < 20; };

**VB.NET**

Dim isStudentTeenAger = Function(s As Student) As Boolean

Return s.Age > 12 And s.Age < 20

End Function

Phương thức ẩn danh ở trên có thể được biểu diễn bằng Biểu thức Lambda trong C # và VB.Net như sau:

**C#**

s => s.Age > 12 && s.Age < 20

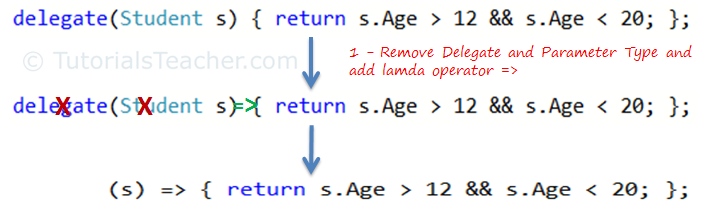
**VB.NET**

Function(s) s.Age > 12 And s.Age < 20

Chúng ta hãy xem biểu thức lambda phát triển như thế nào từ phương thức ẩn danh sau.

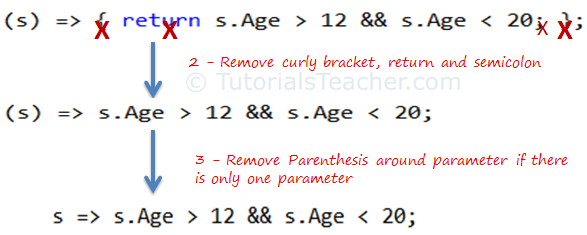
delegate(Student s) { return s.Age > 12 && s.Age < 20; };

Biểu thức Lambda phát triển từ phương thức ẩn danh bằng cách trước tiên loại bỏ tham số và từ khóa delegate và thêm toán tử lambda =>.

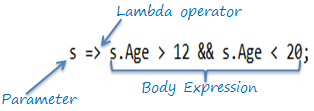


Biểu thức lambda ở trên là hoàn toàn hợp lệ, nhưng chúng ta không cần dấu ngoặc nhọn, return và dấu chấm phẩy nếu chúng ta chỉ có một câu lệnh trả về giá trị. Vì vậy, chúng ta có thể loại bỏ nó.

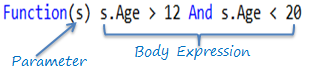
Ngoài ra, chúng ta có thể loại bỏ dấu ngoặc đơn (), nếu chúng ta chỉ có một tham số.



Như vậy, chúng tôi có biểu thức lambda: s => s.Age > 12 && s.Age < 20. S là một tham số, => là toán tử lambda, **s.Age > 12 && s.Age < 20** là thân của biểu thức.



Giống như cách chúng ta có biểu thức lambda trong VB.Net có thể được viết như dưới đây:



Biểu thức lambda có thể được gọi giống như delegate using ().

Lưu ý: VB.Net không hỗ trợ toán tử lambda =>.

***Biểu thức Lambda với nhiều tham số***

Bạn có thể gói các tham số trong ngoặc đơn nếu bạn cần truyền nhiều hơn một tham số, như sau:

**C#**

**(s, youngAge)** => s.Age >= youngage;

Bạn cũng có thể đưa ra loại của từng tham số nếu tham số gây nhầm lẫn:

**C#**

**(Student s,int youngAge)** => s.Age >= youngage;

**VB.NET**

Function(s, youngAge) s.Age >= youngAge

***Biểu thức Lambda không tham số***

Không nhất thiết phải có ít nhất một tham số trong biểu thức lambda. Biểu thức lambda có thể được chỉ định mà không có bất kỳ tham số nào.

**()** => Console.WriteLine("Parameter less lambda expression")

***Nhiều câu lệnh trong thân của biểu thức Lambda***

Bạn có thể gói biểu thức trong dấu ngoặc nhọn nếu bạn muốn có nhiều hơn một câu lệnh trong thân của biểu thức:

**C#**

(s, youngAge) =>

**{**

Console.WriteLine("Lambda expression with multiple statements in the body");

Return s.Age >= youngAge;

**}**

**VB.NET**

Function(s , youngAge)

Console.WriteLine("Lambda expression with multiple statements in the body")

Return s.Age >= youngAge

End Function

***Khai báo biến cục bộ trong thân của biểu thức Lambda***

Bạn có thể khai báo một biến trong thân biểu thức để sử dụng nó ở bất cứ đâu trong thân biểu thức, như sau:

**C#**

s =>

{

**int youngAge = 18;**

Console.WriteLine("Lambda expression with multiple statements in the body");

return s.Age >= youngAge;

}

**VB.NET**

Function(s)

**Dim youngAge As Integer = 18**

Console.WriteLine("Lambda expression with multiple statements in the body")

Return s.Age >= youngAge

End Function

Biểu thức Lambda cũng có thể được gán cho các thành phần được xây dựng trong delegate như Func, Action và Predicate

***Gán biểu thức Lambda cho Delegate***

Biểu thức lambda có thể được gán cho loaị delegate Func<in T, out TResult>. Kiểu tham số cuối cùng trong một delegate func là loại trả về và phần còn lại là các tham số đầu vào.

Hãy xem xét biểu thức lambda sau đây để tìm hiểu xem một học sinh có phải là một thiếu niên hay không.

**C#**

Func<Student, bool> isStudentTeenAger = s => s.age > 12 && s.age < 20;

Student std = new Student() { age = 21 };

bool isTeen = isStudentTeenAger(std);// returns false

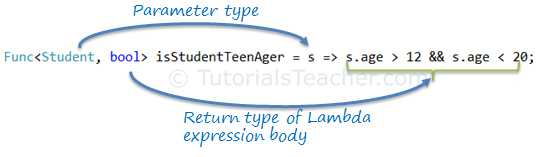
**VB.NET**

Dim isStudentTeenAger As Func(Of Student, Boolean) = Function(s) s.Age > 12 And s.Age < 20

Dim stud As New Student With {.Age = 21}

Dim isTeen As Boolean = isStudentTeenAger(stud) // returns false

Trong ví dụ trên, func delegate yêu cầu ​​tham số đầu vào đầu tiên là loại Sinh viên và loại trả về là kiểu boolean. Biểu thức lambda s => s.age > 12 && s.age < 20 thỏa yêu cầu func delegate, như hình dưới đây:



Func delegate hiển thị ở trên, sẽ trở thành một function như dưới đây.

bool isStudentTeenAger(Student s)

{

return s.Age > 12 && s.Age < 20;

}

***Action Delegate***

Không giống như func delegate. Action delegate chỉ có thể có các tham số đầu vào. Sử dụng action delegate khi bạn không cần trả về bất kỳ giá trị nào từ biểu thức lambda.

