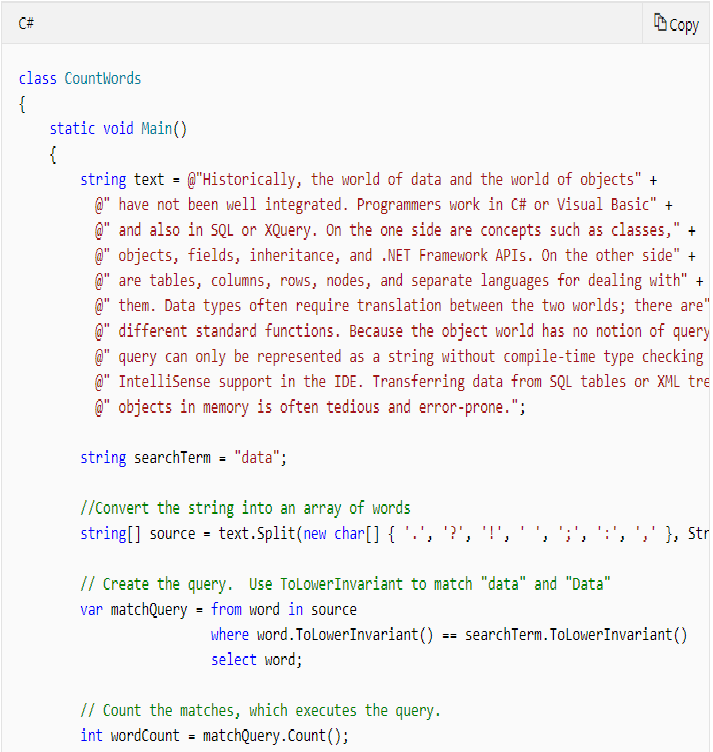
Thuật ngữ "LINQ to Objects" đề cập đến việc sử dụng các truy vấn LINQ với bất  
kỳ IEnumerable hay IEnumerable <T>, mà không cần sử dụng một nhà cung cấp hay API như LINQ to SQL hay LINQ to XML. Bạn có thể sử dụng LINQ để truy vấn bất  
kỳ bộ tập hợp enumerable như: List<T>, Array, hoặc Dictionary<TKey, TValue>. Các tập hợp có thể được người dùng định nghĩa hoặc có thể được return một. NET Framework API.  
Trong một hướng cơ bản, LINQ to Objects đại diện cho một phương pháp tiếp  
cận mới. Trong cách cũ, bạn phải viết vòng lặp foreach phức tạp theo lý  
thuyết để xác định rằng làm thế nào truy xuất dữ liệu từ một tập hợp. Trong LINQ đưa ra  
cách tiếp cận mới, bạn viết code những gì bạn muốn truy xuất.  
Ngoài ra, các truy vấn LINQ cung cấp ba sự tiện lợi hơn các vòng lặp foreach truyền  
thống:  
1. Chúng ngắn gọn và dễ đọc, đặc biệt là khi có nhiều điều kiện lọc.  
2. Chúng cung cấp bộ lọc mạnh mẽ, sắp xếp, và khả năng gom nhóm với đoạn mã ứng  
dụng nhỏ nhất.  
3. Chúng có thể được chuyển đến các nguồn dữ liệu khác với một vài hoặc không có sửa  
đổi, bổ sung.  
Nhìn chung, các hoạt động phức tạp hơn mà bạn muốn thực hiện trên cơ sở dữ  
liệu, các bạn sẽ thấy rõ hơn lợi ích bằng cách sử dụng LINQ thay vì kỹ thuật lặp truyền  
thống.  
 **LINQ and Strings**LINQ có thể được sử dụng để truy vấn và thay đổi những chỗi và tập tập của  
những chuỗi. Nó đặc biệt hữu ích với cấu trúc dữ liệu trong file văn bản. Các truy vấn  
LINQ có thể được kết hợp với các hàm và các biểu thức của chuỗi bình thường. Ví dụ,  
bạn có thể sử dụng các phương thức  [String.Split](https://docs.microsoft.com/en-us/dotnet/api/system.string.split) or [Regex.Split](https://docs.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.regex.split) để tạo ra một mảng string mà bạn có thể truy vấn sau đó hoặc sửa đổi bằng cách sử dụng LINQ. Bạn có thể sử dụng các phương thức [Regex.IsMatch](https://docs.microsoft.com/en-us/dotnet/api/system.text.regularexpressions.regex.ismatch) trong mệnh đề ***where*** của một truy vấn LINQ. Và bạn có thể sử dụng LINQ để truy vấn hoặc sửa đổi *MatchCollection* các kết quả trả lại bởi một biểu thức chính quy.  
 **Cách: Đếm sự xuất hiện của một từ trong một chuỗi (LINQ)**Ví dụ này cho thấy cách sử dụng một truy vấn LINQ để đếm các xuất hiện của  
một từ trong một chuỗi. Lưu ý rằng để thực hiện việc đếm, trước tiên là gọi phương thức  
*Split* để tạo ra một mảng các từ. Ở đây là một chi phí cho sự thực thi phương thức *Split*.  
Nếu chỉ thao tác trên các chuỗi là để đếm các từ, bạn nên cân nhắc việc sử dụng các  
phương thức Matches hoặc IndexOf phù hợp để thay thế. Tuy nhiên, nếu chi phí không  
phải là một vấn đề nghiêm trọng, hoặc bạn đã phân chia các câu để thực hiện các loại truy  
vấn trên nó, thì nó làm cho cảm giác sử dụng LINQ để truy cập các từ hoặc cụm từ cũng  
như.  
** Cách: Truy vấn cho câu đó chứa một bộ từ.**Ví dụ này cho thấy như thế nào để tìm câu trong một tập tin văn bản có chứa kết  
quả phù hợp cho mỗi một bộ từ. Mặc dù các điều kiện tìm kiếm là một đoạn code cứng  
trong ví dụ này, nó cũng có thể được lấy ra tại thời gian chạy. Trong ví dụ này, các truy  
vấn sẽ trả về các câu có chứa các cụm từ "Historically", "data," và "integrated"  
**class FindSentences**

