به نام خداوند گسترده مهر مهربان

آشنایی با LINQ

ترجمه: مهدی محبیان

منبع: MSDN

استفاده از این کتاب با ذکر یک صلوات رایگان است.

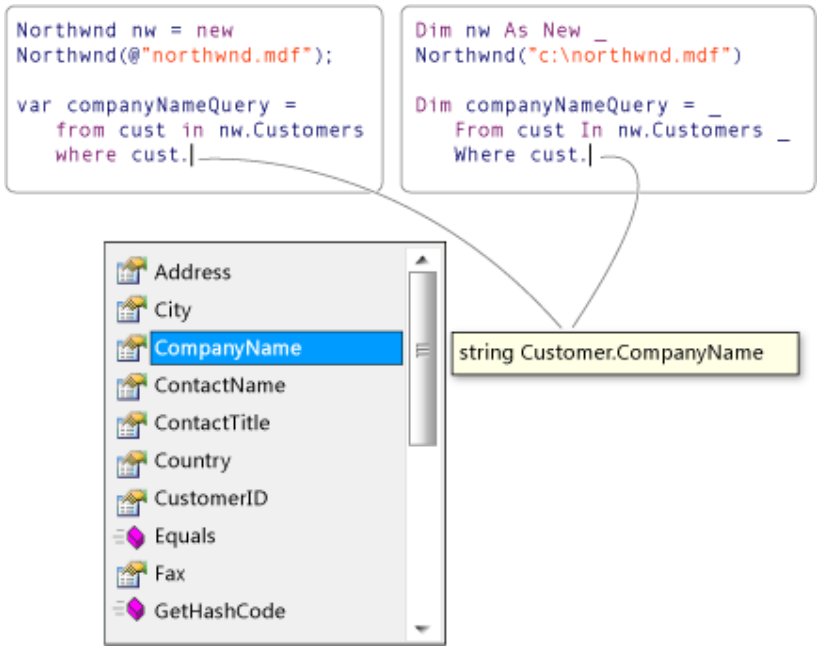
فهرست مطالب

مقـــدمه

آشنایی با LINQ

LINQ یک نوآوری در ویژوال استودیو ۲۰۰۸ و .NET Framework نسخه 3.5 است که مابین دنیای اشیاء و دنیای داده‌ها پل می‌زند.

پرس و جوهای سنتی در قبال داده‌ها به صورت رشته‌های ساده و بدون بررسی نوع در زمان کامپایل و یا بدون پشتیبانی IntelliSense بیان می‌شوند. علاوه بر این، شما باید برای هر یکی از انواع داده یک زبان پرس‌وجوی متفاوت را یاد بگیرید: پایگاه‌های داده SQL، اسناد XML، سرویس‌های وب متعدد و الی‌آخر. LINQ یک پرس‌وجو (*Query*) را یک ساختمان زبانی درجه یک در C# و ویژوال بیسیک می‌سازد. با استفاده از واژه‌های کلیدی و عملگرهای آشنا پرس‌وجوهایی را در قبال مجموعه‌هایی که به شکل قوی نوعدار شده‌اند خواهید نوشت. شکل زیر یک پرس‌وجوی نیمه تمام از یک پایگاه داده SQL Server را در C# نشان می‌دهد که از بررسی نوع کامل و پشتیبانی IntelliSense برخوردار است.



در ویژوال استودیو می‌توانید پرس‌وجوهای LINQ را در ویژوال بیسیک یا #C با پایگاه‌های داده SQL Server، اسناد XML، Datasetهای ADO.NET و هر مجموعه‌ای از اشیاء که از واسط IEnumerable یا واسط جنریک IEnumerable(T) پشتیبانی می‌کنند بنویسید. پشتیبان LINQ برای ADO.NET Entity Framework نیز طراحی شده است و تأمین کننده‌های LINQ توسط اشخاص ثالث برای بسیاری از سرویس‌های وب و دیگر پیاده‌سازی‌های پایگاه داده نوشته می‌شوند.

شما می‌توانید پرس‌وجوهای LINQ را در پروژه‌های جدید، یا در کنار پرس‌وجوهای غیر LINQ در پروژه‌های موجود بنویسید. تنها شرط لازم این است که پروژه از .NET Framework نسخه 3.5 و بالاتر استفاده کند.

آشنایی با پرس‌وجوهای LINQ

یک پرس‌وجو (Query) عبارتی است که داده را از یک منبع داده بازیابی می‌کند. پرس‌وجوها معمولاً در یک زبان پرس‌وجوی اختصاصی بیان می‌شوند. زبان‌های مختلف در طی زمان برای انواع متعددی از منابع داده توسعه داده شده‌اند، برای مثال SQL برای پایگاه‌های داده‌ی رابطه‌ای و XQuery برای XML. ازاینرو، توسعه دهندگان لازم بود تا برای هر نوعی از منبع داده یا فرمت داده‌ای که آنها بایست پشتیبانی می‌کردند یک زبان پرس‌وجوی جدید را یاد بگیرند. LINQ این وضعیت را با ارائه یک مدل پایدار و سازگار برای کار کردن با داده‌ها در میان انواع متعددی از منابع و فرمت‌های داده ساده می‌کند. در یک پرس‌وجوی LINQ، همواره با اشیاء کار می‌کنید. از الگوهای کدنویسی یکسانی به منظور پرس‌وجو و انتقال داده در اسناد XML، پایگاه‌های داده SQL، Datasetهای ADO.NET، مجموعه‌های .NET و هر فرمت دیگری که یک تأمین کننده LINQ برای آن در دسترس است استفاده می‌کنید.

سه بخش یک عملیات پرس‌وجو

تمامی عملیات‌های پرس‌وجو از عمل مجزا تشکیل می‌شوند:

۱. بدست آوردن منبع داده.

۲. ایجاد پرس‌وجو.

۳. اجرای پرس‌وجو.