C#

Action<Student> PrintStudentDetail = s => Console.WriteLine("Name: {0}, Age: {1} ", s.StudentName, s.Age);

Student std = new Student(){ StudentName = "Bill", Age=21};

PrintStudentDetail(std);//output: Name: Bill, Age: 21

VB.NET

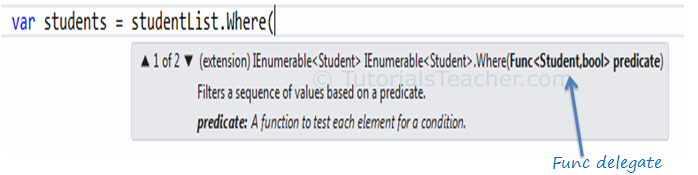
Dim printStudentDetail As Action(Of Student) = Sub(s) Console.WriteLine("Name: {0}, Age: {1} ", s.StudentName, s.Age)

Dim stud As New Student With {.StudentName = "Bill", .Age = 21}

printStudentDetail(stud)//output: Name: Bill, Age: 21

***Biểu thức Lambda trong truy vấn LINQ***

Thông thường biểu thức lambda được sử dụng với truy vấn LINQ. Lớp tĩnh Enumerable bao gồm phương thức mở rộng where cho IEnumerable<T> chấp nhận Func<TSource,bool>. Vì thế, phương thức mở rộng Where() cho collection IEnumerable<Student> được yêu cầu phải nhận Func<Student,bool>. như hình dưới đây:



Vậy bây giờ,bạn có thể đưa biểu thức lambda gán cho func delegate , cho phương thức mở rộng where() trong cú pháp phương thức như dưới đây:

**C#**

IList<Student> studentList = new List<Student>(){...};

Func<Student, bool> isStudentTeenAger = s => s.age > 12 && s.age < 20;

var teenStudents = studentList.Where(isStudentTeenAger).ToList<Student>();

**VB.NET**

IList<Student> studentList = new List<Student>(){...};

Func<Student, bool> isStudentTeenAger = s => s.age > 12 && s.age < 20;

var teenStudents = from s in studentList

where isStudentTeenAger(s)

select s;

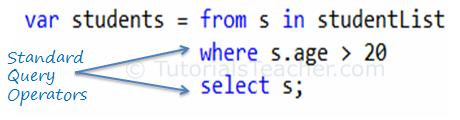
**Những điểm cần nhớ:**

* Biểu thức Lambda à một cách ngắn hơn để biểu diễn phương thức ẩn danh.
* Cú pháp biểu thức Lambda: *parameters => body expression*
* Biểu thức Lambda có thể không có tham số
* Biểu thức Lambda có thể có nhiều tham số trong ngoặc đơn ().
* Biểu thức Lambda có thể có nhiều câu lệnh trong biểu thức cơ thể trong dấu ngoặc nhọn {}.
* Biểu thức Lambda có thể được gán cho Func, Action hoặc Predicate delegate.
* Lambda Expression có thể được gọi theo cách tương tự như delegate.

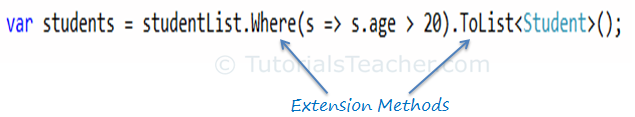
**Toán tử truy vấn chuẩn**

Toán tử truy vấn chuẩn trong LINQ thực sự là các phương thức mở rộng cho loại IEnumerable<T> và IQueryable<T>. Chúng được định nghĩa trong các class System.Linq.Enumerable và System.Linq.Queryable. Có hơn 50 toán tử truy vấn tiêu chuẩn có sẵn trong LINQ cung cấp các chức năng khác nhau như lọc, sắp xếp, nhóm, tổng hợp, nối, v.v.

Toán tử truy vấn chuẩn trong Query syntax



Toán tử truy vấn chuẩn trong Method syntax



Các toán tử truy vấn chuẩn trong cú pháp truy vấn được chuyển đổi thành các phương thức mở rộng tại thời gian biên dịch. Vì vậy, cả hai đều giống nhau.

Toán tử truy vấn tiêu chuẩn có thể được phân loại dựa trên chức năng họ cung cấp. Bảng sau liệt kê tất cả các phân loại của Toán tử truy vấn tiêu chuẩn:

| Phân loại | Các toán tử truy vấn chuẩn |
| --- | --- |
| Lọc | Where, OfType |
| Sắp xếp | OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse |
| Nhóm | GroupBy, ToLookup |
| Phép kết | GroupJoin, Join |
| Phép chiếu | Select, SelectMany |
| Tập hợp | Aggregate, Average, Count, LongCount, Max, Min, Sum |
| Định lượng | All, Any, Contains |
| Phạm vị | ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault |
| Đặt | Distinct, Except, Intersect, Union |
| Phân vùng | Skip, SkipWhile, Take, TakeWhile |
| Ghép | Concat |
| Bằng | SequenceEqual |
| Sinh | DefaultEmpty, Empty, Range, Repeat |
| Chuyển đổi | AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList |

**Toán tử lọc - Where**

Các toán tử lọc trong LINQ lọc liên tục (collection) dựa trên một số tiêu chí đã cho. Bảng sau liệt kê tất cả các toán tử lọc có sẵn trong LINQ.

| Toán tử lọc | Mô tả |
| --- | --- |
| [Where](https://www.tutorialsteacher.com/linq/linq-filtering-operators-where#where) | Trả về giá trị từ collection được xây dựng trên predicate delegate |
| [OfType](https://www.tutorialsteacher.com/linq/linq-filtering-operators-oftype) | Trả về giá trị từ collection được xây dựng trên một loại xác định. Tuy nhiên, nó sẽ phụ thuộc vào khả năng của nó để chuyển sang một loại cụ thể. |

***Where***

Toán tử Where lọc collectioon dựa trên biểu thức tiêu chí đã cho và trả về collection mới. Các tiêu chí có thể được chỉ định là biểu thức lambda hoặc loại Func delegate.

Phương thức mở rộng Where có hai overload. Cả hai phương thức overload đều chấp nhận một loại tham số Func delegate. Một phương thức overload yêu cầu tham số đầu vào Func <TSource, bool> và phương thức overload thứ hai được yêu cầu Func <TSource, int, bool> tham số đầu vào, trong đó int dành cho chỉ mục(index):

public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);

public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, int, bool> predicate);

**Mệnh đề Where trong Query Syntax**

Mẫu truy vấn sau đây sử dụng toán tử Where để lọc các sinh viên tuổi teen từ collection (chuỗi) đã cho. Nó sử dụng biểu thức lambda như predicate function.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 13} ,

new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20} ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }

};

var filteredResult = from s in studentList

where s.Age > 12 && s.Age < 20

select s.StudentName;

**VB.NET**

Dim studentList = New List(Of Student) From {

New Student() With {.StudentID = 1, .StudentName = "John", .Age = 13},

New Student() With {.StudentID = 2, .StudentName = "Moin", .Age = 21},

New Student() With {.StudentID = 3, .StudentName = "Bill", .Age = 18},

New Student() With {.StudentID = 4, .StudentName = "Ram", .Age = 20},

New Student() With {.StudentID = 5, .StudentName = "Ron", .Age = 15}

}

Dim filteredResult = From s In studentList

Where s.Age > 12 And s.Age < 20

Select s.StudentName

Trong ví dụ trên, filterResult sẽ bao gồm các sinh viên sau khi thực hiện truy vấn.

