Základy LINQu

8. dubna 2019

LINQ neboli Language Integrated Query je sada jazykových funkcí pro psaní strukturovaných typově bezpečných dotazů přes lokální kolekce objektů a vzdálená data. LINQ byl představen v C # 3.0 a frameworku 3.5.

Základní jednotky dat v LINQ jsou sekvence a prvky. Sekvence je libovolný objekt, který implementuje obecné IEnumerable rozhraní a prvek je každá položka v sekvenci. V následujícím příkladu jsou názvy sekvencemi a Tom, Říša a Jindra jsou prvky:

```
string [] names = {"Tom", "Říša", "Jindra"};
```

Sekvence, jako je tato, nazýváme lokální sekvencí , protože reprezentuje lokální sbírku objektů v paměti. Operátor dotazu je metoda, která transformuje sekvenci. Typický dotazovací operátor přijme vstupní sekvenci a vydá transformovanou výstupní sekvenci. Ve třídě Enumerable v System.Linq existuje kolem 40 operátorů dotazů, všechny implementovány jako metody statického rozšíření. Ty se nazývají standardní operátory dotazů.

Jednoduchý dotaz

Dotaz je výraz, který transformuje sekvence jedním nebo více operátory dotazů. Nejjednodušší dotaz obsahuje jednu vstupní sekvenci a jeden operátor. Například můžeme použít operátor *Where* na jednoduchém poli, abychom mohli extrahovat jména, jejichž délka je nejméně čtyři znaky, takto:

```
string[] names = { "Tom", "Říša", "Jindra" };

IEnumerable<string> filteredNames =
System.Linq.Enumerable.Where ( names, n => n.Length >= 4);
foreach (string n in filteredNames) Console.Write (n + "|");
// Říša|Jindra|
```

Protože standardní operátory dotazů jsou implementovány jako metody rozšíření, můžeme volat Where přímo na names jakoby to byla metoda instance:

```
IEnumerable<string> filteredNames =
System.Linq.Enumerable.Where ( names, n => n.Length >= 4);
```

(Chcete-li kompilovat, musíte importovat *System.Linq* pomocí direktivy *using*.) Metoda *Where* v *System.Linq.Enumerable* má následující deklaraci:

```
static IEnumerable<TSource> Where<TSource> ( this IEnumerable<TSource> source, Func
    <TSource,bool> predicate)
```

Kde *source* je vstupní sekvence; *predicate* je delegát, který je vyvolán na každém vstupním prvku. Metoda *Where* vybere všechny prvky, pro které delegát vrací hodnotu true. Interně je generován kód pro iterátor:

```
foreach (TSource element in source)
  if (predicate (element))
    yield return element;
```

Projekce (Select)

Je aplikací funkce na každý prvek vstupní sekvence.

```
string[] names = { "Tom", "Říša", "Jindra" };

IEnumerable<string> upperNames = names.Select (n => n.ToUpper());
foreach (string n in upperNames)
  Console.Write (n + "|");
// TOM|ŘÍŠA|JINDRA|
```

S použitím anonymního typu

```
var query = names.Select (n => new { Name = n, Length = n.Length });
foreach (var row in query)
```

```
Console.WriteLine (row);
// { Name = Tom, Length = 3 }
// { Name = Ríša, Length = 4 }
// { Name = Jindra, Length = 5 }
```

(SelectMany)

```
double[] t = {1, 2, 3};
var coordinates = t.SelectMany(x=>new double[] { x, x*x, Pow(x,3)});
// coordinates { 1, 1, 1, 2, 4, 8, 3, 9, 27}
```

Filtry

```
int[] numbers = { 10, 9, 8, 7, 6 };
IEnumerable<int> firstThree = numbers.Take (3);
// firstThree je { 10, 9, 8 }
IEnumerable<int> lastTwo = numbers.Skip (3);
// lastTwo je { 7, 6 }
var whileGreater7 = numbers.TakeWhile(n=>n>7);
// whileGreater7 je { 10, 9, 8 }
var skipGreater7 = numbers.SkipWhile(n=>n>7);
// skipGreater7 je { 7, 6 }
```

```
numbers = { 10, 9, 8, 7, 6, 8, 10 };
var distinct = numbers.Distinct();
// distinct je { 10, 9, 8, 7, 6 }
```