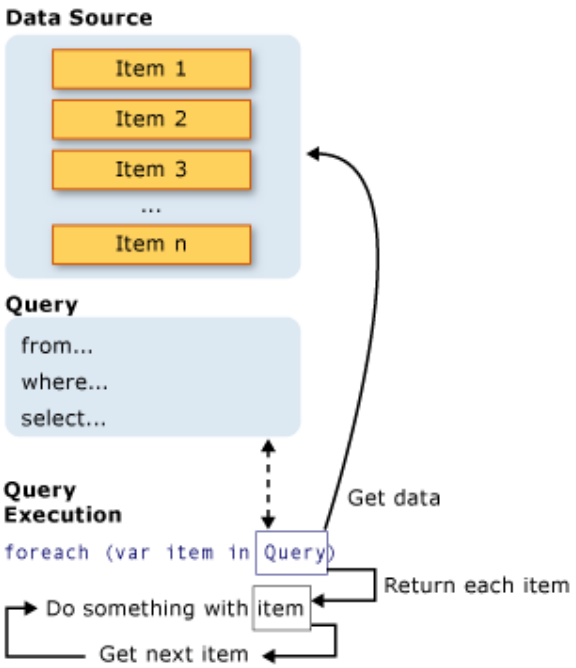
مثال زیر نشان می‌دهد که چگونه سه بخش یک عملیات پرس‌وجو در کد منبع بیان می‌شوند. این مثال برای راحتی کار آرایه‌ای از مقادیر صحیح را به عنوان یک منبع داده مورد استفاده قرار می‌دهد؛ اگرچه، مفاهیم یکسانی به منابع داده‌ی دیگر نیز اعمال می‌شود. این مثال تا انتهای این بخش مورد اشاره واقع خواهد شد.

کد نمونه: **C#**

```
class IntroToLINQ
{
    static void Main()
    {
        // The Three Parts of a LINQ Query:
        //  1. Data source.
```

```
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        // 2. Query creation.
        // numQuery is an IEnumerable<int>
        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;

        // 3. Query execution.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

شکل زیر عملیات پرس‌وجوی کامل را نشان می‌دهد. در LINQ، اجرای پرس‌وجو از خود پرس‌وجو مجزا است؛ به عبارت دیگر تنها با ایجاد یک متغیر پرس‌وجو هیچ گونه داده‌ای را بازیابی نکرده‌اید.



The Data Source

منبع داده

در مثال قبل، از آن جایی که منبع داده یک آرایه است، به صورت ضمنی از واسط جنریک IEnumerable(T) پشتیبانی می‌کند. این حقیقیت به معنای این است که این منبع داده می‌تواند بوسیله LINQ مورد جستار واقع شود.

یک پرس‌وجو در یک دستور **foreach** اجرا می‌شود، و **foreach** نیازمند IEnumerable یا IEnumerable(T) است. انواعی که از IEnumerable(T) یا یک واسط مشتق شده مانند واسط جنریک IQueryable(T) پشتیبانی می‌کنند انواع قابل پرس‌وجو خوانده می‌شوند.

یک نوع قابل پرس وجو شدن برای این که به عنوان یک منبع داده LINQ عمل کند به هیچ گونه تغییر یا رفتار بخصوصی نیاز ندارد. اگر داده منبع پیش از این در حافظه به عنوان یک نوع قابل پرس وجو شدن قرار نگرفته باشد، تأمین کننده‌ی LINQ باید آن را به چنان صورتی بیان کند. برای مثال، LINQ به XML یک سند XML را به میان یک نوع قابل پرس‌وجو شدن XElement بارگذاری می‌کند:

کد نمونه: **C#**

```
// Create a data source from an XML document.
// using System.Xml.Linq;
XElement contacts = XElement.Load(@"c:\myContactList.xml");
```

با LINQ به SQL، نخست یک نگاشت رابطه‌ای–شیئی را در زمان طراحی یا به شکل دستی و یا با استفاده از Object Relational Designer (O/R Designer) ایجاد خواهید کرد. شما پرس‌وجوهای خود را در قبال اشیاء می‌نویسید و در زمان اجرا LINQ به SQL ارتباطات با پایگاه داده را مدیریت می‌کند. در مثال زیر، `Customer` بیانگر یک جدول به خصوص در پایگاه داده است و `Table<Customer>` از نوع جنریک IQueryable(T) که از IEnumerable(T) مشتق می‌شود پشتیبانی می‌کند.

کد نمونه: **C#**

```
// Create a data source from a SQL Server database.
// using System.Data.Linq;
DataContext db = new DataContext(@"c:\northwind\northwnd.mdf");
```

قاعده اصلی خیلی ساده است: یک منبع داده LINQ هر شیئی است که از واسط جنریک IEnumerable(T) یا واسطی که از آن ارث می‌برد پشتیبانی کند.

نکته:

انواعی مانند ArrayList که از واسط غیرجنریک IEnumerable پشتیبانی می‌کنند نیز می‌توانند به عنوان یک

منبع داده LINQ مورد استفاده واقع شوند.

پرس‌وجو

پرس وجو تعیین می‌کند که چه اطلاعاتی از منبع یا منابع داده بازیابی شوند. در ضمن یک پرس‌وجو به صورت اختیاری تعیین می‌کند که اطلاعات قبل از برگردانده شدن چگونه ذخیره، گروه‌بندی و شکل داده شوند. یک پرس‌وجو در یک متغیر پرس‌وجو ذخیره شده و با یک عبارت پرس‌وجو مقداردهی اولیه می‌شود. برای آسان‌تر ساختن نوشتن پرس‌وجوها، C# ترکیب نوشتاری پرس‌وجوی جدیدی را معرفی کرده است.

پرس‌وجوی واقع در مثال قبل تمامی اعداد زوج واقع در آرایه را برمی‌گرداند. عبارت پرس‌وجو شامل سه ضابطه است: **from**، **where** و **select**. (اگر با SQL آشنا باشید متوجه خواهید شد که ترتیب ضابطه‌ها برعکس ترتیب ضابطه‌ها در SQL است.) ضابطه‌ی **from** منبع داده را مشخص می‌کند، ضابطه‌ی **where** فیلتر را اعمال می‌کند و ضابطه‌ی **select** نوع عناصر برگشتی را مشخص می‌کند. اینها و دیگر ضابطه‌های پرس‌وجو با جزئیات بیشتر در بخش عبارات پرس‌وجوی LINQ مورد بحث قرار می‌گیرند. فی‌الحال مهم‌ترین نکته این است که در LINQ، متغیر پرس‌وجو خودش هیچ عملی را اتخاذ نکرده و هیچ گونه داده‌ای را برنمی‌گرداند. این متغیر تنها اطلاعاتی را ذخیره می‌کند که لازم است هنگام اجرای پرس‌وجو در برخی نقاط آتی تولید شوند.

نکته:

پرس‌وجوها می‌توانند با استفاده از ترکیب نوشتاری متد نیز بیان شوند.

- ترکیب نوشتاری پرس‌وجو:

```
//Query syntax:
IEnumerable<int> numQuery1 =
  from num in numbers
  where num % 2 == 0
  orderby num
  select num;
```

- ترکیب نوشتاری متد معادل با ترکیب نوشتاری پرس‌وجو:

```
  //Method syntax:
IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 == 0).OrderBy(n => n);
```

اجرای پرس‌وجو

اجرای معوق

همان طور که پیش از گفته شد متغیر پرس‌وجو خودش تنها فرامین پرس‌وجو را دخیره می‌کند. اجرای واقعی پرس‌وجو به تعویق می‌افتد تا این که متغیر پرس‌وجو را در یک دستور **foreach** برای حرکت بر روی نتایج پرس‌وجو به کار ببرید. این مفهوم با عنوان *اجرای معوق* شناخته می‌شود و در مثال زیر نشان داده می‌شود:

کد نمونه: **C#**

```
//  Query execution.
foreach (int num in numQuery)
{
    Console.Write("{0,1} ", num);
}
```

دستور **foreach** نیز جایی است که نتایج پرس‌وجو بازیابی می‌شوند. برای مثال، در پرس‌وجوی قبلی، متغیر تکرار `num` هر یک از مقادیر (در هر زمانی یکی) واقع در دنباله برگشتی را نگهداری می‌کند.

از آن جایی که متغیر پرس‌وجو خودش هرگز نتایج پرس‌وجو را نگهداری نمی‌کند، شما می‌توانید آن را هر زمان که خواستید اجرا کنید. برای مثال، ممکن است دارای پایگاه داده‌ای باشید که به صورت پیوسته توسط یک برنامه مجزا بروزرسانی می‌شود. شما در برنامه‌ی خود می‌توانید پرس‌وجویی را ایجاد کنید که آخرین داده را بازیابی کند و می‌توانید آن را به صورت مکرر در برخی از بازه‌های زمانی اجرا کنید تا هر بار نتایج متفاوتی را بازیابی کنید.

تحمیل اجرای فوری و بی‌واسطه

پرس‌وجوهایی که اعمال ترکیبی را بر روی گستره‌ای از عناصر انجام می‌دهند باید نخست روی این عناصر حرکت کنند. **Count**، **Max**، **Average** و **First** نمونه‌ای از چنین پرس‌وجوهایی هستند. این پرس‌وجوها بدون یک دستور صریح **foreach** اجرا می‌شوند زیرا خود پرس‌وجو بایستی از **foreach** استفاده کند تا این که نتیجه‌ای را برگرداند. درضمن توجه داشته باشید که این نوع از پرس‌وجوها یک مقدار یکتا را برمی‌گردانند نه یک مجموعه‌ی **IEnumerable** را. پرس‌وجوی زیر تعداد اعداد زوج واقع در آرایه‌ی منبع را برمی‌گرداند:

کد نمونه: **C#**

```
var evenNumQuery =
    from num in numbers
    where (num % 2) == 0
    select num;

int evenNumCount = evenNumQuery.Count();
```

برای تحمیل اجرای فوری هر گونه پرس‌وجو و نهان کردن نتایج آن، می‌توانید متدهای ToList(TSource) یا ToArray(TSource) را فراخوانی کنید.

کد نمونه: **C#**

```
List<int> numQuery2 =
    (from num in numbers
     where (num % 2) == 0
     select num).ToList();

// or like this:
// numQuery3 is still an int[]

var numQuery3 =
    (from num in numbers
     where (num % 2) == 0
     select num).ToArray();
```

در ضمن می‌توانید با جای دادن حلقه‌ی **foreach** بلافاصله بعد از عبارت پرس‌وجو، اجرای پرس‌وجو را جلو بیندازید. اگرچه، با فراخوانی متدهای **ToList** و **ToArray** می‌توانید تمامی داده را در یک شیء مجموعه یکتا نهان کنید.

LINQ و انواع جنریک

پرس‌وجوهای LINQ مبتنی بر انواع جنریک هستند انواعی که در نسخه‌ی 2.0 از .NET Framework معرفی شده‌اند. قبل از این که بتوانید شروع به نوشتن پرس‌وجوها کنید نیازی به دانش عمیق درباره‌ی انواع جنریک ندارید. اگرچه، ممکن است بخواهید تا با این دو مفهوم پایه‌ای آشنا شوید:

۱. هرگاه نمونه‌ای از یک کلاس مجموعه جنریک مانند List(T) را ایجاد کردید، "T" را با نوع اشیائی که لیست نگهداری خواهد کرد جایگزین کنید. برای مثال، لیستی از رشته‌ها به صورت `List<string>` بیان می‌شود و لیستی از اشیاء `Customer` به صورت `List<Customer>` بیان می‌شود. یک لیست جنریک به شکل قدرتمندی نوعدار شده است و نسبت به مجموعه‌هایی که عناصر خود را به صورت Object ذخیره می‌کنند مزایای بیشتری را ارائه می‌کند. اگر سعی کنید تا یک `Customer` را به یک `List<string>` اضافه کنید، در زمان کامپایل با یک خطا مواجه خواهید شد. استفاده از مجموعه‌های جنریک آسان است زیرا لازم نیست تا تبدیل نوع صریح زمان اجرا را انجام دهید.

۲. واسط جنریک IEnumerable(T) واسطی است که کلاس‌های مجموعه جنریک را قادر می‌سازد تا با استفاده از دستور **foreach** شمارش شوند. کلاس‌های مجموعه جنریک از واسط IEnumerable(T) پشتیبانی می‌کنند در حالی که کلاس‌های مجموعه غیرجنریک مانند ArrayList از IEnumerable پشتیبانی می‌کنند.

متغیرهای IEnumerable(T) در پرس‌وجوهای LINQ

متغیرهای پرس‌وجوی LINQ به صورت IEnumerable(T) یا یک نوع مشتق شده مانند IQueryable(T) نوعدار می‌شوند. هرگاه متغیر پرس‌وجویی را دیدید که به صورت `IEnumerable<Customer>` نوعدار شده است تنها به معنای این است که پرس‌وجو، هنگام اجرا شدنش، دنباله‌ای از صفر یا چند شیء `Customer` را تولید خواهد کرد.

کد نمونه: **C#**

```
IEnumerable<Customer> customerQuery =
    from cust in customers
    where cust.City == "London"
    select cust;

foreach (Customer customer in customerQuery)
{
    Console.WriteLine(customer.LastName + ", " + customer.FirstName);
}
```

دادن اجازه‌ی مدیریت اعلان نوعهای جنریک به کامپایلر

اگر ترجیح می‌دهید می‌توانید با استفاده از واژه کلیدی **var** از ترکیب نوشتاری نوع جنریک اجتناب کنید. واژه کلیدی **var** به کامپایلر دستور می‌دهد تا با نگاه به منبع داده‌ی مشخص شده در ضابطه‌ی **from** نوع یک متغیر پرس‌وجو را استنتاج کند. مثال زیر کد کامپایل شده‌ای مانند مثال قبل تولید می‌کند:

C# کد نمونه:

```
var customerQuery2 =
    from cust in customers
    where cust.City == "London"
    select cust;

foreach(var customer in customerQuery2)
{
    Console.WriteLine(customer.LastName + ", " + customer.FirstName);
}
```

واژه کلیدی **var** زمانی مفید است که نوع متغیر واضح باشد یا زمانی که اهمیتی ندارد که انواع جنریک تودرتو مانند آنهایی که با استفاده از پرس‌وجوهای گروهی تولید می‌شوند به شکل صریح مشخص شوند. به طور کلی، توصیه می‌شود که اگر از **var** استفاده می‌کنید، یادتان باشد که این کار خواندن کد شما را برای دیگران مشکل می‌سازد.

عملیات‌های پایه پرس‌وجو (در LINQ)

این بخش شما را به طور مختصر با عبارات پرس‌وجوی LINQ و برخی از انواع معمول عملیات‌هایی که در یک پرس وجو انجام خواهید داد آشنا می‌کند.

بدست آوردن یک منبع داده

در یک پرس‌وجوی LINQ، گام نخست مشخص کردن منبع داده است. در C# همانند اغلب زبان‌های برنامه‌نویسی، یک متغیر قبل از این که بتواند مورد استفاده واقع گردد باید اعلان شود. در یک پرس‌وجوی

LINQ، ضابطه‌ی **from** از نظر ترتیبی در ابتدا می‌آید تا منبع داده (مثلاً `customers`) و متغیر *دامنه* (`cust`) را معرفی کند.

کد نمونه: **C#**

```
//queryAllCustomers is an IEnumerable<Customer>
var queryAllCustomers = from cust in customers
                        select cust;
```

متغیر دامنه شبیه متغیر کنترل حلقه در یک حلقه‌ی **foreach** است به جز این که هیچ گونه تکرار واقعی در یک عبارت پرس‌وجو رخ نمی‌دهد. هرگاه پرس‌وجو اجرا شود متغیر دامنه همانند یک ارجاع (یا اِسناد) به هر یک از عناصر بعدی واقع در `customers` عمل می‌کند. از آن جایی که کامپایلر می‌تواند نوع `cust` را استنتاج کند لزومی ندارد که به شکل صریح آن را مشخص کنید. متغیرهای دامنه اضافی می‌توانند بوسیله یک ضابطه‌ی **let** معرفی شوند.

نکته:

برای منابع داده‌ی غیرجنریک مانند ArrayList، متغیر دامنه باید صریحاً نوعدار شود.

فیلتر کردن

احتمالاً رایج‌ترین عملیات پرس‌وجو این خواهد بود که فیلتری را در قالب یک عبارت بولی اعمال کنید. فیلتر باعث می‌شود که پرس‌وجو تنها عناصری را برگرداند که برای آنها نتیجه ارزیابی عبارت صحیحی (**true**) است. نتیجه با استفاده از ضابطه **where** تولید می‌شود. فیلتر در واقع عناصری را مشخص می‌کند که باید از دنباله منبع حذف شوند. در مثال زیر، تنها آن `customers`هایی را که دارای آدرس London هستند برمی‌گرداند.

کد نمونه: **C#**

```
var queryLondonCustomers = from cust in customers
                           where cust.City == "London"
                           select cust;
```

می‌توانید عملگرهای **AND** و **OR** منطقی C# را برای اعمال بسیاری از عبارات فیلتر به هر اندازه‌ای که نیاز باشد در ضابطه‌ی **where** مورد استفاده قرار دهید. برای مثال، صرفاً برای برگرداندن customersای از "London" که اسمش "Devon" باشد باید کد زیر را بنویسید:

کد نمونه: **C#**

```
where cust.City=="London" && cust.Name == "Devon"
```

برای برگرداندن customersهایی از London یا Paris، کد زیر را خواهید نوشت:

کد نمونه: **C#**

```
where cust.City == "London" || cust.City == "Paris"
```

ضابطه‌ی **where** یک مکانیزم فیلترینگ است. این ضابطه تقریباً می‌تواند در هر جایی در یک عبارت پرس‌وجو جای داده شود به استثنای این که نمی‌تواند نخستین یا آخرین ضابطه باشد. ضابطه‌ی **where** می‌تواند بسته به این که شما بخواهید عناصر منبع را قبل یا بعد از گروه‌بندی کردن آنها فیلتر کنید قبل و یا بعد از یک ضابطه‌ی **group** ظاهر شود.

اگر یک اعلان مشخص برای عناصر واقع در منبع داده معتبر نباشد یک خطای زمان کامپایل بروز خواهد کرد. این کار یکی از مزایای بررسی نوع قوی ارائه شده توسط LINQ است.

در زمان کامپایل واژه کلیدی **where** تبدیل به یک فراخوان به عملگر پرس‌وجوی استاندارد **Where** می‌شود.

مرتب کردن

اغلب اوقات مرتب‌سازی و سوا کردن داده برگشتی مناسب خواهد بود. ضابطه‌ی **orderby** باعث خواهد شد تا عناصر واقع در دنباله برگشتی بر طبق ارزیاب پیش فرض برای نوع در حال مرتب شدن، مرتب شود. برای مثال، پرس‌وجوی زیر می‌تواند بسط داده شود تا نتایج را بر اساس خاصیت Name مرتب کند. از آن جایی که Name یک رشته است، ارزیاب پیش فرض یک جداسازی مبتنی بر الفبای از A تا Z انجام می‌دهد.

کد نمونه: **C#**