**{**

**static void Main()**

**{**

**string text = @"Historically, the world of data and the world of objects " +**

**@"have not been well integrated. Programmers work in C# or Visual Basic " +**

**@"and also in SQL or XQuery. On the one side are concepts such as classes, " +**

**@"objects, fields, inheritance, and .NET Framework APIs. On the other side " +**

**@"are tables, columns, rows, nodes, and separate languages for dealing with " +**

**@"them. Data types often require translation between the two worlds; there are " +**

**@"different standard functions. Because the object world has no notion of query, a " +**

**@"query can only be represented as a string without compile-time type checking or " +**

**@"IntelliSense support in the IDE. Transferring data from SQL tables or XML trees to " +**

**@"objects in memory is often tedious and error-prone.";**

**// Split the text block into an array of sentences.**

**string[] sentences = text.Split(new char[] { '.', '?', '!' });**

**// Define the search terms. This list could also be dynamically populated at runtime.**

**string[] wordsToMatch = { "Historically", "data", "integrated" };**

**// Find sentences that contain all the terms in the wordsToMatch array.**

**// Note that the number of terms to match is not specified at compile time.**

**var sentenceQuery = from sentence in sentences**

**let w = sentence.Split(new char[] { '.', '?', '!', ' ', ';', ':', ',' },**

**StringSplitOptions.RemoveEmptyEntries)**

**where w.Distinct().Intersect(wordsToMatch).Count() == wordsToMatch.Count()**

**select sentence;**

**// Execute the query. Note that you can explicitly type**

**// the iteration variable here even though sentenceQuery**

**// was implicitly typed.**

**foreach (string str in sentenceQuery)**

**{**

**Console.WriteLine(str);**

**}**

**// Keep the console window open in debug mode.**

**Console.WriteLine("Press any key to exit");**

**Console.ReadKey();**

**}**

**}**

**/\* Output:**

**Historically, the world of data and the world of objects have not been well integrated**

**\*/**

Các truy vấn làm việc bằng cách trước tiên phân đôi văn bản vào câu, và sau đó  
tách các câu vào một mảng có chứa những chuỗi của mỗi từ. Đối với mỗi arrays này,  
phương thức *Distinct* loại bỏ tất cả các kí tự nào bị trùng lặp, và sau đó truy vấn thực hiện  
các hoạt động trên một phân cắt từ mảng và các mảng ***wordsToMatch***. Nếu việc đếm các  
điểm giao là giống như đếm của mảng *wordsToMatch*, tất cả các từ đã được tìm thấy  
trong từ và ban đầu là câu trả về.  
Trong lúc gọi đến Split, các dấu chấm câu như dấu tách được sử dụng để loại bỏ  
chúng khỏi chuỗi. Nếu bạn đã không làm được điều này, ví dụ như bạn có thể có một  
chuỗi "Historically," rằng sẽ không phù hợp với "Historically" trong mảng  
*wordsToMatch*. Bạn có thể sử dụng để có thêm các dấu tách, tùy thuộc vào loại dấu chấm  
câu được tìm thấy trong các nguồn văn bản.

**Cách: Truy vấn cho các ký tự trong một String (LINQ)**Bởi vì trong thực hiện các lớp String có chung interface IEnumerable <T>, bất kỳ  
chuỗi có thể được truy vấn như là một chuỗi các ký tự. Tuy nhiên, điều này không phải là  
một cách sử dụng chung của LINQ.  
Ví dụ sau truy vấn một chuỗi để xác định số lượng số chữ số chứa trong nó. Lưu ý  
rằng các truy vấn là "reused" sau khi được thực hiện lần đầu tiên. Điều này có thể vì các  
truy vấn tự nó không lưu trữ bất kỳ kết quả thực sự.  
**class QueryAString**

**{**

**static void Main()**

**{**

**string aString = "ABCDE99F-J74-12-89A";**

**// Select only those characters that are numbers**

**IEnumerable<char> stringQuery =**

**from ch in aString**

**where Char.IsDigit(ch)**

**select ch;**

**// Execute the query**

**foreach (char c in stringQuery)**

**Console.Write(c + " ");**

**// Call the Count method on the existing query.**

**int count = stringQuery.Count();**

**Console.WriteLine("Count = {0}", count);**

**// Select all characters before the first '-'**

**IEnumerable<char> stringQuery2 = aString.TakeWhile(c => c != '-');**

**// Execute the second query**

**foreach (char c in stringQuery2)**

**Console.Write(c);**

**Console.WriteLine(System.Environment.NewLine + "Press any key to exit");**

**Console.ReadKey();**

**}**

**}**

**/\* Output:**

**Output: 9 9 7 4 1 2 8 9**

**Count = 8**

**ABCDE99F**

**\*/**

**Cách: Kết hợp LINQ truy vấn với các biểu thức chính quy.**Ví dụ này cho thấy cách sử dụng lớp Regex để tạo ra một biểu thức chính quy cho  
phù hợp hơn trong chuỗi văn bản. Các truy vấn LINQ là cách dễ dàng để lọc chính xác về  
các tập tin mà bạn muốn tìm kiếm với các biểu thức chính quy, và để hình thành các kết  
quả.  
**class QueryWithRegEx**

**{**

**public static void Main()**

**{**

**// Modify this path as necessary so that it accesses your version of Visual Studio.**

**string startFolder = @"C:\Program Files (x86)\Microsoft Visual Studio 14.0\";**

**// One of the following paths may be more appropriate on your computer.**

**//string startFolder = @"C:\Program Files (x86)\Microsoft Visual Studio\2017\";**

**// Take a snapshot of the file system.**

**IEnumerable<System.IO.FileInfo> fileList = GetFiles(startFolder);**

**// Create the regular expression to find all things "Visual".**

**System.Text.RegularExpressions.Regex searchTerm =**

**new System.Text.RegularExpressions.Regex(@"Visual (Basic|C#|C\+\+|Studio)");**

**// Search the contents of each .htm file.**

**// Remove the where clause to find even more matchedValues!**

**// This query produces a list of files where a match**

**// was found, and a list of the matchedValues in that file.**

**// Note: Explicit typing of "Match" in select clause.**

**// This is required because MatchCollection is not a**

**// generic IEnumerable collection.**

**var queryMatchingFiles =**

**from file in fileList**

**where file.Extension == ".htm"**

**let fileText = System.IO.File.ReadAllText(file.FullName)**

**let matches = searchTerm.Matches(fileText)**

**where matches.Count > 0**

**select new**

**{**

**name = file.FullName,**

**matchedValues = from System.Text.RegularExpressions.Match match in matches**

**select match.Value**

**};**

**// Execute the query.**

**Console.WriteLine("The term \"{0}\" was found in:", searchTerm.ToString());**

**foreach (var v in queryMatchingFiles)**

**{**

**// Trim the path a bit, then write**

**// the file name in which a match was found.**

**string s = v.name.Substring(startFolder.Length - 1);**

**Console.WriteLine(s);**

**// For this file, write out all the matching strings**

**foreach (var v2 in v.matchedValues)**

**{**

**Console.WriteLine(" " + v2);**

**}**

**}**

**// Keep the console window open in debug mode**

**Console.WriteLine("Press any key to exit");**

**Console.ReadKey();**

**}**

**// This method assumes that the application has discovery**

**// permissions for all folders under the specified path.**

**static IEnumerable<System.IO.FileInfo> GetFiles(string path)**

**{**

**if (!System.IO.Directory.Exists(path))**

**throw new System.IO.DirectoryNotFoundException();**

**string[] fileNames = null;**

**List<System.IO.FileInfo> files = new List<System.IO.FileInfo>();**

**fileNames = System.IO.Directory.GetFiles(path, "\*.\*", System.IO.SearchOption.AllDirectories);**

**foreach (string name in fileNames)**

**{**

**files.Add(new System.IO.FileInfo(name));**

**}**

**return files;**

**}**

**}**  
Lưu ý rằng bạn cũng có thể truy vấn đối tượng MatchCollection được trả lại bởi  
một RegEx tìm kiếm. Trong ví dụ này chỉ có giá trị của mỗi tương xứng là được trả về  
trong các kết quả. Tuy nhiên, đây cũng là để có thể sử dụng LINQ để thực hiện tất cả các  
loại lọc, phân loại, và các nhóm trên tập hợp. Bởi vì MatchCollection là một tập hợp  
không có kiểu chung IEnumerable, bạn cần phải rõ ràng trạng tái trong phạm vi của các  
loại biến trong truy vấn.  
**VIII.7 Câu hỏi bán cấu trúc dữ liệu ở định dạng văn bản**Nhiều loại khác nhau của các file văn bản bao gồm một loạt các dòng, thường  
xuyên với các định dạng tương tự, chẳng hạn như tab hay dấu phẩy phân các tập tin hoặc  
cố định-chiều dài dòng. Sau khi bạn đọc như một tập tin văn bản vào bộ nhớ, bạn có thể  
*Đồ án kỹ sư II Tìm hiểu công nghệ LINQ và ứng dụng  
Sinh viên thực hiện Nguyễn Văn Thụy & Hoàng Mạnh Giỏi  
Trang* 105  
sử dụng LINQ để truy vấn và / hoặc sửa đổi dòng. Các truy vấn LINQ cũng đơn giản hóa  
các nhiệm vụ, kết hợp dữ liệu từ nhiều nguồn  
 **Cách: Tìm các tập khác biệt giữa hai List (LINQ)**Ví dụ này cho thấy như thế nào để sử dụng LINQ để so sánh hai List của  
những chuỗi và những dòng có trong names1.txt nhưng không có trong names2.txt.  
**class CompareLists**

