许多文件系统操作本质上都是查询，因此非常适合 LINQ 方法。这些查询是非破坏性的。它们不会更改原始文件或文件夹的内容。查询不应引起任何副作用。一般来说，修改源数据的任何代码（包括执行创建/更新/删除操作的查询）都应与仅查询数据的代码分开。

创建准确表示文件系统内容并妥善处理异常的数据源涉及一些复杂性。本节中的示例创建[FileInfo对象的快照集合，该集合表示指定根文件夹及其所有子文件夹下的所有文件。每个FileInfo](https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo)的实际状态可能会在开始和结束执行查询之间的时间内发生变化。例如，您可以创建一个[FileInfo](https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo)对象列表以用作数据源。如果您尝试Length在查询中访问属性，[FileInfo](https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo)对象将尝试访问文件系统以更新的值Length。如果文件不再存在，您会在查询中收到[FileNotFoundException](https://learn.microsoft.com/en-us/dotnet/api/system.io.filenotfoundexception)，即使您没有直接查询文件系统。

**如何查询具有指定属性或名称的文件**

此示例显示如何在指定的目录树中查找具有指定文件扩展名（例如“.txt”）的所有文件。它还显示了如何根据创建时间返回树中最新或最旧的文件。无论您是在 Windows、Mac 还是 Linux 系统上运行此代码，都可能需要修改许多示例的第一行。

代码#复制

string startFolder = """C:\Program Files\dotnet\sdk""";

// Or

// string startFolder = "/usr/local/share/dotnet/sdk";

DirectoryInfo dir = new DirectoryInfo(startFolder);

var fileList = dir.GetFiles("\*.\*", SearchOption.AllDirectories);

var fileQuery = from file in fileList

where file.Extension == ".txt"

orderby file.Name

select file;

// Uncomment this block to see the full query

// foreach (FileInfo fi in fileQuery)

// {

// Console.WriteLine(fi.FullName);

// }

var newestFile = (from file in fileQuery

orderby file.CreationTime

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

.Last();

Console.WriteLine($"\r\nThe newest .txt file is {newestFile.FullName}. Creation time: {newestFile.CreationTime}");

**如何按扩展名对文件进行分组**

此示例展示了如何使用 LINQ 对文件或文件夹列表执行高级分组和排序操作。它还展示了如何使用[Skip](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.skip)和[Take](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.take)方法在控制台窗口中分页输出。

以下查询显示如何按文件扩展名对指定目录树的内容进行分组。

代码#复制

string startFolder = """C:\Program Files\dotnet\sdk""";

// Or

// string startFolder = "/usr/local/share/dotnet/sdk";

int trimLength = startFolder.Length;

DirectoryInfo dir = new DirectoryInfo(startFolder);

var fileList = dir.GetFiles("\*.\*", SearchOption.AllDirectories);

var queryGroupByExt = from file in fileList

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

orderby fileGroup.Count(), fileGroup.Key

select fileGroup;

// Iterate through the outer collection of groups.

foreach (var filegroup in queryGroupByExt.Take(5))

{

Console.WriteLine($"Extension: {filegroup.Key}");

var resultPage = filegroup.Take(20);

//Execute the resultPage query

foreach (var f in resultPage)

{

Console.WriteLine($"\t{f.FullName.Substring(trimLength)}");

}

Console.WriteLine();

}

此程序的输出可能很长，具体取决于本地文件系统的详细信息以及startFolder设置的内容。为了能够查看所有结果，此示例显示了如何分页查看结果。foreach需要嵌套循环，因为每个组都是单独枚举的。

**如何查询一组文件夹的总字节数**

此示例显示如何检索指定文件夹及其所有子文件夹中所有文件使用的总字节数。[Sum](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.sum)方法将子句中选择的所有项的值相加。 您可以通过调用[Min](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.min)或[Max](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.max)方法（而不是[Sum](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.sum)select ）来修改此查询以检索指定目录树中的最大或最小文件。