```
var queryLondonCustomers3 =
    from cust in customers
    where cust.City == "London"
    orderby cust.Name ascending
    select cust;
```

برای مرتب‌سازی نتایج در ترتیب معکوس، یعنی از Z تا A، از ضابطه‌ی `orderby…descending` استفاده کنید.

در یک عبارت پرس‌وجو، ضابطه‌ی **orderby** باعث می‌شود تا دنباله یا زیردنباله (گروه) برگشتی یا به صورت صعودی و یا به صورت نزولی مرتب شود. کلیدهای چندگانه می‌توانند مشخص شوند تا این که یک یا چند عملیات مرتب‌سازی ثانویه را انجام دهند. عمل مرتب‌سازی بوسیله ارزیاب پیش فرض برای نوع عنصر انجام می‌شود. ترتیب مرتب‌سازی پیش فرض صعودی است. شما می‌توانید یک ارزیاب سفارشی را نیز مشخص کنید. اگرچه، این کار تنها با استفاده از ترکیب نوشتاری مبتنی بر متد قابل دسترسی است.

در زمان کامپایل، ضابطه‌ی **orderby** تبدیل به یک فراخوان به متد OrderBy می‌شود. کلیدهای مضاعف در ضابطه‌ی **orderby** به فراخوانی‌های متد ThenBy ترجمه می‌شوند.

عملیات مرتب‌سازی عناصر یک دنباله را بر اساس یک یا چند مشخصه مرتب می‌کند. اولین قضاوت عملیات مرتب‌سازی یک مرتب‌سازی اولیه را روی عناصر انجام می‌دهد. با مشخص کردن یک قضاوت مرتب‌سازی ثانویه، شما می‌توانید عناصر واقع در میان هر یک از گروه‌های مرتب‌سازی اولیه را مرتب کنید.

شکل زیر نتایج یک عملیات مرتب‌سازی مبتنی بر الفبا را بر روی دنباله‌ای از کاراکترها نشان می‌دهد.

متدهای عملگر پرس‌وجوی استانداردی که داده‌ها را مرتب می‌کنند در جدول زیر لیست شده‌اند:

اسم متد	توضیح	عبارت پرس‌وجوی #C معادل
OrderBy	مقادیر را به ترتیب صعودی مرتب می‌کند.	`orderby`
OrderByDescending	مقادیر را به ترتیب نزولی مرتب می‌کند.	`orderby … descending`
ThenBy	یک مرتب‌سازی ثانویه را به ترتیب صعودی انجام می‌دهد.	`orderby …, …`
ThenByDescending	یک مرتب‌سازی ثانویه را به ترتیب نزولی انجام می‌دهد.	`orderby …, … descending`
Reverse	ترتیب عناصر واقع در یک مجموعه را برعکس می‌کند.	اعمال نمی‌شود.

نمونه‌هایی از مرتب‌سازی اولیه

مرتب‌سازی صعودی اولیه

مثال زیر چگونگی استفاده از ضابطه **orderby** را در یک پرس‌وجوی LINQ برای مرتب‌سازی رشته‌های واقع در یک آرایه به واسطه طول رشته، در ترتیب صعودی نشان می‌دهد.

کد نمونه: **C#**

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                            orderby word.Length
                            select word;

foreach (string str in query)
    Console.WriteLine(str);

/* This code produces the following output:

    the
    fox
    quick
    brown
    jumps
```

```
*/
```

مرتب‌سازی نزولی اولیه

مثال بعدی چگونگی استفاده از ضابطه‌ی **orderbydescending** را در یک پرس‌وجوی LINQ برای مرتب کردن رشته‌ها به واسطه حروف اول‌شان، در ترتیب نزولی نشان می‌دهد.

C# کد نمونه:

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                            orderby word.Substring(0, 1) descending
                            select word;

foreach (string str in query)
    Console.WriteLine(str);

/* This code produces the following output:

    the
    quick
    jumps
    fox
    brown
*/
```

نمونه‌هایی از مرتب‌سازی ثانویه

مرتب‌سازی ثانویه صعودی

مثال زیر چگونگی استفاده از ضابطه‌ی **orderby** را در یک پرس‌وجوی LINQ برای انجام یک عمل مرتب‌سازی اولیه و ثانویه‌ی رشته‌های واقع در یک آرایه نشان می‌دهد. رشته‌ها در ابتدا بواسطه‌ی طولشان مرتب شده و پس از آن به واسطه‌ی حرف نخست رشته مرتب می‌شوند؛ هر دو عمل مرتب‌سازی به شکل صعودی انجام می‌شود.

C# کد نمونه:

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                            orderby word.Length, word.Substring(0, 1)
                            select word;

foreach (string str in query)
    Console.WriteLine(str);

/* This code produces the following output:

    fox
    the
    brown
    jumps
    quick
*/
```

مرتب‌سازی نزولی ثانویه

مثال بعد چگونگی استفاده از ضابطه‌ی **orderbydescending** را در یک پرس‌وجوی LINQ برای انجام یک عمل مرتب‌سازی اولیه به شکل صعودی و یک عمل مرتب‌سازی ثانویه به شکل نزولی نشان می‌دهد. رشته‌ها نخست به واسطه‌ی طول‌شان مرتب شده و پس از آن به واسطه‌ی حرف نخست رشته مرتب می‌شوند.

C# کد نمونه:

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                            orderby word.Length, word.Substring(0, 1)
descending
                            select word;

foreach (string str in query)
    Console.WriteLine(str);

/* This code produces the following output:

    the
    fox
    quick
    jumps
    brown
*/
```

گروه‌بندی

ضابطه‌ی **group** شما را قادر می‌سازد تا نتایج خود را بر اساس کلیدی که مشخص می‌کنید گروه‌بندی نمایید. برای مثال، می‌توانید تصریح کنید که نتایج باید به واسطه‌ی `City` گروه‌بندی شوند طوری که همه‌ی مصرف کنندگان از London یا Paris در گروه‌های اختصاصی باشند. در این مورد، کلید `cust.City` است.

نکته:

در مثال‌های زیر انواع به صورت صریح اعلان شده‌اند تا بتوانند مفاهیم را به خوبی نشان دهند. شما می‌توانید مدیریت نوع ضمنی برای `custQuery`، `group` و `customer` استفاده کنید تا به کامپایلر اجازه دهید تا نوع دقیق را استنتاج کند.

کد نمونه: **C#**

```
// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>
  var queryCustomersByCity =
      from cust in customers
      group cust by cust.City;

  // customerGroup is an IGrouping<string, Customer>
  foreach (var customerGroup in queryCustomersByCity)
  {
      Console.WriteLine(customerGroup.Key);
      foreach (Customer customer in customerGroup)
      {
          Console.WriteLine("    {0}", customer.Name);
      }
  }
```

هرگاه یک پرس‌وجو را با یک ضابطه‌ی **group** به پایان می‌رسانید نتایج شما شکل لیستی از لیست‌ها را می‌گیرند. هریک از عناصر واقع در لیست شیئی است که دارای یک عضو `Key` و لیستی از عناصر است که تحت آن کلید گروه‌بندی می‌شوند. هرگاه روی نتایج یک پرس‌وجو که دنباله‌ای از گروه‌ها را تولید می‌کند حرکت کنید، باید از یک حلقه‌ی تودرتوی **foreach** استفاده کنید. حلقه‌ی بیرونی روی تک تک گروه‌ها حرکت کرده و حلقه‌ی درونی روی اعضای تک تک گروه‌ها حرکت می‌کند.

اگر لازم است تا به نتایج یک عملیات **group** اِسناد کنید شما می‌توانید از واژه کلیدی **into** استفاده کنید تا شناسه‌ای ایجاد شود که بتواند بیش از اینها مورد پرس‌وجو واقع گردد. پرس‌وجوی زیر تنها آن گروه‌هایی را برمی‌گرداند که شامل بیش از دو مصرف کننده هستند:

کد نمونه: **C#**