**{**

**static void Main()**

**{**

**// Create the IEnumerable data sources.**

**string[] names1 = System.IO.File.ReadAllLines(@"../../../names1.txt");**

**string[] names2 = System.IO.File.ReadAllLines(@"../../../names2.txt");**

**// Create the query. Note that method syntax must be used here.**

**IEnumerable<string> differenceQuery =**

**names1.Except(names2);**

**// Execute the query.**

**Console.WriteLine("The following lines are in names1.txt but not names2.txt");**

**foreach (string s in differenceQuery)**

**Console.WriteLine(s);**

**// Keep the console window open in debug mode.**

**Console.WriteLine("Press any key to exit");**

**Console.ReadKey();**

**}**

**}**

**/\* Output:**

**The following lines are in names1.txt but not names2.txt**

**Potra, Cristina**

**Noriega, Fabricio**

**Aw, Kam Foo**

**Toyoshima, Tim**

**Guy, Wey Yuan**

**Garcia, Debra**

**\*/**

**Cách: Sắp xếp hay Lọc dữ liệu Văn bản bởi bất kì một từ hoặc  
một trường (LINQ)**Ví dụ sau cho thấy Cách phân loại cấu trúc của dòng văn bản, chẳng hạn  
như giá trị phân cách bằng dấu phẩy, bởi bất kỳ trường nào trong dòng. Các trường có thể  
được xác định theo kiểu động tại thời gian chạy. Giả định rằng các trường trong  
scores.csv của sinh viên đại diện cho một số ID, tiếp theo là một loạt các bài kiểm tra bốn  
điểm số.  
**public class SortLines**

**{**

**static void Main()**

**{**

**// Create an IEnumerable data source**

**string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");**

**// Change this to any value from 0 to 4.**

**int sortField = 1;**

**Console.WriteLine("Sorted highest to lowest by field [{0}]:", sortField);**

**// Demonstrates how to return query from a method.**

**// The query is executed here.**

**foreach (string str in RunQuery(scores, sortField))**

**{**

**Console.WriteLine(str);**

**}**

**// Keep the console window open in debug mode.**

**Console.WriteLine("Press any key to exit");**

**Console.ReadKey();**

**}**

**// Returns the query variable, not query results!**

**static IEnumerable<string> RunQuery(IEnumerable<string> source, int num)**

**{**

**// Split the string and sort on field[num]**

**var scoreQuery = from line in source**

**let fields = line.Split(',')**

**orderby fields[num] descending**

**select line;**

**return scoreQuery;**

**}**

**}**

**/\* Output (if sortField == 1):**

**Sorted highest to lowest by field [1]:**

**116, 99, 86, 90, 94**

**120, 99, 82, 81, 79**

**111, 97, 92, 81, 60**

**114, 97, 89, 85, 82**

**121, 96, 85, 91, 60**

**122, 94, 92, 91, 91**

**117, 93, 92, 80, 87**

**118, 92, 90, 83, 78**

**113, 88, 94, 65, 91**

**112, 75, 84, 91, 39**

**119, 68, 79, 88, 92**

**115, 35, 72, 91, 70**

**\*/** }  
 **Cách: Sắp xếp lại các trường được giới hạn trong file.**Một giá trị phân cách bằng dấu phẩy (CSV) file là một tập tin văn bản nó thường  
được sử dụng để lưu trữ dữ liệu bảng tính hay xếp như bảng các dữ liệu được đại diện bởi  
các hàng và cột. Bằng việc sử dụng phương thức Split để phân cách các lĩnh vực, nó là  
rất dễ dàng để truy vấn và thao tác các file bằng cách sử dụng LINQ. Trong thực tế, cùng  
một kỹ thuật có thể được sử dụng để sắp xếp lại phần nào cấu trúc của dòng văn bản; nó  
không phải là giới hạn đối với các tập tin CSV.  
Trong ví dụ sau, giả định rằng ba cột đại diện của học sinh "last name", "first  
name", và "ID”. Các trường đang có theo thứ tự chữ cái dựa trên tên của mỗi học sinh.  
Các truy vấn tạo ra một dãy mới, trong đó các cột ID xuất hiện trước, theo sau là một cột  
thứ hai là kết hợp hai phần last name và first name của học sinh. Những dòng đã được  
sắp xếp theo ID. Các kết quả được lưu vào một tập tin mới và các dữ liệu ban đầu là  
không sửa đổi.

class CSVFiles

{

static void Main(string[] args)

{

// Create the IEnumerable data source

string[] lines = System.IO.File.ReadAllLines(@"../../../spreadsheet1.csv");

// Create the query. Put field 2 first, then

// reverse and combine fields 0 and 1 from the old field

IEnumerable<string> query =

from line in lines

let x = line.Split(',')

orderby x[2]

select x[2] + ", " + (x[1] + " " + x[0]);

// Execute the query and write out the new file. Note that WriteAllLines

// takes a string[], so ToArray is called on the query.

System.IO.File.WriteAllLines(@"../../../spreadsheet2.csv", query.ToArray());

Console.WriteLine("Spreadsheet2.csv written to disk. Press any key to exit");

Console.ReadKey();

}

}

/\* Output to spreadsheet2.csv:

111, Svetlana Omelchenko

112, Claire O'Donnell

113, Sven Mortensen

114, Cesar Garcia

115, Debra Garcia

116, Fadi Fakhouri

117, Hanying Feng

118, Hugo Garcia

119, Lance Tucker

120, Terry Adams

121, Eugene Zabokritski

122, Michael Tucker

\*/

**Cách: Kết hợp và so sánh các tập hợp chuỗi (LINQ)**Ví dụ này cho thấy như thế nào để trộn file có chứa dòng văn bản và sau đó phân  
loại các kết quả. Cụ thể, nó cho thấy như thế nào để thực hiện một cách dễ dàng để ghép,  
hợp trên hai bộ dòng văn bản.  
**class MergeStrings**