John   
Bill  
Ron

Trong truy vấn mẫu ở trên, phần thân biểu thức lambda s.Age> 12 && s.Age <20 được truyền dưới dạng predicate function Func <TSource, bool> để đánh giá mọi sinh viên trong bộ sưu tập.

Ngoài ra, bạn cũng có thể sử dụng Func delegate với phương thức ẩn danh để chuyển dưới dạng predicate function như bên dưới (đầu ra sẽ giống nhau):

Func<Student,bool> isTeenAger = delegate(Student s) {

return s.Age > 12 && s.Age < 20;

};

var filteredResult = from s in studentList

where isTeenAger(s)

select s;

Bạn cũng có thể gọi bất kỳ phương thức nào khớp với tham số Func với một trong các phương thức overload Where ()

public static void Main()

{

var filteredResult = from s in studentList

where isTeenAger(s)

select s;

}

public static bool IsTeenAger(Student stud)

{

return stud.Age > 12 && stud.Age < 20;

}

**Phương thức mở rộng Where trong Method Syntax**

Không giống như query syntax, bạn cần truyền toàn bộ biểu thức lambda dưới dạng predicate function thay vì chỉ thân biểu thức trong cú pháp phương thức LINQ.

C#

var filteredResult = studentList.Where(s => s.Age > 12 && s.Age < 20);

VB.NET

Dim filteredResult = studentList.Where(Function(s) s.Age > 12 And s.Age < 20 )

Như đã đề cập ở trên, phương thức mở rộng Where cũng có hai overload bao gồm chỉ mục của phần tử hiện tại trong collection. Bạn có thể sử dụng chỉ số đó trong logic của bạn nếu bạn cần.

Ví dụ sau sử dụng mệnh đề Where để lọc ra các phần tử lẻ trong bộ sưu tập và chỉ trả về các phần tử chẵn. Hãy nhớ rằng chỉ số bắt đầu từ số không.

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 }

};

var filteredResult = studentList.Where((s, i) => {

if(i % 2 == 0) // if it is even element

return true;

return false;

});

foreach (var std in filteredResult)

Console.WriteLine(std.StudentName);

**Mệnh đề Where nhiều lần**

Bạn có thể gọi phương thức mở rộng Where () nhiều lần trong một truy vấn LINQ.

**Query Syntax**

var filteredResult = from s in studentList

where s.Age > 12

where s.Age < 20

select s;

**Method Syntax**

var filteredResult = studentList.Where(s => s.Age > 12).Where(s => s.Age < 20);

**Những điểm cần nhớ:**

* Where được sử dụng để lọc collection dựa trên các tiêu chí nhất định.
* Phương thức mở rộng Where có hai phương thức overload. Sử dụng phương thức overload thứ hai để biết chỉ số của phần tử hiện tại trong collection.
* Method syntax yêu cầu toàn bộ biểu thức lambda trong phương thức mở rộng Where trong khi Query syntax chỉ yêu cầu phần thân biểu thức
* Phương thức mở rộng Where nhiều lần là hợp lệ trong một truy vấn LINQ duy nhất.

**Toán tử lọc - OfType**

Toán tử OfType lọc collection dựa trên khả năng truyền một phần tử trong bộ sưu tập thành một loại được xác định.

**OfType trong Query Syntax**

Sử dụng toán tử OfType để lọc collection trên dựa trên kiểu của từng thành phần

**C#**

IList mixedList = new ArrayList();

mixedList.Add(0);

mixedList.Add("One");

mixedList.Add("Two");

mixedList.Add(3);

mixedList.Add(new Student() { StudentID = 1, StudentName = "Bill" });

var stringResult = from s in mixedList.OfType<string>()

select s;

var intResult = from s in mixedList.OfType<int>()

select s;

**VB.NET**

Dim stringResult = From s In mixedList.OfType(Of String)()

Các truy vấn mẫu ở trên sẽ trả về các mục có kiểu là string mixedList. stringResult chứa các phần tử sau khi thực hiện:

One  
Two   
0   
3   
Bill

**OfType trong Method Syntax**

Bạn có thể sử dụng phương thức mở rộng OfType <TResult> () trong method syntax linq như bên dưới.

**C#**

var stringResult = mixedList.OfType<string>();

**VB.NET**

Dim stringResult = mixedList.OfType(Of String)

StringResult sẽ chứa các phần tử sau.

One  
Two

**Những điểm cần nhớ**

* Toán tử Where lọc collection dựa trên predicate function.
* Toán tử ofType lọc collection dựa trên một loại nhất định.
* Phương thức mở rộng Where và OfType có thể được gọi nhiều lần trong một truy vấn LINQ.

**Toán tử sắp xếp – OrderBy và OrderByDescending**

Một toán tử sắp xếp sắp xếp các phần tử của bộ sưu tập theo thứ tự tăng dần hoặc giảm dần. LINQ bao gồm các toán tử sắp xếp sau.

| Toán tử sắp xếp | Mô tả |
| --- | --- |
| [OrderBy](https://www.tutorialsteacher.com/linq/linq-sorting-operators-orderby-orderbydescending#orderby) | Sắp xếp các phần tử trong collection dựa trên các trường được xác định theo thứ tự tăng dần hoặc giảm dần. |
| [OrderByDescending](https://www.tutorialsteacher.com/linq/linq-sorting-operators-orderby-orderbydescending#orderbydescending) | Sắp xếp collection dựa trên các trường được xác định theo thứ tự giảm dần. Chỉ hợp lệ trong method syntax. |
| [ThenBy](https://www.tutorialsteacher.com/linq/linq-sorting-operators-thenby-thenbydescending) | Chỉ hợp lệ trong method syntax. Được sử dụng để sắp xếp cấp độ thứ hai theo thứ tự tăng dần. |
| [ThenByDescending](https://www.tutorialsteacher.com/linq/linq-sorting-operators-thenby-thenbydescending) | Chỉ hợp lệ trong method syntax. Được sử dụng để sắp xếp cấp độ thứ hai theo thứ tự giảm dần. |

**OderBy**

Sắp xếp các giá trị của một collection theo thứ tự tăng dần hoặc giảm dần. Nó sắp xếp collection theo thứ tự tăng dần theo mặc định. Bởi vì từ khóa ascending là tùy chọn ở đây. Dùng từ khóa descending để sắp xếp bộ sưu tập theo thứ tự giảm dần.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 }

};

var orderByResult = from s in studentList

orderby s.StudentName

select s;

var orderByDescendingResult = from s in studentList

orderby s.StudentName descending

select s;