```
// custQuery is an IEnumerable<IGrouping<string, Customer>>
var custQuery =
    from cust in customers
    group cust by cust.City into custGroup
    where custGroup.Count() > 2
    orderby custGroup.Key
    select custGroup;
```

متصل کردن

عمل اتصال پیوندهایی مابین دنباله‌هایی ایجاد می‌کند که به طور صریح در منابع داده مدل نشده‌اند. برای مثال می‌توانید یک عمل اتصال (**join**) را اجرا کنید تا تمامی مصرف کنندگان واقع در London را که محصولات خود را به فروشندگانی که در Paris هستند سفارش می‌دهند پیدا کنید. در LINQ ضابطه‌ی **join** همواره به جای این که مستقیماً در مقابل جداول پایگاه داده عمل کند در مقابل مجموعه‌های شیئی عمل می‌کند. در LINQ لزومی ندارد که از **join** به همان اندازه‌ای که در SQL استفاده می‌کنید استفاده نمایید زیرا کلیدهای خارجی در LINQ در مدل شیئی به صورت خاصیت‌هایی که مجموعه‌ای از آیتم‌ها را نگهداری می‌کنند بیان می‌شوند. برای مثال، یک شیء `Customer` دربرگیرنده‌ی مجموعه‌ای از اشیاء `Order` است. به جای انجام یک عمل اتصال، با استفاده از نمادگذاری نقطه به سفارشات دسترسی پیدا می‌کنید:

کد نمونه:

```
from order in Customer.Orders...
```

ضابطه‌ی **join** برای مرتبط کردن عناصری از دنباله‌های منبع متفاوت که دارای هیچ گونه ارتباطی در مدل شیئی نیستند مفید است. تنها شرط لازم این است که عناصر واقع در هر یک از منابع مقدار یکسانی را که بتواند برای برابری ارزیابی شود به اشتراک بگذارند. برای مثال، یک توزیع کننده غذا می‌تواند دارای لیستی از فروشندگان

یک محصول بخصوص و لیستی از خریداران باشد. برای مثال یک ضابطه‌ی **join** می‌تواند برای ایجاد لیستی از فروشندگان و خریداران آن محصول که همگی در ناحیه‌ی مشخص یکسانی قرار دارند به کار برده شود.

ضابطه‌ی **join** دو دنباله‌ی منبع را به عنوان ورودی می‌پذیرد. عناصر واقع در هر یک از دنباله‌ها باید یا یک خاصیت باشد که بتواند با یک خاصیت متناظر واقع در دنباله‌ی دیگر ارزیابی شود یا شامل خاصیتی باشد که بتواند بتواند با یک خاصیت متناظر واقع در دنباله‌ی دیگر ارزیابی شود. ضابطه‌ی **join** با استفاده از واژه کلیدی **equals** کلیدهای مشخص شده را به منظور آزمایش برابری مقایسه می‌کند. همه‌ی اتصال‌های انجام شده توسط ضابطه‌ی **join** پیوند برابرند. شکل خروجی یک ضابطه‌ی **join** به نوع بخصوصی از اتصال که در حال انجامش هستید بستگی دارد. در زیر سه نوع خیلی رایج از انواع اتصال آورده شده است:

- اتصال درونی
- اتصال گروهی
- اتصال بیرونی چپ

گزینش کردن

ضابطه‌ی **select** نتایج پرس‌وجو را تولید می‌کند و «شکل» یا نوع هر یک از عناصر برگشتی را مشخص می‌کند. برای مثال، می‌توانید مشخص کنید که آیا نتایج شما از اشیاء `Customer` کامل، تنها یک عضو، زیرمجموعه‌ای از اعضا یا نتیجه‌ی کاملاً متفاوتی بر اساس یک محاسبه یا ایجاد شیء جدید تشکیل خواهد شد. هرگاه ضابطه‌ی **select** چیزی غیر از یک کپی از عنصر منبع تولید کند، عملیات *projection* خوانده می‌شود. کاربرد projectionها برای انتقال داده‌ها یک قابلیت قدرتمند عبارات پرس‌وجوی LINQ است.

عملگرهای پرس‌وجوی استاندارد

عملگرهای پرس‌وجوی استاندارد متدهایی هستند که قالب اصلی LINQ را شکل می‌دهند. بیشتر این متدها روی دنباله‌ها عمل می‌کنند جایی که یک دنباله شیئی است که نوعش واسط IEnumerable(T) یا واسط

IQueryable(T) را پیاده‌سازی می‌کند. عملگرهای پرس‌وجوی استاندارد قابلیت‌های پرس‌وجویی چون فیلترینگ، پروجکشن، گردهم‌آوری، مرتب‌سازی و غیره را در اختیار می‌گذارند.

دو گروه از عملگرهای پرس‌وجوی استاندارد LINQ وجود دارد، یکی که روی اشیائی از نوع IEnumerable(T) عمل می‌کند و دیگری که روی اشیائی از نوع IQueryable(T) عمل می‌کند. متدهایی که هر یک از این گروهها را تشکیل می‌دهند به ترتیب اعضای استاتیک کلاس‌های Enumerable و Queryable هستند. آنها به صورت *متدهای بسط* نوعی که رویش عمل می‌کنند تعریف می‌شوند. این حرف به معنی این است که آنها می‌توانند با استفاده از ترکیب نوشتاری متد استاتیک و یا ترکیب نوشتاری متد نمونه فراخوان شوند.

علاوه بر این، چندین متد عملگر پرس‌وجوی استاندارد روی انواعی غیر از آنهایی که مبتنی بر IEnumerable(T) یا IQueryable(T) هستند عمل می‌کنند. نوع Enumerable دو تا از چنین متدهایی را پیاده‌سازی می‌کند که هردو روی اشیائی از نوع IEnumerable عمل می‌کنند. این متدها، یعنی Cast(TResult)(IEnumerable) و OfType(TResult)(IEnumerable)، به شما اجازه می‌دهد تا یک مجموعه‌ی غیرپارامتری شده یا غیرجنریک را قادر سازید تا در قالب اصلی LINQ مورد پرس‌وجو واقع شوند. آنها این کار را با ایجاد مجموعه‌ای از اشیاء که به شدت نوعدار شده است انجام می‌دهند. کلاس Queryable دو متد مشابه را تعریف می‌کند، یعنی Cast(TResult)(IQueryable) و OfType(TResult)(IQueryable)، که روی اشیائی از نوع Queryable عمل می‌کنند.

عملگرهای پرس‌وجوی استاندارد بسته به این که آیا آنها یک مقدار واحد را برمی‌گردانند یا دنباله‌ای از مقادیر را در زمان‌بندی اجرایشان تفاوت دارند. آن متدهایی که یک مقدار واحد را برمی‌گردانند (برای مثال، **Average** و **Sum**) بلافاصله اجرا می‌شوند. متدهایی که دنباله‌ای از مقادیر را برمی‌گردانند اجرای پرس‌وجو را به تعویق می‌اندازند و یک شیء قابل شمارش را برمی‌گردانند.

در مورد متدهایی که روی مجموعه‌های درون حافظه‌ای عمل می‌کنند، یعنی آن متدهایی که واسط جنریک IEnumerable(T) را بسط می‌دهند، شیء قابل شمارش برگشتی آرگومان‌هایی را که به متد ارسال شده بودند

دریافت می‌کند. هرگاه آن شیء سرشماری شود، منطق عملگر پرس‌وجو به استخدام درآمده و نتایج پرس‌وجو برگشت داده می‌شوند.

درمقابل، متدهایی که واسط IQueryable(T) را بسط می‌دهند هیچ گونه رفتار پرس‌وجویی را پیاده‌سازی نمی‌کنند، اما یک درخت عبارت درست می‌کنند که بیانگر پرس‌وجویی است که انجام خواهد شد. پردازش پرس‌وجو توسط شیء IQueryable(T) منبع اداره می‌شود.

فراخوانی‌ها به متدهای پرس‌وجو می‌توانند در یک پرس‌وجو با یک‌دیگر تشکیل زنجیره دهند که این کار پرس‌وجوها را قادر می‌سازد تا به طور دلبخواهی پیچیده شوند.

کد نمونه زیر نشان می‌دهد که چگونه عملگرهای پرس‌وجو می‌توانند برای کسب اطلاعات درباره‌ی یک دنباله به کار برده شوند.

کد نمونه: **C#**