**{**

**static void Main(string[] args)**

**{**

**//Put text files in your solution folder**

**string[] fileA = System.IO.File.ReadAllLines(@"../../../names1.txt");**

**string[] fileB = System.IO.File.ReadAllLines(@"../../../names2.txt");**

**//Simple concatenation and sort. Duplicates are preserved.**

**IEnumerable<string> concatQuery =**

**fileA.Concat(fileB).OrderBy(s => s);**

**// Pass the query variable to another function for execution.**

**OutputQueryResults(concatQuery, "Simple concatenate and sort. Duplicates are preserved:");**

**// Concatenate and remove duplicate names based on**

**// default string comparer.**

**IEnumerable<string> uniqueNamesQuery =**

**fileA.Union(fileB).OrderBy(s => s);**

**OutputQueryResults(uniqueNamesQuery, "Union removes duplicate names:");**

**// Find the names that occur in both files (based on**

**// default string comparer).**

**IEnumerable<string> commonNamesQuery =**

**fileA.Intersect(fileB);**

**OutputQueryResults(commonNamesQuery, "Merge based on intersect:");**

**// Find the matching fields in each list. Merge the two**

**// results by using Concat, and then**

**// sort using the default string comparer.**

**string nameMatch = "Garcia";**

**IEnumerable<String> tempQuery1 =**

**from name in fileA**

**let n = name.Split(',')**

**where n[0] == nameMatch**

**select name;**

**IEnumerable<string> tempQuery2 =**

**from name2 in fileB**

**let n2 = name2.Split(',')**

**where n2[0] == nameMatch**

**select name2;**

**IEnumerable<string> nameMatchQuery =**

**tempQuery1.Concat(tempQuery2).OrderBy(s => s);**

**OutputQueryResults(nameMatchQuery, $"Concat based on partial name match \"{nameMatch}\":");**

**// Keep the console window open in debug mode.**

**Console.WriteLine("Press any key to exit");**

**Console.ReadKey();**

**}**

**static void OutputQueryResults(IEnumerable<string> query, string message)**

**{**

**Console.WriteLine(System.Environment.NewLine + message);**

**foreach (string item in query)**

**{**

**Console.WriteLine(item);**

**}**

**Console.WriteLine("{0} total names in list", query.Count());**

**}**

**}**

**/\* Output:**

**Simple concatenate and sort. Duplicates are preserved:**

**Aw, Kam Foo**

**Bankov, Peter**

**Bankov, Peter**

**Beebe, Ann**

**Beebe, Ann**

**El Yassir, Mehdi**

**Garcia, Debra**

**Garcia, Hugo**

**Garcia, Hugo**

**Giakoumakis, Leo**

**Gilchrist, Beth**

**Guy, Wey Yuan**

**Holm, Michael**

**Holm, Michael**

**Liu, Jinghao**

**McLin, Nkenge**

**Myrcha, Jacek**

**Noriega, Fabricio**

**Potra, Cristina**

**Toyoshima, Tim**

**20 total names in list**

**Union removes duplicate names:**

**Aw, Kam Foo**

**Bankov, Peter**

**Beebe, Ann**

**El Yassir, Mehdi**

**Garcia, Debra**

**Garcia, Hugo**

**Giakoumakis, Leo**

**Gilchrist, Beth**

**Guy, Wey Yuan**

**Holm, Michael**

**Liu, Jinghao**

**McLin, Nkenge**

**Myrcha, Jacek**

**Noriega, Fabricio**

**Potra, Cristina**

**Toyoshima, Tim**

**16 total names in list**

**Merge based on intersect:**

**Bankov, Peter**

**Holm, Michael**

**Garcia, Hugo**

**Beebe, Ann**

**4 total names in list**

**Concat based on partial name match "Garcia":**

**Garcia, Debra**

**Garcia, Hugo**

**Garcia, Hugo**

**3 total names in list**

**\*/**

**Cách: Lấy ra tập hợp đối tượng từ nhiều nguồn (LINQ)**Ví dụ này cho thấy như thế nào để trộn dữ liệu từ các loại nguồn khác nhau vào  
một chuỗi các loại mới. Các ví dụ trong các mã sau đây sẽ trộn các chuỗi với những  
mảng số nguyên. Tuy nhiên, trong cùng một nguyên tắc áp dụng cho bất kỳ hai nguồn dữ  
liệu, bao gồm cả bất kỳ sự kết hợp của các đối tượng trong bộ nhớ.  
**Ví dụ:** Ví dụ sau cho thấy cách sử dụng một tên kiểu Student để lưu trữ dữ liệu từ hai bộ  
nhớ trong tập hợp của những chuỗi mà mô phỏng dữ liệu trong bảng tính. Trước tiên tạp  
hợp của các chuỗi miêu tả các tên và ID, và tiếp theo tập hợp miêu tả ID của học  
sinh(trong cột đầu tiên).  
using System;

using System.Collections.Generic;

using System.Linq;

class Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int ID { get; set; }

public List<int> ExamScores { get; set; }

}

class PopulateCollection

{

static void Main()

{

// These data files are defined in How to: Join Content from

// Dissimilar Files (LINQ).

// Each line of names.csv consists of a last name, a first name, and an

// ID number, separated by commas. For example, Omelchenko,Svetlana,111

string[] names = System.IO.File.ReadAllLines(@"../../../names.csv");

// Each line of scores.csv consists of an ID number and four test

// scores, separated by commas. For example, 111, 97, 92, 81, 60

string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");

// Merge the data sources using a named type.

// var could be used instead of an explicit type. Note the dynamic

// creation of a list of ints for the ExamScores member. The first item

// is skipped in the split string because it is the student ID,

// not an exam score.

IEnumerable<Student> queryNamesScores =

from nameLine in names

let splitName = nameLine.Split(',')

from scoreLine in scores

let splitScoreLine = scoreLine.Split(',')

where Convert.ToInt32(splitName[2]) == Convert.ToInt32(splitScoreLine[0])

select new Student()

{

FirstName = splitName[0],

LastName = splitName[1],

ID = Convert.ToInt32(splitName[2]),

ExamScores = (from scoreAsText in splitScoreLine.Skip(1)

select Convert.ToInt32(scoreAsText)).

ToList()

};

// Optional. Store the newly created student objects in memory

// for faster access in future queries. This could be useful with

// very large data files.

List<Student> students = queryNamesScores.ToList();

// Display each student's name and exam score average.

foreach (var student in students)

{

Console.WriteLine("The average score of {0} {1} is {2}.",

student.FirstName, student.LastName,

student.ExamScores.Average());

}

//Keep console window open in debug mode

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

/\* Output:

The average score of Omelchenko Svetlana is 82.5.

The average score of O'Donnell Claire is 72.25.

The average score of Mortensen Sven is 84.5.

The average score of Garcia Cesar is 88.25.

The average score of Garcia Debra is 67.

The average score of Fakhouri Fadi is 92.25.

The average score of Feng Hanying is 88.

The average score of Garcia Hugo is 85.75.

The average score of Tucker Lance is 81.75.

The average score of Adams Terry is 85.25.

The average score of Zabokritski Eugene is 83.

The average score of Tucker Michael is 92.

\*/

**Cách: Gia nhập nội dung từ các file không cùng dạng.**Ví dụ này cho thấy như thế nào để tham gia dữ liệu từ hai dấu phẩy phân chia tập  
tin rằng một trong các giá trị được sử dụng như một chìa khóa phù hợp. Kỹ thuật này có  
thể là hữu ích nếu bạn có kết hợp dữ liệu từ hai bảng tính, hay dữ liệu từ một bảng tính và  
một tập tin có định dạng khác, vào một tập tin mới. Bạn cũng có thể sửa đổi ví dụ này để  
làm việc với bất kỳ hình thức nào về cấu trúc văn bản.  
**class** JoinStrings  
{  
**static void** Main()  
{  
string[] names = System.IO.File.ReadAllLines(@"../../../names.csv");  
string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");  
IEnumerable<string> scoreQuery1 =  
**from** name **in** names  
**let** nameFields = name.Split(',')  
**from** id **in** scores  
**let** scoreFields = id.Split(',')  
**where** nameFields[2] == scoreFields[0]  
**select** nameFields[0] + "," + scoreFields[1] + "," + scoreFields[2]  
+ "," + scoreFields[3] + "," + scoreFields[4];  
OutputQueryResults(scoreQuery1, "Merge two spreadsheets:");  
Console.WriteLine("Press any key to exit");  
Console.ReadKey();  
}  
**static void** OutputQueryResults(IEnumerable<string> query, string message)  
{  
Console.WriteLine(System.Environment.NewLine + message);  
**foreach** (string item **in** query)  
{  
Console.WriteLine(item);  
}  
Console.WriteLine("{0} total names in list", query.Count());  
}  
}  
**VIII.8.4 Cách: Tách một file vào các file bằng cách sử dụng Groups  
(LINQ)**Ví dụ sau cho thấy công việc đó.  
**class SplitWithGroups**