Elementy

```
int[] numbers = { 10, 9, 8, 7, 6 };
int firstNumber = numbers.First(); // 10
int lastNumber = numbers.Last(); // 6
int secondNumber = numbers.ElementAt (2); // 8
int firstOddNum = numbers.First (n => n%2 == 1); // 9
```

Agregace

```
int[] numbers = { 10, 9, 8, 7, 6 };
int count = numbers.Count(); // 5
int min = numbers.Min(); // 6
int max = numbers.Max(); // 10
double avg = numbers.Average(); // 8
int evenNums = numbers.Count (n => n % 2 == 0); // 3
int maxRemainderAfterDivBy5 = numbers.Max (n => n % 5); // 4
```

Kvantifikátory

```
int[] numbers = { 10, 9, 8, 7, 6 };
bool hasTheNumberNine = numbers.Contains (9); // true
bool hasMoreThanZeroElements = numbers.Any(); // true
bool hasOddNum = numbers.Any (n => n % 2 == 1); // true
bool allOddNums = numbers.All (n => n % 2 == 1); // false
```

Množinové operace

```
int[] seq1 = { 1, 2, 3 }, seq2 = { 3, 4, 5 };
IEnumerable<int> concat = seq1.Concat (seq2), // { 1, 2, 3, 3, 4, 5 }
union = seq1.Union (seq2); // { 1, 2, 3, 4, 5 }
IEnumerable<int> common = seq1.Intersect (seq2), // { 3 }
difference1 = seq1.Except (seq2), // { 1, 2 }
difference2 = seq2.Except (seq1); // { 4, 5 }
```

Odložená aplikace dotazu

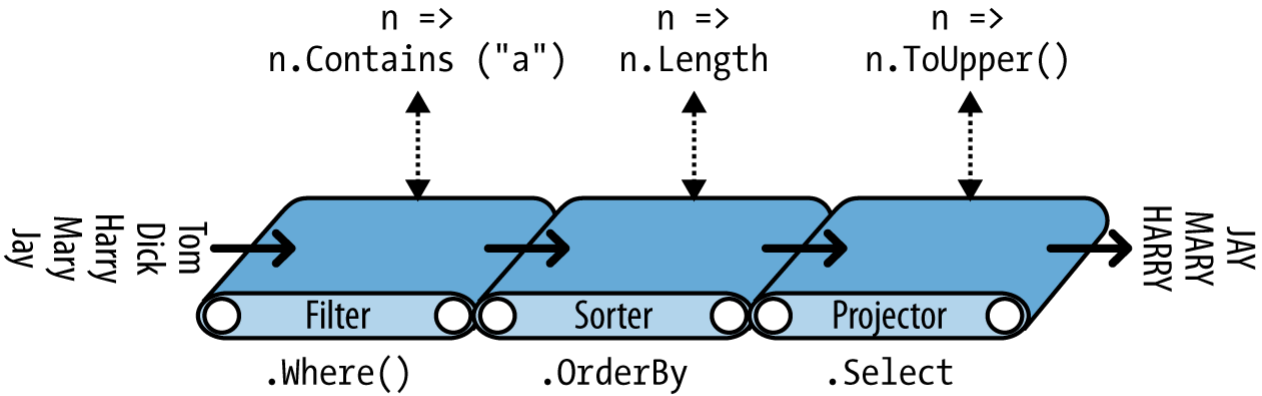
```
var numbers = new List<int> { 1 };
IEnumerable<int> query = numbers.Select (n => n * 10); numbers.Add (2); // Přidáme
    další prvek
foreach (int n in query)
```

```
  Console.Write (n + "|"); // 10|20|

var numbers = new List<int>() { 1, 2 };
List<int> timesTen = numbers .Select (n => n * 10) .ToList(); // Aplikace dotazu do
    typu List<int>
numbers.Clear();
Console.WriteLine (timesTen.Count); // Stále 2, kešování
```