```
string sentence = "the quick brown fox jumps over the lazy dog";
// Split the string into individual words to create a collection.
string[] words = sentence.Split(' ');

// Using query expression syntax.
var query = from word in words
            group word.ToUpper() by word.Length into gr
            orderby gr.Key
            select new { Length = gr.Key, Words = gr };

// Using method-based query syntax.
var query2 = words.
    GroupBy(w => w.Length, w => w.ToUpper()).
    Select(g => new { Length = g.Key, Words = g }).
    OrderBy(o => o.Length);

foreach (var obj in query)
{
    Console.WriteLine("Words of length {0}:", obj.Length);
    foreach (string word in obj.Words)
        Console.WriteLine(word);
}

// This code example produces the following output:
//
// Words of length 3:
// THE
// FOX
// THE
// DOG
```

```
// Words of length 4:
// OVER
// LAZY
// Words of length 5:
// QUICK
// BROWN
// JUMPS
```

ترکیب نوشتاری عبارت پرس‌وجو

برخی از عملگرهای پرس‌وجوی استانداردی که دارای کاربرد وسیعی در LINQ هستند دارای واژه‌های کلیدی اختصاصی در زبان C# هستند که آنها را قادر می‌سازد تا به عنوان بخشی از یک عبارت پرس‌وجو فراخوان شوند.

بسط دادن عملگرهای پرس‌وجوی استاندارد

شما می‌توانید با ایجاد متدهای مختص دامینی که درخور دامین یا تکنولوژی هدف‌تان هستند مجموعه‌ای از عملگرهای پرس‌وجوی استاندارد را ترقی دهید. شما می‌توانید عملگرهای پرس‌وجوی استاندارد را با پیاده‌سازی‌های متعلق به خودتان که سرویس‌های اضافه‌ای چون ارزیابی ریموت، ترجمه‌ی پرس‌وجو و بهینه‌سازی را ارائه می‌دهند جایگزین کنید.

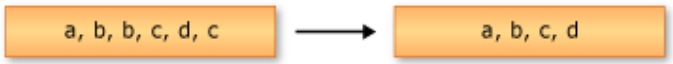
عملیات‌های انجام‌پذیر بر روی مجموعه

عملیات‌های مجموعه به عملیات‌های پرس‌وجویی اشاره می‌کند که مجموعه‌ی نتیجه‌ای را تولید می‌کنند که مبتنی بر حضور یا عدم حضور عناصر معادل در میان مجموعه‌های یکسان یا مجزا است.

مقایسه‌ی عملیات‌های مجموعه

Distinct

طرحواره‌ی زیر رفتار متد Enumerable.Distinct را بر روی دنباله‌ای از کاراکترها نشان می‌دهد. دنباله‌ی برگشتی شامل عناصر منحصر به فردی از دنباله‌ی ورودی است.

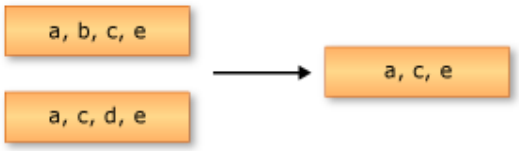


Except

طرحواره‌ی زیر رفتار Enumerable.Except را نشان می‌دهد. دنباله‌ی برگشتی تنها شامل عناصری از دنباله‌ی ورودی نخست است که در دنباله‌ی دوم حضور ندارند.

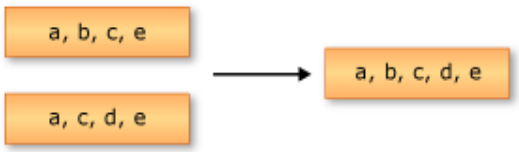
Intersect

طرحواره‌ی زیر رفتار Enumerable.Intersect را نشان می‌دهد. دنباله‌ی برگشتی شامل عناصری است که برای هردو دنباله‌ی ورودی مشترکند.



Union

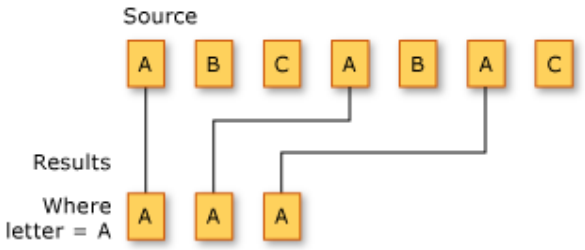
طرحواره‌ی زیر عمل اجتماع‌گیری را که بر روی دو دنباله‌ی کاراکتری انجام شده است نشان می‌دهد. دنباله‌ی برگشتی شامل عناصر منحصربه فردی از هر دو دنباله‌ی ورودی است.



فیلتر کردن داده‌ها

فیلترینگ به عملیاتی اشاره دارد که مجموعه نتیجه را محدود به داشتن تنها آن عناصری می‌کند که یک شرط بخصوص را ارضا کنند.

طرحواره‌ی زیر نتیجه‌ی فیلتر کردن دنباله‌ای از کاراکترها را نشان می‌دهد. گزاره‌ی مربوط به عملیات فیلترینگ تصریح می‌کند که کاراکتر باید 'A' باشد.



متدهای عملگر پرس‌وجوی استانداردی که عمل انتخاب را انجام می‌دهند در جدول زیر لیست شده‌اند.

نام متد	توضیح	ترکیب نوشتاری عبارت پرس‌وجوی C#
OfType	مقادیر را بسته توانایی‌شان برای قالب‌بندی شدن به یک نوع بخصوص انتخاب می‌کند.	قابل اعمال نیست.
Where	مقادیری را که مبتنی بر عمل یک گزاره هستند انتخاب می‌کند.	`where`

نمونه‌ای از ترکیب نوشتاری عبارت پرس‌وجو

مثال زیر از ضابطه‌ی **where** استفاده می‌کند تا از یک آرایه آن رشته‌هایی را که دارای یک طول بخصوص هستند فیلتر کند.

کد نمونه: **C#**

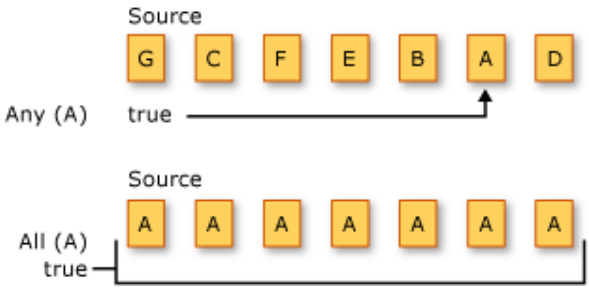
```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query = from word in words
                            where word.Length == 3
                            select word;