**{**

**static void Main()**

**{**

**string[] fileA = System.IO.File.ReadAllLines(@"../../../names1.txt");**

**string[] fileB = System.IO.File.ReadAllLines(@"../../../names2.txt");**

**// Concatenate and remove duplicate names based on**

**// default string comparer**

**var mergeQuery = fileA.Union(fileB);**

**// Group the names by the first letter in the last name.**

**var groupQuery = from name in mergeQuery**

**let n = name.Split(',')**

**group name by n[0][0] into g**

**orderby g.Key**

**select g;**

**// Create a new file for each group that was created**

**// Note that nested foreach loops are required to access**

**// individual items with each group.**

**foreach (var g in groupQuery)**

**{**

**// Create the new file name.**

**string fileName = @"../../../testFile\_" + g.Key + ".txt";**

**// Output to display.**

**Console.WriteLine(g.Key);**

**// Write file.**

**using (System.IO.StreamWriter sw = new System.IO.StreamWriter(fileName))**

**{**

**foreach (var item in g)**

**{**

**sw.WriteLine(item);**

**// Output to console for example purposes.**

**Console.WriteLine(" {0}", item);**

**}**

**}**

**}**

**// Keep console window open in debug mode.**

**Console.WriteLine("Files have been written. Press any key to exit");**

**Console.ReadKey();**

**}**

**}**

**/\* Output:**

**A**

**Aw, Kam Foo**

**B**

**Bankov, Peter**

**Beebe, Ann**

**E**

**El Yassir, Mehdi**

**G**

**Garcia, Hugo**

**Guy, Wey Yuan**

**Garcia, Debra**

**Gilchrist, Beth**

**Giakoumakis, Leo**

**H**

**Holm, Michael**

**L**

**Liu, Jinghao**

**M**

**Myrcha, Jacek**

**McLin, Nkenge**

**N**

**Noriega, Fabricio**

**P**

**Potra, Cristina**

**T**

**Toyoshima, Tim**

**\*/**

**VIII.8.5 Cách: Tính toán giá trị của cột trong một văn bản của tệp CSV  
(LINQ)**Ví dụ này cho thấy như thế nào để thực hiện tổng hợp thao tác tính toán như Sum,  
Average, Min, Max và trên các cột của một file. Csv. Ví dụ nguyên tắc được hiển thị ở  
đây có thể được áp dụng cho các loại cấu trúc văn bản.

class SumColumns

{

static void Main(string[] args)

{

string[] lines = System.IO.File.ReadAllLines(@"../../../scores.csv");

// Specifies the column to compute.

int exam = 3;

// Spreadsheet format:

// Student ID Exam#1 Exam#2 Exam#3 Exam#4

// 111, 97, 92, 81, 60

// Add one to exam to skip over the first column,

// which holds the student ID.

SingleColumn(lines, exam + 1);

Console.WriteLine();

MultiColumns(lines);

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

static void SingleColumn(IEnumerable<string> strs, int examNum)

{

Console.WriteLine("Single Column Query:");

// Parameter examNum specifies the column to

// run the calculations on. This value could be

// passed in dynamically at runtime.

// Variable columnQuery is an IEnumerable<int>.

// The following query performs two steps:

// 1) use Split to break each row (a string) into an array

// of strings,

// 2) convert the element at position examNum to an int

// and select it.

var columnQuery =

from line in strs

let elements = line.Split(',')

select Convert.ToInt32(elements[examNum]);

// Execute the query and cache the results to improve

// performance. This is helpful only with very large files.

var results = columnQuery.ToList();

// Perform aggregate calculations Average, Max, and

// Min on the column specified by examNum.

double average = results.Average();

int max = results.Max();

int min = results.Min();

Console.WriteLine("Exam #{0}: Average:{1:##.##} High Score:{2} Low Score:{3}",

examNum, average, max, min);

}

static void MultiColumns(IEnumerable<string> strs)

{

Console.WriteLine("Multi Column Query:");

// Create a query, multiColQuery. Explicit typing is used

// to make clear that, when executed, multiColQuery produces

// nested sequences. However, you get the same results by

// using 'var'.

// The multiColQuery query performs the following steps:

// 1) use Split to break each row (a string) into an array

// of strings,

// 2) use Skip to skip the "Student ID" column, and store the

// rest of the row in scores.

// 3) convert each score in the current row from a string to

// an int, and select that entire sequence as one row

// in the results.

IEnumerable<IEnumerable<int>> multiColQuery =

from line in strs

let elements = line.Split(',')

let scores = elements.Skip(1)

select (from str in scores

select Convert.ToInt32(str));

// Execute the query and cache the results to improve

// performance.

// ToArray could be used instead of ToList.

var results = multiColQuery.ToList();

// Find out how many columns you have in results.

int columnCount = results[0].Count();

// Perform aggregate calculations Average, Max, and

// Min on each column.

// Perform one iteration of the loop for each column

// of scores.

// You can use a for loop instead of a foreach loop

// because you already executed the multiColQuery

// query by calling ToList.

for (int column = 0; column < columnCount; column++)

{

var results2 = from row in results

select row.ElementAt(column);

double average = results2.Average();

int max = results2.Max();

int min = results2.Min();

// Add one to column because the first exam is Exam #1,

// not Exam #0.

Console.WriteLine("Exam #{0} Average: {1:##.##} High Score: {2} Low Score: {3}",

column + 1, average, max, min);

}

}

}

/\* Output:

Single Column Query:

Exam #4: Average:76.92 High Score:94 Low Score:39

Multi Column Query:

Exam #1 Average: 86.08 High Score: 99 Low Score: 35

Exam #2 Average: 86.42 High Score: 94 Low Score: 72

Exam #3 Average: 84.75 High Score: 91 Low Score: 65

Exam #4 Average: 76.92 High Score: 94 Low Score: 39

\*/

**Cách: Truy vấn Siêu dữ liệu của Hội đồng với Phản xạ (LINQ)**

Ví dụ sau đây cho thấy cách LINQ có thể được sử dụng với sự phản chiếu để truy xuất siêu dữ liệu cụ thể về các phương thức phù hợp với tiêu chí tìm kiếm được chỉ định. Trong trường hợp này, truy vấn sẽ tìm tên của tất cả các phương thức trong tập hợp trả về các kiểu có thể đếm được như mảng.

using System.Reflection;

using System.IO;

namespace LINQReflection

{

class ReflectionHowTO

{

static void Main(string[] args)

{

Assembly assembly = Assembly.Load("System.Core, Version=3.5.0.0, Culture=neutral, PublicKeyToken= b77a5c561934e089");

var pubTypesQuery = from type in assembly.GetTypes()

where type.IsPublic

from method in type.GetMethods()

where method.ReturnType.IsArray == true

|| ( method.ReturnType.GetInterface(

typeof(System.Collections.Generic.IEnumerable<>).FullName ) != null

&& method.ReturnType.FullName != "System.String" )

group method.ToString() by type.ToString();

foreach (var groupOfMethods in pubTypesQuery)

{

Console.WriteLine("Type: {0}", groupOfMethods.Key);

foreach (var method in groupOfMethods)

{

Console.WriteLine(" {0}", method);

}

}

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

}

}

**LINQ và FILE**

**Cách: Truy vấn tệp có thuộc tính hoặc tên được chỉ định**

Ví dụ này cho thấy cách tìm tất cả các tệp có phần mở rộng tên tệp được chỉ định (ví dụ ".txt") trong cây thư mục được chỉ định. Nó cũng chỉ ra cách trả về tệp mới nhất hoặc cũ nhất trong cây dựa trên thời gian tạo.

class FindFileByExtension

{

// This query will produce the full path for all .txt files

// under the specified folder including subfolders.