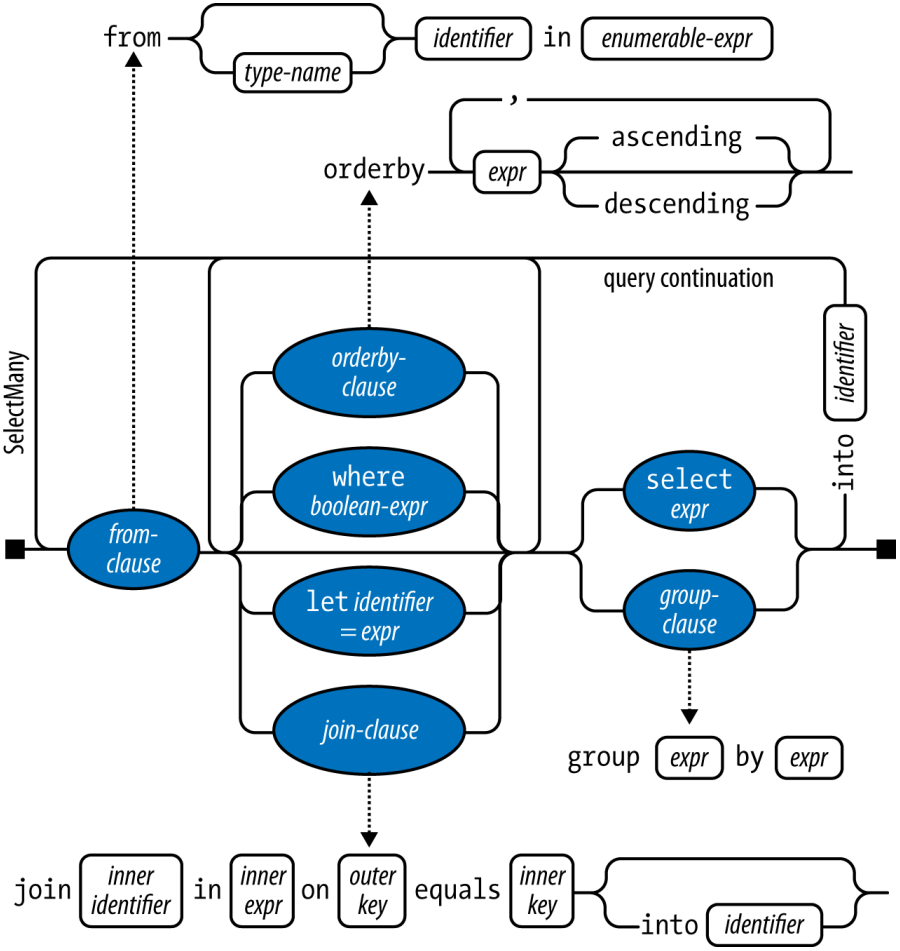
Kompozice dotazovacích operátorů

```
string[] names = { "Tom","Říša","Jindra","Marie","Jan" };
IEnumerable<string> query = names
  .Where (n => n.Contains ("a"))
  .OrderBy (n => n.Length)
  .Select (n => n.ToUpper());
foreach (string name in query)
  Console.Write (name + "|"); // JAN|RÍŠA|MARIE|JINDRA|
```



