How to: Compute Column Values in a CSV Text File (LINQ) (C#)

Visual Studio 2015

This example shows how to perform aggregate computations such as Sum, Average, Min, and Max on the columns of a .csv file. The example principles that are shown here can be applied to other types of structured text.

To create the source file

1. Copy the following lines into a file that is named scores.csv and save it in your project folder. Assume that the first column represents a student ID, and subsequent columns represent scores from four exams.

```
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
```

Example

C#

```
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
*/
```

The query works by using the Split method to convert each line of text into an array. Each array element represents a column. Finally, the text in each column is converted to its numeric representation. If your file is a tab-separated file, just update the argument in the **Split** method to `\t`.

Compiling the Code

Create a project that targets the .NET Framework version 3.5 or higher, with a reference to System.Core.dll and **using** directives for the System.Linq and System.IO namespaces.

See Also

LINQ and Strings (C#)
LINQ and File Directories (C#)