**VB.NET**

Dim orderByResult = From s In studentList

Order By s.StudentName

Select s

Dim orderByDescendingResult = From s In studentList

Order By s.StudentName Descending

Select s

orderByResult trong ví dụ trên sẽ chứa các phần tử sau sau khi thực thi:

Bill   
John   
Ram  
Ron  
Steve

orderByDesceinatingResult trong ví dụ trên sẽ chứa các phần tử sau sau khi thực hiện:

Steve  
Ron  
Ram  
John   
Bill

**OderBy trong method syntax**

Phương thức mở rộng OrderBy có hai overload. Overload đầu tiên của phương thức mở rộng OrderBy nhận loại tham số func delegate. Vì vậy, bạn cần truyền biểu thức lambda cho trường mà bạn muốn sắp xếp collection. Overload thứ hai của OrderBy nhận đối tượng của IComparer cùng với loại Func delegate để sử dụng so sánh tùy chỉnh để sắp xếp.

public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(this IEnumerable<TSource> source,Func<TSource, TKey> keySelector);

public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector, IComparer<TKey> comparer);

Ví dụ sau đây sắp xếp collection studentList theo thứ tự tăng dần của StudentName bằng phương thức mở rộng OrderBy.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 }

};

var studentsInAscOrder = studentList.OrderBy(s => s.StudentName);

**VB.NET**

Dim studentsInAscOrder = studentList.OrderBy(Function(s) s.StudentName)

Lưu ý: Method syntax không cho phép từ khóa decending để sắp xếp collection theo thứ tự giảm dần. Sử dụng phương thức OrderByDecending () cho để giải quyết trường hợp này.

**OderByDescending**

OrderByDesceinating sắp xếp collection theo thứ tự giảm dần. OrderByDesceinating chỉ hợp lệ với Method syntax. Nó không hợp lệ trong query syntax vì cú pháp truy vấn sử dụng các thuộc tính tăng dần và giảm dần như được hiển thị ở trên.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 }

};

var studentsInDescOrder = studentList.OrderByDescending(s => s.StudentName);

**VB.NET**

Dim studentsInDescOrder = studentList.OrderByDescending(Function(s) s.StudentName)

Một kết quả trong ví dụ trên sẽ chứa các phần tử sau sau khi thực hiện.

Steve  
Ron  
Ram  
John   
Bill

Xin lưu ý rằng OrderByDesceinating không được hỗ trợ trong Query syntax. Sử dụng từ khóa descending để thay thế.

**Multiple Sorting**

Bạn có thể sắp xếp bộ sưu tập trên nhiều trường được phân tách bằng dấu phẩy. Collection đã cho sẽ được sắp xếp đầu tiên dựa trên trường đầu tiên và sau đó nếu giá trị của trường đầu tiên sẽ giống nhau đối với hai phần tử, sau đó nó sẽ sử dụng trường thứ hai để sắp xếp và cứ thế.

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 },

new Student() { StudentID = 6, StudentName = "Ram" , Age = 18 }

};

var orderByResult = from s in studentList

orderby s.StudentName, s.Age

select new { s.StudentName, s.Age };

Trong ví dụ trên, collection studentList bao gồm hai StudentNames giống hệt nhau, Ram. Vì vậy, bây giờ, studentList sẽ được sắp xếp đầu tiên dựa trên StudentName và sau đó theo Age theo thứ tự tăng dần. Vì vậy, orderByResult sẽ chứa các phần tử sau sau khi thực thi.

StudentName: Bill, Age: 25  
StudentName: John, Age: 18   
StudentName: Ram, Age: 18  
StudentName: Ram, Age: 20  
StudentName: Ron, Age: 19  
StudentName: Steve, Age: 15

**Những điểm cần nhớ**

* LINQ bao gồm năm toán tử sắp xếp: OrderBy, OrderByDesceinating, ThenBy, ThenByDesceinating và Reverse.
* LINQ Query Syntax không hỗ trợ OrderByDesceinating, ThenBy, ThenByDesceinating và Reverse. Nó chỉ hỗ trợ mệnh đề OrderBy' với hướng sắp xếp ‘ascending’ và ‘descending’.
* LINQ Query Syntax hỗ trợ sắp xếp nhiều trường phân tách bằng dấu phẩy trong khi bạn phải sử dụng các phương thức ThenBy & ThenByDesceinating để sắp xếp thứ cấp.

**Toán tử sắp xếp – ThenBy và ThenByDescending**

Chúng ta đã thấy cách thực hiện sắp xếp bằng nhiều trường trong cú pháp truy vấn trong phần trước. Nhiều cách sắp xếp theo method syntax được hỗ trợ bằng cách sử dụng các phương thức mở rộng ThenBy và ThenByDesceinating.

Phương thức ThenBy() theo sau OrderBy() để sắp xếp collection trên một trường khác theo thứ tự tăng dần. LINQ đầu tiên sẽ sắp xếp collection dựa trên trường chính được chỉ định bởi phương thức OrderBy và sau đó sắp xếp collection kết quả theo thứ tự tăng dần một lần nữa dựa trên trường thứ cấp được chỉ định bởi phương thức ThenBy.

Tương tự, sử dụng phương thức ThenByDesceinating để áp dụng sắp xếp thứ cấp theo thứ tự giảm dần.

Ví dụ sau đây cho thấy cách sử dụng phương thức ThenBy và ThenByDesceinating để sắp xếp cấp độ thứ hai:

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 },

new Student() { StudentID = 6, StudentName = "Ram" , Age = 18 }

};

var thenByResult = studentList.OrderBy(s => s.StudentName).ThenBy(s => s.Age);

var thenByDescResult = studentList.OrderBy(s => s.StudentName).ThenByDescending(s => s.Age);

Như bạn có thể thấy trong ví dụ trên, trước tiên chúng ta sắp xếp một studentList collection theo studentName và sau đó là Age. Vậy bây giờ, thenByResult sẽ chứa các phần tử sau khi sắp xếp:

StudentName: Bill, Age: 25   
StudentName: John, Age: 18   
StudentName: Ram, Age: 18  
StudentName: Ram, Age: 20  
StudentName: Ron, Age: 19  
StudentName: Steve, Age: 15

thenByDescResult sẽ chứa các phần tử sau. Xin lưu ý rằng Ram 20 tuổi ở trước Ram 18 tuổi vì ta đã sử dụng ThenByDescending.