代码#复制

string startFolder = """C:\Program Files\dotnet\sdk""";

// Or

// string startFolder = "/usr/local/share/dotnet/sdk";

var fileList = Directory.GetFiles(startFolder, "\*.\*", SearchOption.AllDirectories);

var fileQuery = from file in fileList

let fileLen = new FileInfo(file).Length

where fileLen > 0

select fileLen;

// 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 {totalBytes} bytes in {fileList.Count()} files under {startFolder}");

Console.WriteLine($"The largest file is {largestFile} bytes.");

此示例扩展了前面的示例，以执行以下操作：

* 如何检索最大文件的大小（以字节为单位）。
* 如何检索最小文件的大小（以字节为单位）。
* 如何从指定根文件夹下的一个或多个文件夹中检索[FileInfo对象最大或最小的文件。](https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo)
* 如何检索诸如 10 个最大文件的序列。
* 如何根据文件大小（以字节为单位）将文件分组，忽略小于指定大小的文件。

以下示例包含五个单独的查询，它们显示了如何根据文件大小（以字节为单位）查询和分组文件。您可以修改这些示例，使查询基于[FileInfo](https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo)对象的其他属性。

代码#复制

// Return the FileInfo object for the largest file

// by sorting and selecting from beginning of list

FileInfo longestFile = (from file in fileList

let fileInfo = new FileInfo(file)

where fileInfo.Length > 0

orderby fileInfo.Length descending

select fileInfo

).First();

Console.WriteLine($"The largest file under {startFolder} is {longestFile.FullName} with a length of {longestFile.Length} bytes");

//Return the FileInfo of the smallest file

FileInfo smallestFile = (from file in fileList

let fileInfo = new FileInfo(file)

where fileInfo.Length > 0

orderby fileInfo.Length ascending

select fileInfo

).First();

Console.WriteLine($"The smallest file under {startFolder} is {smallestFile.FullName} with a length of {smallestFile.Length} bytes");

//Return the FileInfos for the 10 largest files

var queryTenLargest = (from file in fileList

let fileInfo = new FileInfo(file)

let len = fileInfo.Length

orderby len descending

select fileInfo

).Take(10);

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

foreach (var v in queryTenLargest)

{

Console.WriteLine($"{v.FullName}: {v.Length} bytes");

}

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

// files that are less than 200000 bytes.

var querySizeGroups = from file in fileList

let fileInfo = new FileInfo(file)

let len = fileInfo.Length

where len > 0

group fileInfo by (len / 100000) into fileGroup

where fileGroup.Key >= 2

orderby fileGroup.Key descending

select fileGroup;

foreach (var filegroup in querySizeGroups)

{

Console.WriteLine($"{filegroup.Key}00000");

foreach (var item in filegroup)

{

Console.WriteLine($"\t{item.Name}: {item.Length}");

}

}

要返回一个或多个完整的[FileInfo](https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo)对象，查询首先必须检查数据源中的每个对象，然后按其 Length 属性的值对它们进行排序。然后它可以返回单个对象或具有最大长度的序列。使用[First](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.first)返回列表中的第一个元素。使用[Take](https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.take)返回前 n 个元素。指定降序排序顺序以将最小元素放在列表的开头。

**如何查询目录树中的重复文件**

有时，同名文件可能位于多个文件夹中。此示例显示如何在指定的根文件夹下查询此类重复文件名。第二个示例显示如何查询大小和 LastWrite 时间也匹配的文件。

代码#复制

string startFolder = """C:\Program Files\dotnet\sdk""";

// Or

// string startFolder = "/usr/local/share/dotnet/sdk";

DirectoryInfo dir = new DirectoryInfo(startFolder);

IEnumerable<FileInfo> fileList = dir.GetFiles("\*.\*", 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;

foreach (var queryDup in queryDupNames.Take(20))

{

Console.WriteLine($"Filename = {(queryDup.Key.ToString() == string.Empty ? "[none]" : queryDup.Key.ToString())}");

foreach (var fileName in queryDup.Take(10))