// It orders the list according to the file name.

static void Main()

{

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";

// Take a snapshot of the file system.

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

// This method assumes that the application has discovery permissions

// for all folders under the specified path.

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

//Create the query

IEnumerable<System.IO.FileInfo> fileQuery =

from file in fileList

where file.Extension == ".txt"

orderby file.Name

select file;

//Execute the query. This might write out a lot of files!

foreach (System.IO.FileInfo fi in fileQuery)

{

Console.WriteLine(fi.FullName);

}

// Create and execute a new query by using the previous

// query as a starting point. fileQuery is not

// executed again until the call to Last()

var newestFile =

(from file in fileQuery

orderby file.CreationTime

select new { file.FullName, file.CreationTime })

.Last();

Console.WriteLine("\r\nThe newest .txt file is {0}. Creation time: {1}",

newestFile.FullName, newestFile.CreationTime);

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

}

**Cách: Nhóm tệp theo tiện ích mở rộng (LINQ)**

Ví dụ này cho thấy cách LINQ có thể được sử dụng để thực hiện các hoạt động nhóm và sắp xếp nâng cao trên danh sách các tệp hoặc thư mục. Nó cũng chỉ ra cách xuất trang trong cửa sổ giao diện điều khiển bằng cách sử dụng các phương thức Skip và Take .

class GroupByExtension

{

// This query will sort all the files under the specified folder

// and subfolder into groups keyed by the file extension.

private static void Main()

{

// Take a snapshot of the file system.

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\Common7";

// Used in WriteLine to trim output lines.

int trimLength = startFolder.Length;

// Take a snapshot of the file system.

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

// This method assumes that the application has discovery permissions

// for all folders under the specified path.

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

// Create the query.

var queryGroupByExt =

from file in fileList

group file by file.Extension.ToLower() into fileGroup

orderby fileGroup.Key

select fileGroup;

// Display one group at a time. If the number of

// entries is greater than the number of lines

// in the console window, then page the output.

PageOutput(trimLength, queryGroupByExt);

}

// This method specifically handles group queries of FileInfo objects with string keys.

// It can be modified to work for any long listings of data. Note that explicit typing

// must be used in method signatures. The groupbyExtList parameter is a query that produces

// groups of FileInfo objects with string keys.

private static void PageOutput(int rootLength,

IEnumerable<System.Linq.IGrouping<string, System.IO.FileInfo>> groupByExtList)

{

// Flag to break out of paging loop.

bool goAgain = true;

// "3" = 1 line for extension + 1 for "Press any key" + 1 for input cursor.

int numLines = Console.WindowHeight - 3;

// Iterate through the outer collection of groups.

foreach (var filegroup in groupByExtList)

{

// Start a new extension at the top of a page.

int currentLine = 0;

// Output only as many lines of the current group as will fit in the window.

do

{

Console.Clear();

Console.WriteLine(filegroup.Key == String.Empty ? "[none]" : filegroup.Key);

// Get 'numLines' number of items starting at number 'currentLine'.

var resultPage = filegroup.Skip(currentLine).Take(numLines);

//Execute the resultPage query

foreach (var f in resultPage)

{

Console.WriteLine("\t{0}", f.FullName.Substring(rootLength));

}

// Increment the line counter.

currentLine += numLines;

// Give the user a chance to escape.

Console.WriteLine("Press any key to continue or the 'End' key to break...");

ConsoleKey key = Console.ReadKey().Key;

if (key == ConsoleKey.End)

{

goAgain = false;

break;

}

} while (currentLine < filegroup.Count());

if (goAgain == false)

break;

}

}

}

**Truy vấn tổng số byte trong một bộ thư mục**

class QuerySize

{

public static void Main()

{

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\VC#";

// Take a snapshot of the file system.

// This method assumes that the application has discovery permissions

// for all folders under the specified path.

IEnumerable<string> fileList = System.IO.Directory.GetFiles(startFolder, "\*.\*", System.IO.SearchOption.AllDirectories);

var fileQuery = from file in fileList

select GetFileLength(file);

// Cache the results to avoid multiple trips to the file system.

long[] fileLengths = fileQuery.ToArray();

// Return the size of the largest file

long largestFile = fileLengths.Max();

// Return the total number of bytes in all the files under the specified folder.

long totalBytes = fileLengths.Sum();

Console.WriteLine("There are {0} bytes in {1} files under {2}",

totalBytes, fileList.Count(), startFolder);

Console.WriteLine("The largest files is {0} bytes.", largestFile);

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

// This method is used to swallow the possible exception

// that can be raised when accessing the System.IO.FileInfo.Length property.

static long GetFileLength(string filename)

{

long retval;

try

{

System.IO.FileInfo fi = new System.IO.FileInfo(filename);

retval = fi.Length;

}

catch (System.IO.FileNotFoundException)

{

// If a file is no longer present,

// just add zero bytes to the total.

retval = 0;

}

return retval;

}

}

**So sánh nội dung của hai thư mục**

Ví dụ này cho thấy ba cách để so sánh hai danh sách tệp:

Bằng cách truy vấn giá trị Boolean xác định xem hai danh sách tệp có giống nhau không.

Bằng cách truy vấn giao lộ để lấy các tệp trong cả hai thư mục.

Bằng cách truy vấn sự khác biệt được thiết lập để truy xuất các tệp trong một thư mục nhưng không phải là thư mục khác.

namespace QueryCompareTwoDirs

{

class CompareDirs

{

static void Main(string[] args)

{

// Create two identical or different temporary folders

// on a local drive and change these file paths.

string pathA = @"C:\TestDir";

string pathB = @"C:\TestDir2";

System.IO.DirectoryInfo dir1 = new System.IO.DirectoryInfo(pathA);

System.IO.DirectoryInfo dir2 = new System.IO.DirectoryInfo(pathB);

// Take a snapshot of the file system.

IEnumerable<System.IO.FileInfo> list1 = dir1.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

IEnumerable<System.IO.FileInfo> list2 = dir2.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

//A custom file comparer defined below

FileCompare myFileCompare = new FileCompare();

// This query determines whether the two folders contain

// identical file lists, based on the custom file comparer

// that is defined in the FileCompare class.

// The query executes immediately because it returns a bool.

bool areIdentical = list1.SequenceEqual(list2, myFileCompare);

if (areIdentical == true)

{

Console.WriteLine("the two folders are the same");

}

else

{

Console.WriteLine("The two folders are not the same");

}

// Find the common files. It produces a sequence and doesn't

// execute until the foreach statement.

var queryCommonFiles = list1.Intersect(list2, myFileCompare);

if (queryCommonFiles.Count() > 0)

{

Console.WriteLine("The following files are in both folders:");

foreach (var v in queryCommonFiles)

{

Console.WriteLine(v.FullName); //shows which items end up in result list

}

}

else

{

Console.WriteLine("There are no common files in the two folders.");

}

// Find the set difference between the two folders.