StudentName: Bill, Age: 25   
StudentName: John, Age: 18   
StudentName: Ram, Age: 20  
StudentName: Ram, Age: 18  
StudentName: Ron, Age: 19  
StudentName: Steve, Age: 15

Bạn có thể sử dụng phương thức ThenBy và ThenByDesceinating theo cách tương tự trong VB.Net như sau:

Dim sortedResult = studentList.OrderBy(Function(s) s.StudentName)

.ThenBy(Function(s) s.Age)

Dim sortedResult = studentList.OrderBy(Function(s) s.StudentName)

.ThenByDescending(Function(s) s.Age)

**Những điểm cần nhớ**

* OrderBy và ThenBy sắp xếp các collection theo thứ tự tăng dần mặc định.
* ThenBy hoặc ThenByDesceinating được sử dụng để sắp xếp mức thứ hai trong method syntax.
* Phương thức ThenByDescending sắp xếp collection theo thứ tự giảm dần trên một trường khác.
* ThenBy hoặc ThenByDesceinating KHÔNG được áp dụng trong Query syntax.
* Áp dụng sắp xếp thứ cấp trong cú pháp truy vấn bằng cách tách các trường bằng dấu phẩy.

**Toán tử gom nhóm – GroupBy và ToLookUp**

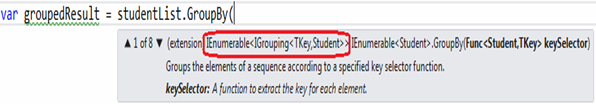
Các toán tử nhóm thực hiện tương tự như mệnh đề GroupBy của truy vấn SQL. Các toán tử nhóm tạo một nhóm các phần tử dựa trên khóa đã cho. Nhóm này được chứa trong một loại collection đặc biệt hiện thực một interface IGrouping<TKey,TSource> trong đó TKey là một giá trị khóa, trên đó nhóm đã được hình thành và TSource là tập hợp các phần tử khớp với giá trị khóa nhóm.

| Toán tử nhóm | Mô tả |
| --- | --- |
| [GroupBy](https://www.tutorialsteacher.com/linq/linq-grouping-operator-groupby-tolookup#groupby) | Toán tử GroupBy trả về các nhóm phần tử dựa trên một số giá trị khóa. Mỗi nhóm được đại diện bởi đối tượng IGrouping <TKey, TEuity>. |
| [ToLookup](https://www.tutorialsteacher.com/linq/tolookup) | ToLookup giống như GroupBy; sự khác biệt duy nhất là việc thực thi GroupBy bị hoãn trong khi thực thi ToLookup là ngay lập tức. |

**GroupBy**

Toán tử GroupBy trả về một nhóm các phần tử từ bộ sưu tập đã cho dựa trên một số giá trị khóa. Mỗi nhóm được đại diện bởi đối tượng IGrouping <TKey, TEuity>. Ngoài ra, phương thức GroupBy có tám phương thức overload, vì vậy bạn có thể sử dụng phương thức mở rộng phù hợp dựa trên yêu cầu của bạn trong method syntax.

Kết quả của các toán tử GroupBy là một tập hợp các nhóm. Ví dụ: GroupBy trả về IENumerable<IGrouping <TKey, Student >> từ collection Student:



**GroupBy trong Query Syntax**

Ví dụ sau đây tạo ra một nhóm sinh viên có cùng độ tuổi. Học sinh cùng tuổi sẽ ở trong cùng một collection và mỗi nhóm collection sẽ có một khóa và collection bên trong. Trong đó khóa sẽ là tuổi và collection bên trong sẽ bao gồm các học sinh có tuổi phù hợp với khóa.

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Abram" , Age = 21 }

};

var groupedResult = from s in studentList

group s by s.Age;

//iterate each group

foreach (var ageGroup in groupedResult)

{

Console.WriteLine("Age Group: {0}", ageGroup .Key); //Each group has a key

foreach(Student s in ageGroup) // Each group has inner collection

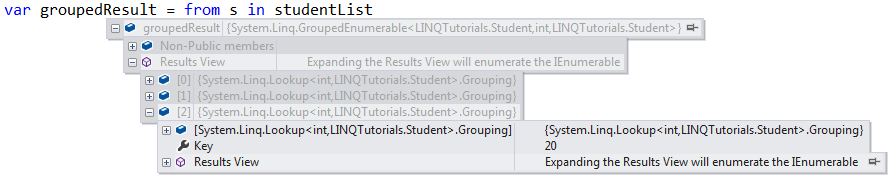
Console.WriteLine("Student Name: {0}", s.StudentName);

}

**Output:**

AgeGroup: 18  
StudentName: John  
StudentName: Bill  
AgeGroup: 21  
StudentName: Steve  
StudentName: Abram  
AgeGroup: 20  
StudentName: Ram

Như bạn có thể thấy trong ví dụ trên, bạn có thể lặp lại nhóm bằng cách sử dụng vòng lặp 'foreach', trong đó mỗi nhóm chứa một khóa và collection bên trong. Hình dưới đây cho thấy kết quả trong chế độ debug.



Sử dụng "Into Group" với mệnh đề 'Group By' trong VB.Net như bên dưới.

Dim groupQuery = From s In studentList

Group By s.Age Into Group

For Each group In groupQuery

Console.WriteLine("Age Group: {0}", group.Age) // Each group has key property name

For Each student In group.Group // Each group has inner collection

Console.WriteLine("Student Name: {0}", student.StudentName)

Next

Next

Lưu ý rằng mỗi nhóm sẽ có một tên thuộc tính mà nhóm được thực hiện. Trong ví dụ trên, chúng tôi đã sử dụng Age để tạo thành một nhóm để mỗi nhóm sẽ có tên thuộc tính "Age" thay vì "Key" làm tên thuộc tính.

AgeGroup: 18  
StudentName: John  
StudentName: Bill  
AgeGroup: 21  
StudentName: Steve  
StudentName: Abram  
AgeGroup: 20  
StudentName: Ram

**GroupBy trong Method Syntax**

Phương thức mở rộng GroupBy hoạt động theo cách tương tự trong Method syntax. Xác định biểu thức lambda cho tên của trường được chọn làm khóa trong phương thức mở rộng GroupBy.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Abram" , Age = 21 }

};

var groupedResult = studentList.GroupBy(s => s.Age);

foreach (var ageGroup in groupedResult)

{

Console.WriteLine("Age Group: {0}", ageGroup.Key); //Each group has a key

foreach(Student s in ageGroup) //Each group has a inner collection

Console.WriteLine("Student Name: {0}", s.StudentName);

}

**VB.NET**

Dim groupQuery = studentList.GroupBy(Function(s) s.Age)

For Each ageGroup In groupQuery

Console.WriteLine("Age Group: {0}", ageGroup.Key) //Each group has a key

For Each student In ageGroup.AsEnumerable() //Each group has a inner collection

Console.WriteLine("Student Name: {0}", student.StudentName)

Next

Next

**Output**

AgeGroup: 18  
StudentName: John  
StudentName: Bill  
AgeGroup: 21  
StudentName: Steve  
StudentName: Abram  
AgeGroup: 20  
StudentName: Ram

**ToLookUp**

ToLookup giống như GroupBy; sự khác biệt duy nhất là việc thực thi GroupBy được hoãn lại. Trong khi thực thi ToLookup là ngay lập tức. Ngoài ra, ToLookup chỉ áp dụng theo Method Syntax. ToLookup không được hỗ trợ trong Query Syntax.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Abram" , Age = 21 }

};

var lookupResult = studentList.ToLookup(s => s.age);

foreach (var group in lookupResult)

{

Console.WriteLine("Age Group: {0}", group.Key); //Each group has a key

foreach(Student s in group) //Each group has a inner collection

Console.WriteLine("Student Name: {0}", s.StudentName);

}

**VB.NET**

Dim loopupResult = studentList.ToLookup(Function(s) s.Age)

**Những điểm cần nhớ**

* GroupBy & ToLookup trả về một collection có khóa và collection bên trong dựa trên giá trị trường khóa.
* Việc thực thi GroupBy được hoãn lại trong khi đó ToLookup là ngay lập tức.
* LINQ Query Syntax có thể kết thúc bằng mệnh đề GroupBy hoặc Select.