{

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

}

}

第一个查询使用键来确定匹配项。它会查找具有相同名称但内容可能不同的文件。第二个查询使用复合键来匹配[FileInfo](https://learn.microsoft.com/en-us/dotnet/api/system.io.fileinfo)对象的三个属性。此查询更有可能找到具有相同名称且内容相似或相同的文件。

代码#复制

string startFolder = """C:\Program Files\dotnet\sdk""";

// Or

// string startFolder = "/usr/local/share/dotnet/sdk";

// Make the lines shorter for the console display

int charsToSkip = startFolder.Length;

// Take a snapshot of the file system.

DirectoryInfo dir = new DirectoryInfo(startFolder);

IEnumerable<FileInfo> fileList = dir.GetFiles("\*.\*", 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

(Name: file.Name, LastWriteTime: file.LastWriteTime, Length: file.Length )

into fileGroup

where fileGroup.Count() > 1

select fileGroup;

foreach (var queryDup in queryDupFiles.Take(20))

{

Console.WriteLine($"Filename = {(queryDup.Key.ToString() == string.Empty ? "[none]" : queryDup.Key.ToString())}");

foreach (var fileName in queryDup)

{

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

}

}

}

**如何查询文件夹中的文本文件的内容**

此示例显示如何查询指定目录树中的所有文件、打开每个文件并检查其内容。此类技术可用于创建目录树内容的索引或反向索引。此示例中执行了简单的字符串搜索。但是，可以使用正则表达式执行更复杂类型的模式匹配。

代码#复制

string startFolder = """C:\Program Files\dotnet\sdk""";

// Or

// string startFolder = "/usr/local/share/dotnet/sdk";

DirectoryInfo dir = new DirectoryInfo(startFolder);

var fileList = dir.GetFiles("\*.\*", SearchOption.AllDirectories);

string searchTerm = "change";

var queryMatchingFiles = from file in fileList

where file.Extension == ".txt"

let fileText = File.ReadAllText(file.FullName)

where fileText.Contains(searchTerm)

select file.FullName;

// Execute the query.

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

foreach (string filename in queryMatchingFiles)

{

Console.WriteLine(filename);

}

**如何比较两个文件夹的内容**

此示例演示了比较两个文件列表的三种方法：

* 通过查询指定两个文件列表是否相同的布尔值。
* 通过查询交集来检索两个文件夹中的文件。
* 通过查询设置差异来检索一个文件夹中存在但不在另一个文件夹中的文件。

这里展示的技术可以适用于比较任何类型的对象序列。

此处显示的类FileComparer演示了如何将自定义比较器类与标准查询运算符一起使用。该类不适用于实际场景。它仅使用每个文件的名称和长度（以字节为单位）来确定每个文件夹的内容是否相同。在实际场景中，您应该修改此比较器以执行更严格的相等性检查。

代码#复制

// 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 : IEqualityComparer<FileInfo>

{

public bool Equals(FileInfo? f1, 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(FileInfo fi)

{

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

return s.GetHashCode();

}

}

public static void CompareDirectories()

{

string pathA = """C:\Program Files\dotnet\sdk\8.0.104""";

string pathB = """C:\Program Files\dotnet\sdk\8.0.204""";

DirectoryInfo dir1 = new DirectoryInfo(pathA);

DirectoryInfo dir2 = new DirectoryInfo(pathB);

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

IEnumerable<FileInfo> list2 = dir2.GetFiles("\*.\*", 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.Any())

{

Console.WriteLine($"The following files are in both folders (total number = {queryCommonFiles.Count()}):");

foreach (var v in queryCommonFiles.Take(10))

{

Console.WriteLine(v.Name); //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.

var queryList1Only = (from file in list1

select file)

.Except(list2, myFileCompare);

Console.WriteLine();

Console.WriteLine($"The following files are in list1 but not list2 (total number = {queryList1Only.Count()}):");

foreach (var v in queryList1Only.Take(10))