Dotazovací jazyk LINQu

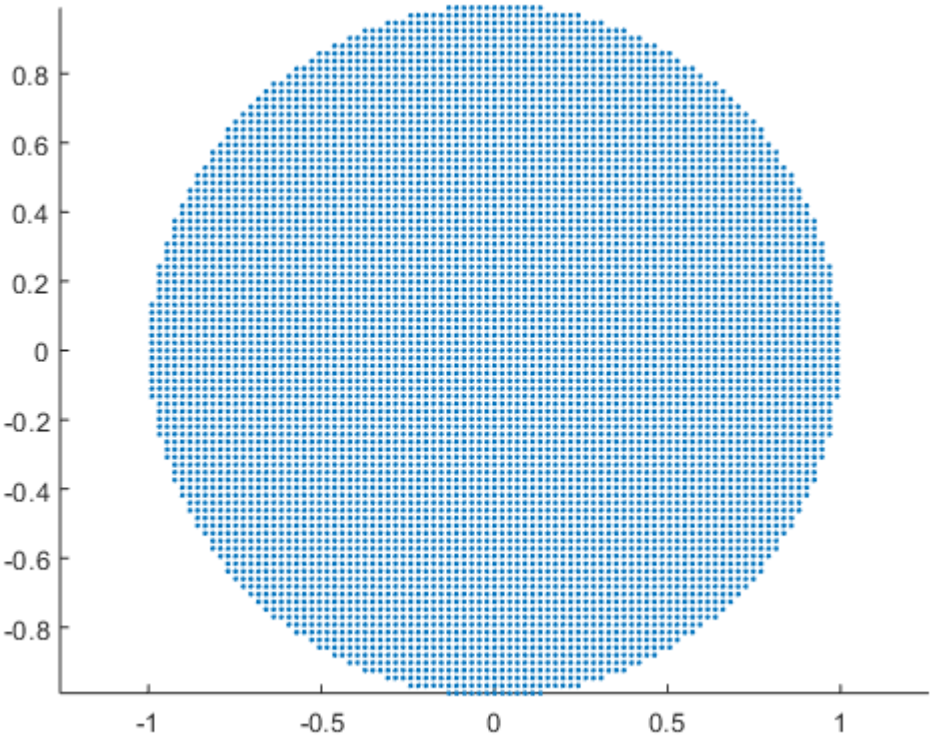
```
IEnumerable<string> query = from n in names
              where n.Contains ("a")
              orderby n.Length
              select n.ToUpper();
```


from
type-name
identifier
in
enumerable-expr
,
orderby
expr
ascending
descending
query continuation
SelectMany
orderby-clause
where boolean-expr
from-clause
let identifier = expr
join-clause
select expr
group-clause
identifier
into
group expr by expr
join inner identifier in inner expr on outer key equals inner key
into identifier

Příklad

```
double l=1.1, r=1;
var n = 100;
var e = Enumerable.Range(0,n+1);
var h=2*l/n;
var x = from i in e select -l+i*h;
var grid = from cx in x
       from cy in x
       select new {x=cx,y=cy};
var domain = from p in grid
       where Norm(p.x,p.y) <= r
       select p;

double Norm(double x, double y) => Sqrt(Pow(x,2)+Pow(y,2));
```