**Toán tử kết – Join**

Các toán tử nối nối hai chuỗi (collection) và tạo ra kết quả.

| Toán tử kết | Sử dụng |
| --- | --- |
| [Join](https://www.tutorialsteacher.com/linq/linq-joining-operator-join#join) | Toán tử kết kết hai chuỗi (collection) dựa trên một khóa và trả về một chuỗi kết quả. |
| [GroupJoin](https://www.tutorialsteacher.com/linq/linq-joining-operator-groupjoin) | Toán tử GroupJoin kết hai chuỗi dựa trên các khóa và trả về các nhóm liên tục. Nó giống như kết trái của SQL. |

**JOIN**

Toán tử Join hoạt động trên hai collection, collection bên trong & collection bên ngoài. Nó trả về một collection mới chứa các phần tử từ cả hai collection thỏa mãn biểu thức đã chỉ định. Nó giống inner join của SQL.

**JOIN trong Method Syntax**

Phương thức mở rộng Join có hai overload như dưới đây.

public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer,

IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector,

Func<TInner, TKey> innerKeySelector,

Func<TOuter, TInner, TResult> resultSelector);

public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer,

IEnumerable<TInner> inner,

Func<TOuter, TKey> outerKeySelector,

Func<TInner, TKey> innerKeySelector,

Func<TOuter, TInner, TResult> resultSelector,

IEqualityComparer<TKey> comparer);

Như bạn có thể thấy trong phương thức nạp chồng đầu tiên có năm tham số đầu vào (ngoại trừ tham số 'this' đầu tiên): 1) outer 2) inner 3) outerKeySelector 4) innerKeySelector 5) resultSelector.

Hãy lấy một ví dụ đơn giản. Ví dụ sau đây kết hợp hai collection kiểu string và trả về collection mới bao gồm các string phù hợp trong cả hai collection.

IList<string> strList1 = new List<string>() {

"One",

"Two",

"Three",

"Four"

};

IList<string> strList2 = new List<string>() {

"One",

"Two",

"Five",

"Six"

};

var innerJoin = strList1.Join(strList2,

str1 => str1,

str2 => str2,

(str1, str2) => str1);

**Output**

One  
Two

Bây giờ, hãy hiểu phương thức Join bằng cách sử dụng lớp Student và Standard trong đó Student bao gồm StandardID phù hợp với StandardID của lớp Standard.

public class Student{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int StandardID { get; set; }

}

public class Standard{

public int StandardID { get; set; }

public string StandardName { get; set; }

}

Ví dụ sau đây cho thấy truy vấn LINQ Join.

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", StandardID =1 },

new Student() { StudentID = 2, StudentName = "Moin", StandardID =1 },

new Student() { StudentID = 3, StudentName = "Bill", StandardID =2 },

new Student() { StudentID = 4, StudentName = "Ram" , StandardID =2 },

new Student() { StudentID = 5, StudentName = "Ron" }

};

IList<Standard> standardList = new List<Standard>() {

new Standard(){ StandardID = 1, StandardName="Standard 1"},

new Standard(){ StandardID = 2, StandardName="Standard 2"},

new Standard(){ StandardID = 3, StandardName="Standard 3"}

};

var innerJoin = studentList.Join(// outer sequence

standardList, // inner sequence

student => student.StudentID, // outerKeySelector

standard => standard.StandardID, // innerKeySelector

(student, standard) => new // result selector

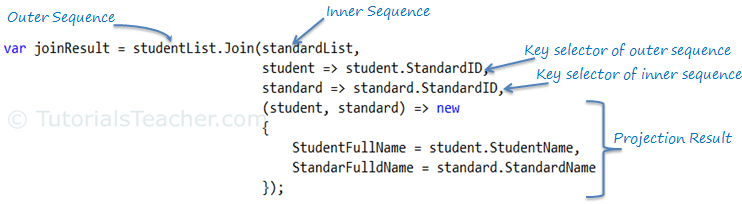
{

StudentName = student.StudentName,

StandardName = standard.StandardName

});

Hình ảnh sau đây minh họa các phần của toán tử Join trong ví dụ trên.



Trong ví dụ trên về truy vấn Join, studentList là chuỗi bên ngoài vì truy vấn bắt đầu từ nó. Tham số đầu tiên trong phương thức Join được sử dụng để xác định chuỗi bên trong là StandardList trong ví dụ trên. Tham số thứ hai và thứ ba của phương thức Join được sử dụng để chỉ định trường có giá trị khớp với biểu thức lambda để đưa phần tử vào kết quả. Bộ chọn khóa cho chuỗi bên ngoài student => student.StandardID chỉ ra rằng sẽ lấy trường StandardID của từng thành phần của studentList phải phù hợp với khóa của chuỗi bên trong standard => standard.StandardID. Nếu giá trị của cả hai trường khóa được khớp thì đưa phần tử đó vào kết quả.

Tham số cuối cùng trong phương thức Join là một biểu thức để tạo kết quả. Trong ví dụ trên, bộ chọn kết quả bao gồm thuộc tính StudentName và StandardName của cả hai chuỗi.

Khóa StandardID của cả hai chuỗi (collection) phải khớp nếu không item sẽ không được đưa vào kết quả. Ví dụ: Ron không được liên kết với bất kỳ tiêu chuẩn nào nên Ron không được đưa vào trong collection kết quả. InternalJoinResult trong ví dụ trên sẽ chứa các phần tử sau sau khi thực thi:

John - Standard 1  
Moin - Standard 1  
Bill - Standard 2  
Ram - Standard 2

Ví dụ sau đây cho thấy toán tử Join theo method syntax trong VB.Net.

Dim innerJoin = studentList.Join(standardList,

Function(s) s.StandardID,

Function(std) std.StandardID,

Function(s, std) New With

{

.StudentName = s.StudentName,

.StandardName = std.StandardName

});

**JOIN trong Query Syntax**

Toán tử Join trong Query Syntax hoạt động hơi khác so với Method Syntax. Nó đòi hỏi trình tự bên ngoài, trình tự bên trong, bộ chọn khóa và bộ chọn kết quả. Từ khóa ‘ON’ được sử dụng cho bộ chọn khóa. Trong đó phía bên trái của toán tử 'equals' là outsKeySelector và bên phải của 'equals' là InternalKeySelector.

from ... in outerSequence

join ... in innerSequence

on outerKey equals innerKey

select ...

