Create a nested group

The following example shows how to create nested groups in a LINQ query expression. Each group that is created according to student year or grade level is then further subdivided into groups based on the individuals' names.

Example

Note

This example contains references to objects that are defined in the sample code in **Query a collection of objects**.

```
public class StudentClass
{
    #region data
    protected enum GradeLevel { FirstYear = 1, SecondYear, ThirdYear, FourthYear
};
    protected class Student
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int ID { get; set; }
        public GradeLevel Year;
        public List<int> ExamScores;
    }

    protected static List<Student> students = new List<Student>
    {
        new Student {FirstName = "Terry", LastName = "Adams", ID = 120,
            Year = GradeLevel.SecondYear,
            ExamScores = new List<int>{ 99, 82, 81, 79}},
        new Student {FirstName = "Fadi", LastName = "Fakhouri", ID = 116,
            Year = GradeLevel.ThirdYear,
            ExamScores = new List<int>{ 99, 86, 90, 94}},
        new Student {FirstName = "Hanying", LastName = "Feng", ID = 117,
            Year = GradeLevel.FirstYear,
            ExamScores = new List<int>{ 93, 92, 80, 87}},
        new Student {FirstName = "Cesar", LastName = "Garcia", ID = 114,
            Year = GradeLevel.FourthYear,
            ExamScores = new List<int>{ 97, 89, 85, 82}},
        new Student {FirstName = "Debra", LastName = "Garcia", ID = 115,
            Year = GradeLevel.ThirdYear,
            ExamScores = new List<int>{ 35, 72, 91, 70}},
        new Student {FirstName = "Hugo", LastName = "Garcia", ID = 118,
            Year = GradeLevel.SecondYear,
            ExamScores = new List<int>{ 92, 90, 83, 78}},
        new Student {FirstName = "Sven", LastName = "Mortensen", ID = 113,
            Year = GradeLevel.FirstYear,
            ExamScores = new List<int>{ 88, 94, 65, 91}},
        new Student {FirstName = "Claire", LastName = "O'Donnell", ID = 112,
```

```
            Year = GradeLevel.FourthYear,
            ExamScores = new List<int>{ 75, 84, 91, 39}},
        new Student {FirstName = "Svetlana", LastName = "Omelchenko", ID = 111,
            Year = GradeLevel.SecondYear,
            ExamScores = new List<int>{ 97, 92, 81, 60}},
        new Student {FirstName = "Lance", LastName = "Tucker", ID = 119,
            Year = GradeLevel.ThirdYear,
            ExamScores = new List<int>{ 68, 79, 88, 92}},
        new Student {FirstName = "Michael", LastName = "Tucker", ID = 122,
            Year = GradeLevel.FirstYear,
            ExamScores = new List<int>{ 94, 92, 91, 91}},
        new Student {FirstName = "Eugene", LastName = "Zabokritski", ID = 121,
            Year = GradeLevel.FourthYear,
            ExamScores = new List<int>{ 96, 85, 91, 60}}
    };
    #endregion

    //Helper method, used in GroupByRange.
    protected static int GetPercentile(Student s)
    {
        double avg = s.ExamScores.Average();
        return avg > 0 ? (int)avg / 10 : 0;
    }



    public void QueryHighScores(int exam, int score)
    {
        var highScores = from student in students
                         where student.ExamScores[exam] > score
                         select new {Name = student.FirstName, Score =
student.ExamScores[exam]};

        foreach (var item in highScores)
        {
            Console.WriteLine($"{item.Name,-15}{item.Score}");
        }
    }
}

public class Program
{
    public static void Main()
    {
        StudentClass sc = new StudentClass();
        sc.QueryHighScores(1, 90);

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
```

```
public void QueryNestedGroups()
{
    var queryNestedGroups =
        from student in students
        group student by student.Year into newGroup1
        from newGroup2 in
            (from student in newGroup1
             group student by student.LastName)
        group newGroup2 by newGroup1.Key;

    // Three nested foreach loops are required to iterate
    // over all elements of a grouped group. Hover the mouse
    // cursor over the iteration variables to see their actual type.
    foreach (var outerGroup in queryNestedGroups)
    {
        Console.WriteLine($"DataClass.Student Level = {outerGroup.Key}");
        foreach (var innerGroup in outerGroup)
        {
            Console.WriteLine($"\tNames that begin with: {innerGroup.Key}");
            foreach (var innerGroupElement in innerGroup)
            {
                Console.WriteLine($"\t\t{innerGroupElement.LastName}
{innerGroupElement.FirstName}");
            }
        }
    }
}
/*
 Output:
DataClass.Student Level = SecondYear
        Names that begin with: Adams
                Adams Terry
        Names that begin with: Garcia
                Garcia Hugo
        Names that begin with: Omelchenko
                Omelchenko Svetlana
DataClass.Student Level = ThirdYear
        Names that begin with: Fakhouri
                Fakhouri Fadi
        Names that begin with: Garcia
                Garcia Debra
        Names that begin with: Tucker
                Tucker Lance
DataClass.Student Level = FirstYear
        Names that begin with: Feng
```

```
                Feng Hanying
        Names that begin with: Mortensen
                Mortensen Sven
        Names that begin with: Tucker
                Tucker Michael
DataClass.Student Level = FourthYear
        Names that begin with: Garcia
                Garcia Cesar
        Names that begin with: O'Donnell
                O'Donnell Claire
        Names that begin with: Zabokritski
                Zabokritski Eugene
 */
```

Note that three nested `foreach` loops are required to iterate over the inner elements of a nested group.

Perform a subquery on a grouping operation

This article shows two different ways to create a query that orders the source data into groups, and then performs a subquery over each group individually. The basic technique in each example is to group the source elements by using a *continuation* named `newGroup`, and then generating a new subquery against `newGroup`. This subquery is run against each new group that is created by the outer query. Note that in this particular example the final output is not a group, but a flat sequence of anonymous types.

For more information about how to group, see group clause.

For more information about continuations, see into. The following example uses an in-memory data structure as the data source, but the same principles apply for any kind of LINQ data source.

Example

```
public void QueryMax()
{
    var queryGroupMax =
        from student in students
```

```
        group student by student.Year into studentGroup
        select new
        {
            Level = studentGroup.Key,
            HighestScore =
            (from student2 in studentGroup
             select student2.ExamScores.Average()).Max()
        };

    int count = queryGroupMax.Count();
    Console.WriteLine($"Number of groups = {count}");

    foreach (var item in queryGroupMax)
    {
        Console.WriteLine($"  {item.Level} Highest
Score={item.HighestScore}");
    }
}
```

The query in the snippet above can also be written using method syntax. The following code snippet has a semantically equivalent query written using method syntax.
C#Copy

```
public void QueryMaxUsingMethodSyntax()
{
    var queryGroupMax = students
        .GroupBy(student => student.Year)
        .Select(studentGroup => new
        {
            Level = studentGroup.Key,
            HighestScore = studentGroup.Select(student2 =>
student2.ExamScores.Average()).Max()
        });

    int count = queryGroupMax.Count();
    Console.WriteLine($"Number of groups = {count}");

    foreach (var item in queryGroupMax)
    {
        Console.WriteLine($"  {item.Level} Highest
Score={item.HighestScore}");
    }
}
```