0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-0.5
0
0.5
1

Příklad

```
namespace ME
{
    public struct Coordinate
    {
        public int id;
        public double value;
        public bool boundary;
        public override string ToString()
        {
            return string.Format($"{nameof(Coordinate)}( id={id}, boundary={
                boundary}, value={value} )");
        }
    }

    public struct Point
    {
        public int id, idx, idy;
        public bool boundary;
        public override string ToString()
        {
            return string.Format($"{nameof(Point)}( id={id} idx={idx} idy={idy}
                boundary={boundary})");
        }
    }

    public struct Point3
```

```
    {
        public int id, idx, idy, idz;
        public bool boundary;
        public override string ToString()
        {
            return string.Format($"{nameof(Point)}( id={id} idx={idx} idy={idy} idz
                ={idz} boundary={boundary})");
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ME
{
    public static class Ex
    {
        public static void Dump<T>(this IEnumerable<T> collection, string name, int
            max = 5)
        {
            int n = collection.Count();
            IEnumerable<string> s;
            if (n > 2 * max)
                s = collection
                    .Take(max).Select((a, i) => string.Format($"{name}[{i}] = {a}")
```

```
                )
            .Append("...")
            .Concat(collection.Skip(n - max).Select((a, i) => string.Format
                ($"{name}[{n - max + i}] = {a}")));
    else
        s = collection.Select((a, i) => string.Format($"{name}[{i}] = {a}")
            );
    foreach (var item in s)
    {
        Console.WriteLine(item);
    }
}

public static IEnumerable<Coordinate> LinSpace(double a, double b, int n)
{
    n--;
    return
    Enumerable.Range(0, n).Select((r, i) => new Coordinate { id = i,
        boundary = i == 0, value = a + r * (b - a) / n }).
        Append(new Coordinate { id = n, boundary = true, value = b });
}

public static IEnumerable<Point> Grid(IEnumerable<Coordinate> x,
    IEnumerable<Coordinate> y)
{
    var xy = from px in x
             from py in y
             select new { idx = px.id, idy = py.id, bdry = px.boundary ||
                 py.boundary };
```

```
    return xy.Select((a, i) => new Point { id = i, idx = a.idx, idy = a.idy
        , boundary = a.bdry });
}

public static IEnumerable<Point3> Grid(IEnumerable<Coordinate> x,
    IEnumerable<Coordinate> y, IEnumerable<Coordinate> z)
{
    var xyz = from px in x
              from py in y
              from pz in z
              select new { idx = px.id, idy = py.id, idz = pz.id, bdry = px
                  .boundary || py.boundary || pz.boundary };
    return xyz.Select((a, i) => new Point3 { id = i, idx = a.idx, idy = a.
        idy, idz = a.idz, boundary = a.bdry });
}

public static IEnumerable<double[]> Grid(IEnumerable<Coordinate> x,
    IEnumerable<Coordinate> y, Func<double, double, double> f)
{
    return
    from px in x
    from py in y
    select new double[] { px.value, py.value, f(px.value, py.value) };
}

public static IEnumerable<double[]> Grid(IEnumerable<Coordinate> x,
    IEnumerable<Coordinate> y, IEnumerable<Coordinate> z, Func<double,
    double, double, double> f)
{
```

```
            return
            from px in x
            from py in y
            from pz in z
            select new double[] { px.value, py.value, pz.value, f(px.value, py.
                value, pz.value) };
        }

    }
}
```

```
using System;
using System.Linq;
using MathNet.Numerics.LinearAlgebra;

namespace ME
{
    class Program
    {
        static void Main(string[] args)
        {
            int n = 6, m = 11;
            var x = Ex.LinSpace(0, 2, n);
            var y = Ex.LinSpace(0, 1, m);
            x.Dump(nameof(x));
            y.Dump(nameof(y));
            var grid = Ex.Grid(x, y);
```

```
            grid.Dump(nameof(grid));
            var B = Matrix<double>.Build.Dense(n,m);
            foreach (var p in grid)
            {
                B[p.idx, p.idy] = p.boundary ? 1 : 0;
            }
            Console.WriteLine(B);
            var xa = x.ToArray();
            var ya = y.ToArray();
            var xyz = Ex.Grid(x,y,(a,b)=>a+b);
            var XYZ=Matrix<double>.Build.DenseOfRows(xyz);
            Console.WriteLine(XYZ);
        }
    }
}
```

```
x[0] = Coordinate( id=0, boundary=True, value=0 )
x[1] = Coordinate( id=1, boundary=False, value=0.4 )
x[2] = Coordinate( id=2, boundary=False, value=0.8 )
x[3] = Coordinate( id=3, boundary=False, value=1.2 )
x[4] = Coordinate( id=4, boundary=False, value=1.6 )
x[5] = Coordinate( id=5, boundary=True, value=2 )
y[0] = Coordinate( id=0, boundary=True, value=0 )
y[1] = Coordinate( id=1, boundary=False, value=0.1 )
y[2] = Coordinate( id=2, boundary=False, value=0.2 )
y[3] = Coordinate( id=3, boundary=False, value=0.3 )
y[4] = Coordinate( id=4, boundary=False, value=0.4 )
...
y[6] = Coordinate( id=6, boundary=False, value=0.6 )
```

```
y[7] = Coordinate( id=7, boundary=False, value=0.7 )
y[8] = Coordinate( id=8, boundary=False, value=0.8 )
y[9] = Coordinate( id=9, boundary=False, value=0.9 )
y[10] = Coordinate( id=10, boundary=True, value=1 )
grid[0] = Point( id=0 idx=0 idy=0 boundary=True)
grid[1] = Point( id=1 idx=0 idy=1 boundary=True)
grid[2] = Point( id=2 idx=0 idy=2 boundary=True)
grid[3] = Point( id=3 idx=0 idy=3 boundary=True)
grid[4] = Point( id=4 idx=0 idy=4 boundary=True)
...
grid[61] = Point( id=61 idx=5 idy=6 boundary=True)
grid[62] = Point( id=62 idx=5 idy=7 boundary=True)
grid[63] = Point( id=63 idx=5 idy=8 boundary=True)
grid[64] = Point( id=64 idx=5 idy=9 boundary=True)
grid[65] = Point( id=65 idx=5 idy=10 boundary=True)
DenseMatrix 6x11-Double
1  1  1  1  1  1  1  1  1  1  1
1  0  0  0  0  0  0  0  0  0  1
1  0  0  0  0  0  0  0  0  0  1
1  0  0  0  0  0  0  0  0  0  1
1  0  0  0  0  0  0  0  0  0  1
1  1  1  1  1  1  1  1  1  1  1

DenseMatrix 66x3-Double
 0    0    0
 0  0.1  0.1
 0  0.2  0.2
 0  0.3  0.3
 0  0.4  0.4
```

```
 0   0.5   0.5
 0   0.6   0.6
 0   0.7   0.7
..    ..    ..
 2   0.7   2.7
 2   0.8   2.8
 2   0.9   2.9
 2     1     3
```