Ví dụ sau đây về toán tử Join trong Query Syntax trả về một tập hợp các phần tử từ studentList và standardList. Nếu Student.StandardID và Standard.StandardID là phù hợp.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 13, StandardID =1 },

new Student() { StudentID = 2, StudentName = "Moin", Age = 21, StandardID =1 },

new Student() { StudentID = 3, StudentName = "Bill", Age = 18, StandardID =2 },

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20, StandardID =2 },

new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }

};

IList<Standard> standardList = new List<Standard>() {

new Standard(){ StandardID = 1, StandardName="Standard 1"},

new Standard(){ StandardID = 2, StandardName="Standard 2"},

new Standard(){ StandardID = 3, StandardName="Standard 3"}

};

var innerJoin = from s in studentList // outer sequence

join st in standardList //inner sequence

on s.StandardID equals st.StandardID // key selector

select new { // result selector

StudentName = s.StudentName,

StandardName = st.StandardName

};

**VB.NET**

Dim innerJoin = From s In studentList ' outer sequence

Join std In standardList ' inner sequence

On s.StandardID Equals std.StandardID ' key selector

Select \_ ' result selector

StudentName = s.StudentName,

StandardName = std.StandardName

**Output**

John - Standard 1  
Moin - Standard 1  
Bill - Standard 2  
Ram - Standard 2

Lưu ý: Sử dụng toán tử equals để so khớp với bộ chọn khóa trong Query Syntax . Toán tử == không hợp lệ.

**Những điểm cần nhớ**

* Join và GroupJoin là các toán tử kết.
* Join giống như sự inner join của SQL. Nó trả về một collection mới chứa các thành phần chung từ hai collection khớp với khóa.
* Join hoạt động trên hai chuỗi bên trong và bên ngoài và tạo ra một chuỗi kết quả.
* Cú pháp Join query Syntax:

From … in OuterSequence

Join … in InnerSequence

On outerkey equals innerkey

Select …

**Toán tử kết – GroupJoin**

Toán tử GroupJoin thực hiện nhiệm vụ tương tự như toán tử Join ngoại trừ GroupJoin trả về một kết quả trong nhóm dựa trên khóa nhóm được chỉ định. Toán tử GroupJoin kết hai chuỗi dựa trên khóa và nhóm kết quả bằng cách khớp khóa và sau đó trả về tập hợp kết quả và khóa được nhóm.

GroupJoin yêu cầu các tham số tương tự như Join. GroupJoin có hai overload như sau:

public static IEnumerable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector, Func<TOuter, IEnumerable<TInner>, TResult> resultSelector);

public static IEnumerable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(this IEnumerable<TOuter> outer, IEnumerable<TInner> inner, Func<TOuter, TKey> outerKeySelector, Func<TInner, TKey> innerKeySelector, Func<TOuter, IEnumerable<TInner>, TResult> resultSelector, IEqualityComparer<TKey> comparer);

Như bạn có thể thấy trong overload đầu tiên có năm tham số đầu vào (ngoại trừ tham số 'this' đầu tiên): 1) outer 2) inner 3) outerKeySelector 4) innerKeySelector 5) resultSelector. Xin lưu ý rằng resultSelector thuộc Func delegate có tham số đầu vào thứ hai là loại IEnumerable cho chuỗi bên trong.

Bây giờ, hãy hiểu GroupJoin bằng cách sử dụng lớp Student và Standard trong đó Lớp Student bao gồm StandardID phù hợp với StandardID của lớp Standard.

public class Student{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int StandardID { get; set; }

}

public class Standard{

public int StandardID { get; set; }

public string StandardName { get; set; }

}

Hãy xem xét ví dụ truy vấn GroupJoin sau đây

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", StandardID =1 },

new Student() { StudentID = 2, StudentName = "Moin", StandardID =1 },

new Student() { StudentID = 3, StudentName = "Bill", StandardID =2 },

new Student() { StudentID = 4, StudentName = "Ram", StandardID =2 },

new Student() { StudentID = 5, StudentName = "Ron" }

};

IList<Standard> standardList = new List<Standard>() {

new Standard(){ StandardID = 1, StandardName="Standard 1"},

new Standard(){ StandardID = 2, StandardName="Standard 2"},

new Standard(){ StandardID = 3, StandardName="Standard 3"}

};

var groupJoin = standardList.GroupJoin(studentList, //inner sequence

std => std.StandardID, //outerKeySelector

s => s.StandardID, //innerKeySelector

(std, studentsGroup) => new // resultSelector

{

Students = studentsGroup,

StandarFulldName = std.StandardName

});

foreach (var item in groupJoin)

{

Console.WriteLine(item.StandarFulldName );

foreach(var stud in item.Students)

Console.WriteLine(stud.StudentName);

}

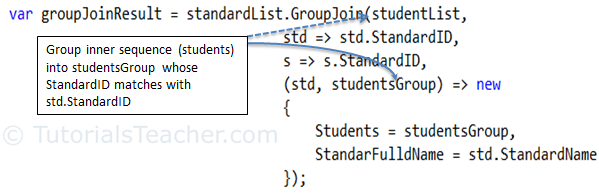
**Output**

Standard 1:   
John,  
Moin,  
Standard 2:  
Bill,  
Ram,  
Standard 3:

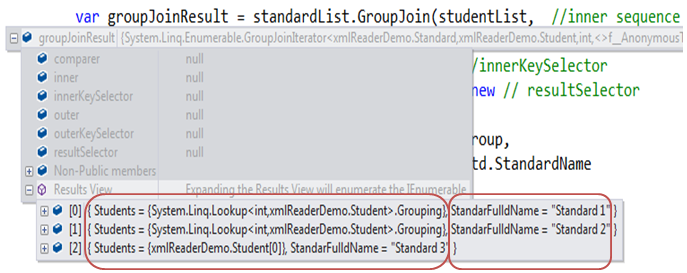
Trong ví dụ trên về truy vấn GroupJoin. standardList là chuỗi bên ngoài, bởi vì truy vấn bắt đầu từ nó. Tham số đầu tiên trong phương thức GroupJoin là chỉ định chuỗi bên trong, là studentList trong ví dụ trên. Các tham số thứ hai và thứ ba của phương thức GroupJoin () là chỉ định một trường có giá trị phải được khớp bằng biểu thức lambda, để bao gồm các phần tử trong kết quả. Bộ chọn khóa cho chuỗi bên ngoài standard => standard.StandardID chỉ ra rằng trường StandardID của từng thành phần trong StandardList phải khớp với khóa của chuỗi bên trong studentList student => student.StandardID. Nếu giá trị của cả hai trường khóa được khớp thì đưa phần tử đó vào nhóm sinh viên tập hợp được nhóm trong đó khóa sẽ là StandardID.