{

Console.WriteLine(v.FullName);

}

var queryList2Only = (from file in list2

select file)

.Except(list1, myFileCompare);

Console.WriteLine();

Console.WriteLine($"The following files are in list2 but not list1 (total number = {queryList2Only.Count()}:");

foreach (var v in queryList2Only.Take(10))

{

Console.WriteLine(v.FullName);

}

}

**如何重新排序分隔文件的字段**

逗号分隔值 (CSV) 文件是一种文本文件，通常用于存储电子表格数据或其他由行和列表示的表格数据。通过使用[Split](https://learn.microsoft.com/en-us/dotnet/api/system.string.split)方法分隔字段，可以轻松使用 LINQ 查询和操作 CSV 文件。事实上，同样的技术可用于重新排序任何结构化文本行的各个部分；它并不局限于 CSV 文件。

在以下示例中，假设三列代表学生的“姓氏”、“名字”和“ID”。这些字段根据学生的姓氏按字母顺序排列。查询会生成一个新序列，其中 ID 列首先出现，然后是第二列，该列结合了学生的名字和姓氏。这些行会根据 ID 字段重新排序。结果会保存到新文件中，原始数据不会被修改。以下文本显示了以下示例中使用的*电子表格 1.csv*文件的内容：

TXT复制

Adams,Terry,120

Fakhouri,Fadi,116

Feng,Hanying,117

Garcia,Cesar,114

Garcia,Debra,115

Garcia,Hugo,118

Mortensen,Sven,113

O'Donnell,Claire,112

Omelchenko,Svetlana,111

Tucker,Lance,119

Tucker,Michael,122

Zabokritski,Eugene,121

以下代码读取源文件并重新排列 CSV 文件中的每一列，以重新排列列的顺序：

代码#复制

string[] lines = 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 fields = line.Split(',')

orderby fields[2]

select $"{fields[2]}, {fields[1]} {fields[0]}";

File.WriteAllLines("spreadsheet2.csv", query.ToArray());

/\* 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

\*/

**如何使用组将文件拆分为多个文件**

此示例展示了一种合并两个文件内容然后创建一组以新方式组织数据的新文件的方法。查询使用两个文件的内容。以下文本显示第一个文件*names1.txt*的内容：

TXT复制

Bankov, Peter

Holm, Michael

Garcia, Hugo

Potra, Cristina

Noriega, Fabricio

Aw, Kam Foo

Beebe, Ann

Toyoshima, Tim

Guy, Wey Yuan

Garcia, Debra

第二个文件*names2.txt*包含一组不同的名称，其中一些名称与第一组名称相同：

TXT复制

Liu, Jinghao

Bankov, Peter

Holm, Michael

Garcia, Hugo

Beebe, Ann

Gilchrist, Beth

Myrcha, Jacek

Giakoumakis, Leo

McLin, Nkenge

El Yassir, Mehdi

以下代码查询两个文件，获取两个文件的并集，然后为每个组写入一个新文件，由姓氏的首字母定义：

代码#复制

string[] fileA = File.ReadAllLines("names1.txt");

string[] fileB = File.ReadAllLines("names2.txt");

// Concatenate and remove duplicate names

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(',')[0]

group name by n[0] into g

orderby g.Key

select g;

foreach (var g in groupQuery)

{

string fileName = $"testFile\_{g.Key}.txt";

Console.WriteLine(g.Key);

using StreamWriter sw = new StreamWriter(fileName);

foreach (var item in g)

{

sw.WriteLine(item);

// Output to console for example purposes.