Join (příklad z MSDN dokumentace)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ME
{
    public class JoinDemonstration
    {

        #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="nápoje", ID=001},
      new Category(){Name="koření", ID=002},
      new Category(){Name="zelenina", ID=003},
      new Category(){Name="obiloviny", ID=004},
      new Category(){Name="ovoce", ID=005}
  };

    // Specify the second data source.
    List<Product> products = new List<Product>()
 {
     new Product{Name="káva",  CategoryID=001},
     new Product{Name="čaj",  CategoryID=001},
     new Product{Name="pepř", CategoryID=002},
     new Product{Name="kmín", CategoryID=002},
     new Product{Name="mrkev", CategoryID=003},
     new Product{Name="zelí", CategoryID=003},
     new Product{Name="jablka", CategoryID=005},
```

```
    new Product{Name="hrušky", CategoryID=005},
};
#endregion


public static void Example()
{
    JoinDemonstration app = new JoinDemonstration();

    app.InnerJoin();
    app.GroupJoin();
    app.GroupInnerJoin();
    app.GroupJoin3();
    app.LeftOuterJoin();
    app.LeftOuterJoin2();

}

void InnerJoin()
{
    var innerJoinQuery =
       from category in categories
       join prod in products on category.ID equals prod.CategoryID
       select new { Category = category.ID, Product = prod.Name };

    Console.WriteLine("InnerJoin:");
    foreach (var item in innerJoinQuery)
    {
        Console.WriteLine("{0,-10}{1}", item.Product, item.Category);
```

```
    }
    Console.WriteLine("InnerJoin: {0} items in 1 group.", innerJoinQuery.
        Count());
    Console.WriteLine(System.Environment.NewLine);

}

void GroupJoin()
{
    var groupJoinQuery =
        from category in categories
        join prod in products on category.ID equals prod.CategoryID into
            prodGroup
        select prodGroup;


    int totalItems = 0;
    Console.WriteLine("Simple GroupJoin:");
    foreach (var prodGrouping in groupJoinQuery)
    {
        Console.WriteLine("Group:");
        foreach (var item in prodGrouping)
        {
            totalItems++;
            Console.WriteLine("   {0,-10}{1}", item.Name, item.CategoryID);
        }
    }
    Console.WriteLine("Unshaped GroupJoin: {0} items in {1} unnamed groups"
        , totalItems, groupJoinQuery.Count());
```

```
    Console.WriteLine(System.Environment.NewLine);
}

void GroupInnerJoin()
{
    var groupJoinQuery2 =
        from category in categories
        orderby category.ID
        join prod in products on category.ID equals prod.CategoryID into
           prodGroup
        select new
        {
            Category = category.Name,
            Products = from prod2 in prodGroup
                       orderby prod2.Name
                       select prod2
        };


    int totalItems = 0;

    Console.WriteLine("GroupInnerJoin:");
    foreach (var productGroup in groupJoinQuery2)
    {
        Console.WriteLine(productGroup.Category);
        foreach (var prodItem in productGroup.Products)
        {
            totalItems++;
```