Tham số cuối cùng trong phương thức Join là một biểu thức để tạo kết quả. Trong ví dụ trên, bộ chọn kết quả bao gồm collection studentgroup và StandardName được nhóm.

Hình ảnh sau đây minh họa rằng chuỗi bên trong được nhóm vào collection của student để khớp với khóa StandardID và collection được nhóm đó có thể được sử dụng để tạo kết quả.



Resultset sẽ bao gồm một đối tượng ẩn danh có các thuộc tính Student và StandardFullName. Student sẽ là collection của các student có StandardID khớp với Standard.StandardID.



Bạn có thể truy cập kết quả bằng vòng lặp 'foreach'. Mỗi phần tử sẽ có thuộc tính StandardFullName & Student, trong đó Student sẽ là một collection.

**C#**

foreach (var item in groupJoinResult)

{

Console.WriteLine(item.StandarFulldName );

foreach(var stud in item.Students)

Console.WriteLine(stud.StudentName);

}

**VB.NET**

Dim groupJoin = standardList.GroupJoin( ' outer sequence

studentList, ' inner sequence

Function(s) s.StandardID, ' outerKeySelector

Function(stud) stud.StandardID, ' innerKeySelector

Function(s, studentGroup) New With { ' result selector

.students = studentGroup,

.standardName = s.StandardName

})

For Each item In groupJoin

Console.WriteLine(item.standardName)

For Each std In item.students

Console.WriteLine( std.StudentName)

Next

Next

**Output**

Standard 1:   
John,  
Moin,  
Standard 2:  
Bill,  
Ram,  
Standard 3:

**Group Join trong Query Syntax**

Toán tử GroupJoin trong Query Syntax hoạt động hơi khác so với Method Syntax. Nó đòi hỏi một chuỗi bên ngoài, chuỗi bên trong, bộ chọn khóa và bộ chọn kết quả. Từ khóa ‘ON’ được sử dụng cho bộ chọn khóa trong đó phía bên trái của toán tử equals là outerKeySelector và phía bên phải của equals là innerKeySelector. Sử dụng từ khóa into để tạo collection được nhóm.

from ... in outerSequence

join ... in innerSequence

on outerKey equals innerKey

into groupedCollection

select ...

Ví dụ sau đây cho thấy GroupJoin trong Query Syntax

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 13, StandardID =1 },

new Student() { StudentID = 2, StudentName = "Moin", Age = 21, StandardID =1 },

new Student() { StudentID = 3, StudentName = "Bill", Age = 18, StandardID =2 },

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20, StandardID =2 },

new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }

};

IList<Standard> standardList = new List<Standard>() {

new Standard(){ StandardID = 1, StandardName="Standard 1"},

new Standard(){ StandardID = 2, StandardName="Standard 2"},

new Standard(){ StandardID = 3, StandardName="Standard 3"}

};

var groupJoin = from std in standardList

join s in studentList

on std.StandardID equals s.StandardID

into studentGroup

select new {

Students = studentGroup ,

StandardName = std.StandardName

};

foreach (var item in groupJoin)

{

Console.WriteLine(item.StandarFulldName );

foreach(var stud in item.Students)

Console.WriteLine(stud.StudentName);

}

**VB.NET**

Dim groupJoin = From s In standardList

Group Join stud In studentList

On stud.StandardID Equals s.StandardID

Into Group \_

Select \_

StudentsGroup = Group,

StandardName = s.StandardName

**Output**

Standard 1:   
John,  
Moin,  
Standard 2:  
Bill,  
Ram,  
Standard 3:

Lưu ý: Sử dụng toán tử equals để khớp với bộ chọn khóa. Toán tử == không hợp lệ.

**Toán tử chiếu – Select & SelectMany**

**Select**

Toán tử Select luôn trả về một collection IEnumerable chứa các phần tử dựa trên hàm biến đổi. Nó tương tự như mệnh đề Select của SQL tạo ra tập kết quả phẳng.

Bây giờ, hãy hiểu toán tử select truy vấn bằng cách sử dụng lớp Student sau.

public class Student{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

**Select trong Query Syntax**

Cú pháp truy vấn LINQ phải kết thúc bằng mệnh đề Select hoặc GroupBy. Ví dụ sau đây cho thấy toán tử select trả về một collection string StudentName.

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John" },

new Student() { StudentID = 2, StudentName = "Moin" },

new Student() { StudentID = 3, StudentName = "Bill" },

new Student() { StudentID = 4, StudentName = "Ram" },

new Student() { StudentID = 5, StudentName = "Ron" }

};

var selectResult = from s in studentList

select s.StudentName;

Toán tử chọn có thể được sử dụng để định dạng kết quả theo yêu cầu của chúng tôi. Nó có thể được sử dụng để trả về một collection các lớp tùy chỉnh hoặc loại ẩn danh bao gồm các thuộc tính theo nhu cầu của chúng tôi.

Ví dụ sau đây của mệnh đề select trả về một collection của các loại ẩn danh chứa thuộc tính Tên và Tuổi.

**C#**

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 13 } ,

new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 15 }

};

// returns collection of anonymous objects with Name and Age property

var selectResult = from s in studentList

select new { Name = "Mr. " + s.StudentName, Age = s.Age };

// iterate selectResult

foreach (var item in selectResult)

Console.WriteLine("Student Name: {0}, Age: {1}", item.Name, item.Age);

**VB.NET**

Dim selectResult = From s In studentList

Select New With {.Name = s.StudentName, .Age = s.Age}

**Output**

Student Name: Mr. John, Age: 13   
Student Name: Mr. Moin, Age: 21  
Student Name: Mr. Bill, Age: 18  
Student Name: Mr. Ram, Age: 20  
Student Name: Mr. Ron, Age: 15

**Select trong Method Syntax**

Toán tử select là tùy chọn trong Method syntax. Tuy nhiên, bạn có thể sử dụng nó để định hình dữ liệu. Trong ví dụ sau, phương thức mở rộng select trả về một collection các đối tượng ẩn danh với thuộc tính Tên và Tuổi:

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Moin", Age = 21 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

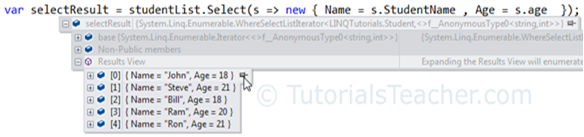
new Student() { StudentID = 5, StudentName = "Ron" , Age = 21 }

};

var selectResult = studentList.Select(s => new { Name = s.StudentName ,

Age = s.Age });

Trong ví dụ trên, selectResult sẽ chứa các đối tượng ẩn danh với thuộc tính Name và Age như hiển thị bên dưới trong chế độ debug.



**VB.NET**

Dim selectResult = studentList.Select(Function(s) New With {.Name = s.StudentName,

.Age = s.Age})