// For this example we only check one way.

var queryList1Only = (from file in list1

select file).Except(list2, myFileCompare);

Console.WriteLine("The following files are in list1 but not list2:");

foreach (var v in queryList1Only)

{

Console.WriteLine(v.FullName);

}

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

// This implementation defines a very simple comparison

// between two FileInfo objects. It only compares the name

// of the files being compared and their length in bytes.

class FileCompare : System.Collections.Generic.IEqualityComparer<System.IO.FileInfo>

{

public FileCompare() { }

public bool Equals(System.IO.FileInfo f1, System.IO.FileInfo f2)

{

return (f1.Name == f2.Name &&

f1.Length == f2.Length);

}

// Return a hash that reflects the comparison criteria. According to the

// rules for IEqualityComparer<T>, if Equals is true, then the hash codes must

// also be equal. Because equality as defined here is a simple value equality, not

// reference identity, it is possible that two or more objects will produce the same

// hash code.

public int GetHashCode(System.IO.FileInfo fi)

{

string s = $"{fi.Name}{fi.Length}";

return s.GetHashCode();

}

}

}

**Truy vấn tệp hoặc những tệp lớn nhất trong cây thư mục**

Ví dụ này cho thấy năm truy vấn liên quan đến kích thước tệp theo byte:

Làm thế nào để lấy kích thước theo byte của tệp lớn nhất.

Làm thế nào để lấy kích thước theo byte của tệp nhỏ nhất.

Cách truy xuất đối tượng FileInfo tệp lớn nhất hoặc nhỏ nhất từ một hoặc nhiều thư mục trong thư mục gốc đã chỉ định.

Cách lấy một chuỗi như 10 tệp lớn nhất.

Cách sắp xếp các tệp thành các nhóm dựa trên kích thước tệp của chúng theo byte, bỏ qua các tệp nhỏ hơn kích thước đã chỉ định.

class QueryBySize

{

static void Main(string[] args)

{

QueryFilesBySize();

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

private static void QueryFilesBySize()

{

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";

// Take a snapshot of the file system.

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

// This method assumes that the application has discovery permissions

// for all folders under the specified path.

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

//Return the size of the largest file

long maxSize =

(from file in fileList

let len = GetFileLength(file)

select len)

.Max();

Console.WriteLine("The length of the largest file under {0} is {1}",

startFolder, maxSize);

// Return the FileInfo object for the largest file

// by sorting and selecting from beginning of list

System.IO.FileInfo longestFile =

(from file in fileList

let len = GetFileLength(file)

where len > 0

orderby len descending

select file)

.First();

Console.WriteLine("The largest file under {0} is {1} with a length of {2} bytes",

startFolder, longestFile.FullName, longestFile.Length);

//Return the FileInfo of the smallest file

System.IO.FileInfo smallestFile =

(from file in fileList

let len = GetFileLength(file)

where len > 0

orderby len ascending

select file).First();

Console.WriteLine("The smallest file under {0} is {1} with a length of {2} bytes",

startFolder, smallestFile.FullName, smallestFile.Length);

//Return the FileInfos for the 10 largest files

// queryTenLargest is an IEnumerable<System.IO.FileInfo>

var queryTenLargest =

(from file in fileList

let len = GetFileLength(file)

orderby len descending

select file).Take(10);

Console.WriteLine("The 10 largest files under {0} are:", startFolder);

foreach (var v in queryTenLargest)

{

Console.WriteLine("{0}: {1} bytes", v.FullName, v.Length);

}

// Group the files according to their size, leaving out

// files that are less than 200000 bytes.

var querySizeGroups =

from file in fileList

let len = GetFileLength(file)

where len > 0

group file by (len / 100000) into fileGroup

where fileGroup.Key >= 2

orderby fileGroup.Key descending

select fileGroup;

foreach (var filegroup in querySizeGroups)

{

Console.WriteLine(filegroup.Key.ToString() + "00000");

foreach (var item in filegroup)

{

Console.WriteLine("\t{0}: {1}", item.Name, item.Length);

}

}

}

// This method is used to swallow the possible exception

// that can be raised when accessing the FileInfo.Length property.

// In this particular case, it is safe to swallow the exception.

static long GetFileLength(System.IO.FileInfo fi)

{

long retval;

try

{

retval = fi.Length;

}

catch (System.IO.FileNotFoundException)

{

// If a file is no longer present,

// just add zero bytes to the total.

retval = 0;

}

return retval;

}

}

**Cách: Truy vấn các tệp trùng lặp trong Cây thư mục**

Đôi khi các tệp có cùng tên có thể được đặt trong nhiều thư mục. Ví dụ: trong thư mục cài đặt Visual Studio, một số thư mục có tệp readme.htm. Ví dụ này cho thấy cách truy vấn các tên tệp trùng lặp như vậy trong một thư mục gốc được chỉ định. Ví dụ thứ hai cho thấy cách truy vấn các tệp có kích thước và thời gian tạo cũng khớp.

class QueryDuplicateFileNames

{

static void Main(string[] args)

{

// Uncomment QueryDuplicates2 to run that query.

QueryDuplicates();

// QueryDuplicates2();

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

static void QueryDuplicates()

{

// Change the root drive or folder if necessary

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";

// Take a snapshot of the file system.

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

// This method assumes that the application has discovery permissions

// for all folders under the specified path.

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

// used in WriteLine to keep the lines shorter

int charsToSkip = startFolder.Length;

// var can be used for convenience with groups.

var queryDupNames =

from file in fileList

group file.FullName.Substring(charsToSkip) by file.Name into fileGroup

where fileGroup.Count() > 1

select fileGroup;

// Pass the query to a method that will

// output one page at a time.

PageOutput<string, string>(queryDupNames);

}

// A Group key that can be passed to a separate method.

// Override Equals and GetHashCode to define equality for the key.

// Override ToString to provide a friendly name for Key.ToString()

class PortableKey

{

public string Name { get; set; }

public DateTime CreationTime { get; set; }

public long Length { get; set; }

public override bool Equals(object obj)

{

PortableKey other = (PortableKey)obj;

return other.CreationTime == this.CreationTime &&

other.Length == this.Length &&

other.Name == this.Name;

}

public override int GetHashCode()

{

string str = $"{this.CreationTime}{this.Length}{this.Name}";

return str.GetHashCode();

}

public override string ToString()

{

return $"{this.Name} {this.Length} {this.CreationTime}";

}

}

static void QueryDuplicates2()

{

// Change the root drive or folder if necessary.

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\Common7";

// Make the lines shorter for the console display

int charsToSkip = startFolder.Length;

// Take a snapshot of the file system.

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

// Note the use of a compound key. Files that match

// all three properties belong to the same group.