```
            Console.WriteLine("  {0,-10} {1}", prodItem.Name, prodItem.
                CategoryID);
        }
    }
    Console.WriteLine("GroupInnerJoin: {0} items in {1} named groups",
        totalItems, groupJoinQuery2.Count());
    Console.WriteLine(System.Environment.NewLine);
}

void GroupJoin3()
{

    var groupJoinQuery3 =
        from category in categories
        join product in products on category.ID equals product.CategoryID
            into prodGroup
        from prod in prodGroup
        orderby prod.CategoryID
        select new { Category = prod.CategoryID, ProductName = prod.Name };

    int totalItems = 0;

    Console.WriteLine("GroupJoin3:");
    foreach (var item in groupJoinQuery3)
    {
        totalItems++;
        Console.WriteLine("   {0}:{1}", item.ProductName, item.Category);
    }
```

```
    Console.WriteLine("GroupJoin3: {0} items in 1 group", totalItems,
        groupJoinQuery3.Count());
    Console.WriteLine(System.Environment.NewLine);
}

void LeftOuterJoin()
{
    var leftOuterQuery =
        from category in categories
        join prod in products on category.ID equals prod.CategoryID into
            prodGroup
        select prodGroup.DefaultIfEmpty(new Product() { Name = "Nic!",
            CategoryID = category.ID });

    int totalItems = 0;

    Console.WriteLine("Left Outer Join:");

    foreach (var prodGrouping in leftOuterQuery)
    {
        Console.WriteLine("Group:", prodGrouping.Count());
        foreach (var item in prodGrouping)
        {
            totalItems++;
            Console.WriteLine("  {0,-10}{1}", item.Name, item.CategoryID);
        }
    }
    Console.WriteLine("LeftOuterJoin: {0} items in {1} groups", totalItems,
        leftOuterQuery.Count());
```

```
        Console.WriteLine(System.Environment.NewLine);
    }

    void LeftOuterJoin2()
    {
        var leftOuterQuery2 =
           from category in categories
           join prod in products on category.ID equals prod.CategoryID into
              prodGroup
           from item in prodGroup.DefaultIfEmpty()
           select new { Name = item == null ? "Nic!" : item.Name, CategoryID =
              category.ID };

        Console.WriteLine("LeftOuterJoin2: {0} items in 1 group",
           leftOuterQuery2.Count());

        int totalItems = 0;

        Console.WriteLine("Left Outer Join 2:");


        foreach (var item in leftOuterQuery2)
        {
            totalItems++;
            Console.WriteLine("{0,-10}{1}", item.Name, item.CategoryID);
        }
        Console.WriteLine("LeftOuterJoin2: {0} items in 1 group", totalItems);
    }
}
```

```
    /*
    InnerJoin:
káva      1
čaj       1
pepř      2
kmín      2
mrkev     3
zelí      3
jablka    5
hrušky    5
InnerJoin: 8 items in 1 group.


Simple GroupJoin:
Group:
   káva      1
   čaj       1
Group:
   pepř      2
   kmín      2
Group:
   mrkev     3
   zelí      3
Group:
Group:
   jablka    5
   hrušky    5
Unshaped GroupJoin: 8 items in 5 unnamed groups
```

```
GroupInnerJoin:
nápoje
  čaj         1
  káva        1
koření
  kmín        2
  pepř        2
zelenina
  mrkev       3
  zelí        3
obiloviny
ovoce
  hrušky      5
  jablka      5
GroupInnerJoin: 8 items in 5 named groups


GroupJoin3:
   káva:1
   čaj:1
   pepř:2
   kmín:2
   mrkev:3
   zelí:3
   jablka:5
   hrušky:5
GroupJoin3: 8 items in 1 group
```

```
Left Outer Join:
Group:
  káva       1
  čaj        1
Group:
  pepř       2
  kmín       2
Group:
  mrkev      3
  zelí       3
Group:
  Nic!       4
Group:
  jablka     5
  hrušky     5
LeftOuterJoin: 9 items in 5 groups


LeftOuterJoin2: 9 items in 1 group
Left Outer Join 2:
káva       1
čaj        1
pepř       2
kmín       2
mrkev      3
zelí       3
Nic!       4
```

```
jablka     5
hrušky     5
LeftOuterJoin2: 9 items in 1 group
 */
}
```