Console.WriteLine($" {item}");

}

}

/\* 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

\*/

**如何合并不同文件的内容**

此示例显示如何合并两个逗号分隔文件中的数据，这两个文件共享一个公共值，该公共值用作匹配键。如果您必须将两个电子表格中的数据，或一个电子表格中的数据和一个具有另一种格式的文件中的数据合并到一个新文件中，此方法非常有用。您可以修改此示例以处理任何类型的结构化文本。

*以下文本显示了scores.csv*的内容。该文件表示电子表格数据。第 1 列是学生的 ID，第 2 列至第 5 列是考试成绩。

TXT复制

111, 97, 92, 81, 60

112, 75, 84, 91, 39

113, 88, 94, 65, 91

114, 97, 89, 85, 82

115, 35, 72, 91, 70

116, 99, 86, 90, 94

117, 93, 92, 80, 87

118, 92, 90, 83, 78

119, 68, 79, 88, 92

120, 99, 82, 81, 79

121, 96, 85, 91, 60

122, 94, 92, 91, 91

*以下文本显示names.csv*的内容。该文件表示包含学生姓氏、名字和学生 ID 的电子表格。

TXT复制

Omelchenko,Svetlana,111

O'Donnell,Claire,112

Mortensen,Sven,113

Garcia,Cesar,114

Garcia,Debra,115

Fakhouri,Fadi,116

Feng,Hanying,117

Garcia,Hugo,118

Tucker,Lance,119

Adams,Terry,120

Zabokritski,Eugene,121

Tucker,Michael,122

连接包含相关信息的不同文件的内容。文件*names.csv*包含学生姓名和 ID 号。文件*scores.csv*包含 ID 和一组四个测试分数。以下查询使用 ID 作为匹配键将分数与学生姓名连接起来。代码如下例所示：

代码#复制

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

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

var scoreQuery = from name in names

let nameFields = name.Split(',')

from id in scores

let scoreFields = id.Split(',')

where Convert.ToInt32(nameFields[2]) == Convert.ToInt32(scoreFields[0])

select $"{nameFields[0]},{scoreFields[1]},{scoreFields[2]},{scoreFields[3]},{scoreFields[4]}";

Console.WriteLine("\r\nMerge two spreadsheets:");

foreach (string item in scoreQuery)

{

Console.WriteLine(item);

}

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

/\* Output:

Merge two spreadsheets:

Omelchenko, 97, 92, 81, 60

O'Donnell, 75, 84, 91, 39

Mortensen, 88, 94, 65, 91

Garcia, 97, 89, 85, 82

Garcia, 35, 72, 91, 70

Fakhouri, 99, 86, 90, 94

Feng, 93, 92, 80, 87

Garcia, 92, 90, 83, 78

Tucker, 68, 79, 88, 92

Adams, 99, 82, 81, 79

Zabokritski, 96, 85, 91, 60

Tucker, 94, 92, 91, 91

12 total names in list

\*/

**如何计算 CSV 文本文件中的列值**

此示例显示如何对 .csv 文件的列执行聚合计算，例如 Sum、Average、Min 和 Max。此处显示的示例原理可应用于其他类型的结构化文本。

*以下文本显示了scores.csv*的内容。假设第一列代表学生 ID，后续列代表四次考试的成绩。

TXT复制

111, 97, 92, 81, 60

112, 75, 84, 91, 39

113, 88, 94, 65, 91

114, 97, 89, 85, 82

115, 35, 72, 91, 70

116, 99, 86, 90, 94

117, 93, 92, 80, 87

118, 92, 90, 83, 78

119, 68, 79, 88, 92

120, 99, 82, 81, 79

121, 96, 85, 91, 60

122, 94, 92, 91, 91

以下文本显示如何使用[Split](https://learn.microsoft.com/en-us/dotnet/api/system.string.split)方法将每行文本转换为数组。每个数组元素代表一列。最后，每列中的文本将转换为其数字表示形式。

代码#复制

public class SumColumns

{

public static void SumCSVColumns(string fileName)

{

string[] lines = File.ReadAllLines(fileName);

// 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);

}

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 run time.

// 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 #{examNum}: Average:{average:##.##} High Score:{max} Low Score:{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.

var 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 #{column + 1} Average: {average:##.##} High Score: {max} Low Score: {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

\*/

如果您的文件是制表符分隔的文件，只需将方法中的参数更新Split为\t。