**执行自定义联接操作（C# 编程指南）**

此示例演示如何执行无法使用 **join** 子句执行的联接操作。在查询表达式中，**join** 子句仅适用于同等联接（这是迄今为止最常见的联接操作类型），并针对同等联接进行了优化。执行同等联接时，一般总是可以通过使用 **join** 子句获得最佳性能。

但是，在下面一些情况中，无法使用 **join** 子句：

* 联接是在不等式（非同等联接）上断言的。
* 联接是在多个等式或不等式上断言的。
* 必须在联接操作之前为右侧（内部）序列引入一个临时范围变量。

若要执行非同等联接，可以使用多个 **from** 子句单独引入每个数据源。然后，在 **where** 子句中将谓词表达式应用于每个源的范围变量。该表达式还可以采用方法调用的形式。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 不要将这种自定义联接操作与使用多个 **from** 子句访问内部集合相混淆。有关更多信息，请参见 [join 子句（C# 参考）](https://msdn.microsoft.com/zh-cn/library/bb311040.aspx)。 |

下面示例中的第一个方法演示了一个简单的交叉联接。必须慎用交叉联接，因为它们可能产生非常大的结果集。但在某些方案中，可以使用它们创建源序列以供运行附加查询。

第二个方法产生其类别 ID 列在左侧类别列表中的所有产品的序列。请注意，这种方法使用 **let** 子句和 **Contains** 方法创建了一个临时数组。还可以在查询前创建该数组并去掉第一个 **from** 子句。

## 实例1

class CustomJoins

{

#region Data

class Product

{

public string Name { get; set; }

public int CategoryID { get; set; }

}

class Category

{

public string Name { get; set; }

public int ID { get; set; }

}

// Specify the first data source.

List<Category> categories = new List<Category>()

{

new Category(){Name="Beverages", ID=001},

new Category(){ Name="Condiments", ID=002},

new Category(){ Name="Vegetables", ID=003},

};

// Specify the second data source.

List<Product> products = new List<Product>()

{

new Product{Name="Tea", CategoryID=001},

new Product{Name="Mustard", CategoryID=002},

new Product{Name="Pickles", CategoryID=002},

new Product{Name="Carrots", CategoryID=003},

new Product{Name="Bok Choy", CategoryID=003},

new Product{Name="Peaches", CategoryID=005},

new Product{Name="Melons", CategoryID=005},

new Product{Name="Ice Cream", CategoryID=007},

new Product{Name="Mackerel", CategoryID=012},

};

#endregion

static void Main()

{

CustomJoins app = new CustomJoins();

app.NonEquijoin();

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

Console.ReadKey();

}

void NonEquijoin()

{

var nonEquijoinQuery =

from p in products

let catIds = from c in categories

select c.ID

where catIds.Contains(p.CategoryID) == true

select new { Product = p.Name, CategoryID = p.CategoryID };

Console.WriteLine("Non-equijoin query:");

foreach (var v in nonEquijoinQuery)

{

Console.WriteLine("{0,-5}{1}", v.CategoryID, v.Product);

}

}

}

/\*结果

Non-equijoin query:

1 Tea

2 Mustard

2 Pickles

3 Carrots

3 Bok Choy

Press any key to exit.\*/

## 实例2

### *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。

|  |
| --- |
| 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 |

### 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。

|  |
| --- |
| 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 |

### 这两个文件的数据是通过一个学生编号来连接的.可以利用这个编号找到学生的姓名并且取出成绩

## 代码如下,不过不知道什么原因抛异常了.但是数据能够正常读取

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.IO;  using System.ComponentModel;  namespace linq\_custom\_query\_from\_file  {  internal class Program  {  static void CustomQueryFromFile()  {  string[] names = File.ReadAllLines(@"names.csv");  string[] scores = File.ReadAllLines(@"scores.csv");  try  {  var scoreQuery = from name in names  let nameFields = name.Split(',')  from score in scores  let scoreFields = score.Split(',')  where nameFields[2] == scoreFields[0]  select $"{nameFields[0]},{scoreFields[1]},{scoreFields[2]},{scoreFields[3]},{scoreFields[4]}";  foreach (var item in scoreQuery)  {  Console.WriteLine(item);  }  Console.WriteLine($"{scoreQuery.Count()} records");  }  catch (Exception ex) {  Console.WriteLine("there is an exception");  }    }  static void Main(string[] args)  {  CustomQueryFromFile();  }  }  } |