// A named type is used to enable the query to be

// passed to another method. Anonymous types can also be used

// for composite keys but cannot be passed across method boundaries

//

var queryDupFiles =

from file in fileList

group file.FullName.Substring(charsToSkip) by

new PortableKey { Name = file.Name, CreationTime = file.CreationTime, Length = file.Length } into fileGroup

where fileGroup.Count() > 1

select fileGroup;

var list = queryDupFiles.ToList();

int i = queryDupFiles.Count();

PageOutput<PortableKey, string>(queryDupFiles);

}

// A generic method to page the output of the QueryDuplications methods

// Here the type of the group must be specified explicitly. "var" cannot

// be used in method signatures. This method does not display more than one

// group per page.

private static void PageOutput<K, V>(IEnumerable<System.Linq.IGrouping<K, V>> groupByExtList)

{

// Flag to break out of paging loop.

bool goAgain = true;

// "3" = 1 line for extension + 1 for "Press any key" + 1 for input cursor.

int numLines = Console.WindowHeight - 3;

// Iterate through the outer collection of groups.

foreach (var filegroup in groupByExtList)

{

// Start a new extension at the top of a page.

int currentLine = 0;

// Output only as many lines of the current group as will fit in the window.

do

{

Console.Clear();

Console.WriteLine("Filename = {0}", filegroup.Key.ToString() == String.Empty ? "[none]" : filegroup.Key.ToString());

// Get 'numLines' number of items starting at number 'currentLine'.

var resultPage = filegroup.Skip(currentLine).Take(numLines);

//Execute the resultPage query

foreach (var fileName in resultPage)

{

Console.WriteLine("\t{0}", fileName);

}

// Increment the line counter.

currentLine += numLines;

// Give the user a chance to escape.

Console.WriteLine("Press any key to continue or the 'End' key to break...");

ConsoleKey key = Console.ReadKey().Key;

if (key == ConsoleKey.End)

{

goAgain = false;

break;

}

} while (currentLine < filegroup.Count());

if (goAgain == false)

break;

}

}

}

**Cách: Truy vấn nội dung của tệp văn bản trong thư mục**

Ví dụ này cho thấy cách truy vấn tất cả các tệp trong cây thư mục được chỉ định, mở từng tệp và kiểm tra nội dung của nó. Loại kỹ thuật này có thể được sử dụng để tạo các chỉ mục hoặc chỉ mục ngược của nội dung của cây thư mục. Một tìm kiếm chuỗi đơn giản được thực hiện trong ví dụ này. Tuy nhiên, các kiểu khớp mẫu phức tạp hơn có thể được thực hiện bằng biểu thức chính quy.

class QueryContents

{

public static void Main()

{

// Modify this path as necessary.

string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";

// Take a snapshot of the file system.

System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(startFolder);

// This method assumes that the application has discovery permissions

// for all folders under the specified path.

IEnumerable<System.IO.FileInfo> fileList = dir.GetFiles("\*.\*", System.IO.SearchOption.AllDirectories);

string searchTerm = @"Visual Studio";

// Search the contents of each file.

// A regular expression created with the RegEx class

// could be used instead of the Contains method.

// queryMatchingFiles is an IEnumerable<string>.

var queryMatchingFiles =

from file in fileList

where file.Extension == ".htm"

let fileText = GetFileText(file.FullName)

where fileText.Contains(searchTerm)

select file.FullName;

// Execute the query.

Console.WriteLine("The term \"{0}\" was found in:", searchTerm);

foreach (string filename in queryMatchingFiles)

{

Console.WriteLine(filename);

}

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit");

Console.ReadKey();

}

// Read the contents of the file.

static string GetFileText(string name)

{

string fileContents = String.Empty;

// If the file has been deleted since we took

// the snapshot, ignore it and return the empty string.

if (System.IO.File.Exists(name))

{

fileContents = System.IO.File.ReadAllText(name);

}

return fileContents;

}

}

**Cách: Truy vấn một ArrayList bằng LINQ**

Khi sử dụng LINQ để truy vấn các bộ sưu tập IEnumerable không chung chung như ArrayList , bạn phải khai báo rõ ràng loại biến phạm vi để phản ánh loại cụ thể của các đối tượng trong bộ sưu tập. Ví dụ: nếu bạn có một ArrayList của Studentcác đối tượng, mệnh đề from của bạn sẽ trông như thế này:

var query = from Student s in arrList

Bằng cách chỉ định loại biến phạm vi, bạn đang truyền từng mục trong ArrayList sang a Student.

Việc sử dụng biến phạm vi được gõ rõ ràng trong biểu thức truy vấn tương đương với việc gọi phương thức Cast . Cast ném một ngoại lệ nếu cast được chỉ định không thể được thực hiện. Cast và OfType là hai phương thức Toán tử truy vấn tiêu chuẩn hoạt động trên các loại IEnumerable không chung chung.

using System;

using System.Collections;

using System.Linq;

namespace NonGenericLINQ

{

public class Student

{

public string FirstName { get; set; }

public string LastName { get; set; }

public int[] Scores { get; set; }

}

class Program

{

static void Main(string[] args)

{

ArrayList arrList = new ArrayList();

arrList.Add(

new Student

{

FirstName = "Svetlana", LastName = "Omelchenko", Scores = new int[] { 98, 92, 81, 60 }

});

arrList.Add(

new Student

{

FirstName = "Claire", LastName = "O’Donnell", Scores = new int[] { 75, 84, 91, 39 }

});

arrList.Add(

new Student

{

FirstName = "Sven", LastName = "Mortensen", Scores = new int[] { 88, 94, 65, 91 }

});

arrList.Add(

new Student

{

FirstName = "Cesar", LastName = "Garcia", Scores = new int[] { 97, 89, 85, 82 }

});

var query = from Student student in arrList

where student.Scores[0] > 95

select student;

foreach (Student s in query)

Console.WriteLine(s.LastName + ": " + s.Scores[0]);

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

}

/\* Output:

Omelchenko: 98

Garcia: 97

\*/

**Cách: Thêm phương thức tùy chỉnh cho truy vấn LINQ**

Bạn có thể mở rộng tập hợp các phương thức mà bạn có thể sử dụng cho các truy vấn LINQ bằng cách thêm các phương thức mở rộng vào giao diện <T> của IEnumerable . Ví dụ: ngoài các hoạt động trung bình hoặc tối đa tiêu chuẩn, bạn có thể tạo một phương pháp tổng hợp tùy chỉnh để tính toán một giá trị từ một chuỗi các giá trị. Bạn cũng có thể tạo một phương thức hoạt động như một bộ lọc tùy chỉnh hoặc một biến đổi dữ liệu cụ thể cho một chuỗi các giá trị và trả về một chuỗi mới. Ví dụ về các phương pháp như vậy là Phân biệt , Bỏ qua và Đảo ngược .

Khi bạn mở rộng giao diện <T> IEnumerable , bạn có thể áp dụng các phương thức tùy chỉnh của mình cho bất kỳ bộ sưu tập nào. Để biết thêm thông tin, xem Phương pháp mở rộng .

**Adding an Aggregate Method**

public static class LINQExtension

{

public static double Median(this IEnumerable<double> source)

{

if (source.Count() == 0)

{

throw new InvalidOperationException("Cannot compute median for an empty set.");

}

var sortedList = from number in source

orderby number

select number;

int itemIndex = (int)sortedList.Count() / 2;

if (sortedList.Count() % 2 == 0)

{

// Even number of items.

return (sortedList.ElementAt(itemIndex) + sortedList.ElementAt(itemIndex - 1)) / 2;

}

else

{

// Odd number of items.

return sortedList.ElementAt(itemIndex);

}

}

}

**Adding a Method That Returns a Collectio**

// Extension method for the IEnumerable<T> interface.

// The method returns every other element of a sequence.

public static IEnumerable<T> AlternateElements<T>(this IEnumerable<T> source)

{

List<T> list = new List<T>();

int i = 0;

foreach (var element in source)

{

if (i % 2 == 0)

{

list.Add(element);

}

i++;

}

return list;

}