foreach (string str in query)
    Console.WriteLine(str);

/* This code produces the following output:

    the
    fox
```

```
*/
```

عملیات‌های سور

عملیات‌های سور یک مقدار Boolean برمی‌گردانند که نشانگر این هستند که برخی یا همه‌ی عناصر واقع در یک دنباله یک شرط را ارضا می‌کنند.

طرحواره‌ی زیر دو عملیات سور متفاوت را بر روی دو دنباله‌ی منبع متفاوت نشان می‌دهد. عملیات نخست دنبال یک یا چند عنصری که کاراکتر 'A' باشند می‌گردد و نتیجه **true** است. عملیات دوم دنبال این است که همه‌ی عناصر کاراکتر 'A' باشند و نتیجه **true** است.



متدهای زیر عملیات‌های سور را انجام می‌دهند:

نام متد	توضیح
All	تشخیص می‌دهد که آیا تمامی عناصر واقع در یک دنباله یک شرط را ارضا می‌کنند یا خیر.
Any	تشخیص می‌دهد که آیا برخی از عناصر واقع در یک دنباله یک شرط را ارضا می‌کنند یا نه.
Contains	تشخیص می‌دهد که آیا یک دنباله شامل یک عنصر بخصوص است یا نه.

مثال زیر متد جنریک Enumerable.All(TSource) را به کار می‌برد تا مشخص کند که آیا اسامی تمامی پت‌ها با حرف "B" شروع می‌شود یا نه. نتیجه‌ی عملیات **false** است زیرا یکی از پت‌ها اسمش `Whiskers` است.

کد نمونه: **C#**

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
public static void AllEx()
{
    // Create an array of Pets.
    Pet[] pets = { new Pet { Name="Barley", Age=10 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=6 } };

    // Determine whether all pet names
    // in the array start with 'B'.
    bool allStartWithB = pets.All(pet =>
                                      pet.Name.StartsWith("B"));

    Console.WriteLine(
        "{0} pet names start with 'B'.",
        allStartWithB ? "All" : "Not all");
}

// This code produces the following output:
//
//  Not all pet names start with 'B'.
```

عملیات‌های پروجکشن (انتخاب)

پروجکشن (انتخاب) به عمل انتقال یک شیء به فرم جدیدی که اغلب تنها از آن خاصیت‌هایی که متعاقباً به کار برده خواهند شد تشکیل می‌شود اشاره دارد. با استفاده از عمل پروجکشن (یا انتخاب)، شما می‌توانید نوع جدیدی را ایجاد کنید که از هر شیئی ساخته می‌شود. شما می‌توانید یک خاصیت را انتخاب کرده و عمل محاسباتی را بر روی آن انجام دهید. در ضمن می‌توانید شیء اصلی را بدون تغییر دادن آن انتخاب کنید.

Select

مثال زیر از ضابطه‌ی **select** استفاده می‌کند تا حرف نخست هر کدام از رشته‌های واقع در یک لیست را انتخاب کند.

کد نمونه: **C#**

```
List<string> words = new List<string>() { "an", "apple", "a", "day" };

var query = from word in words
            select word.Substring(0, 1);

foreach (string s in query)
    Console.WriteLine(s);

/* This code produces the following output:

    a
    a
    a
    d
```

```
*/
```

SelectMany

مثال زیر از ضابطه‌های **from** متعدد استفاده می کند تا هر یک از کلمات واقع در تک تک رشته‌های واقع در لیستی از رشته‌ها را انتخاب کند.

کد نمونه: **C#**

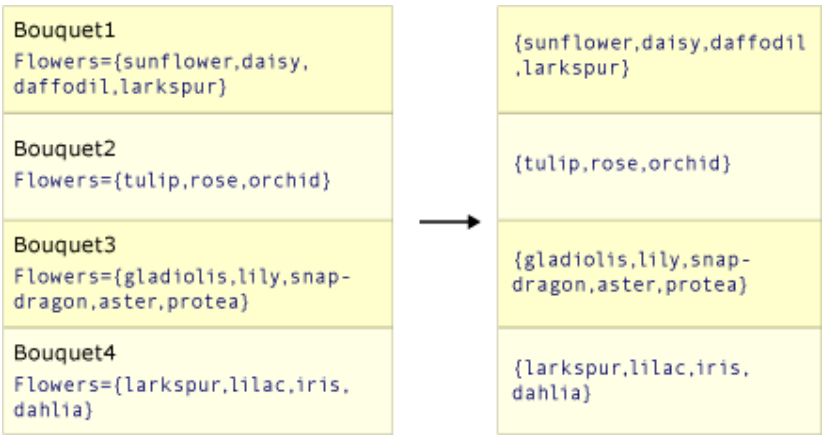
```
List<string> phrases = new List<string>() { "an apple a day", "the quick brown
fox" };

var query = from phrase in phrases
            from word in phrase.Split(' ')
            select word;

foreach (string s in query)
    Console.WriteLine(s);

/* This code produces the following output:

    an
    apple
    a
    day
    the
    quick
    brown
    fox
*/
```

متد Select در مقایسه با متد SelectMany

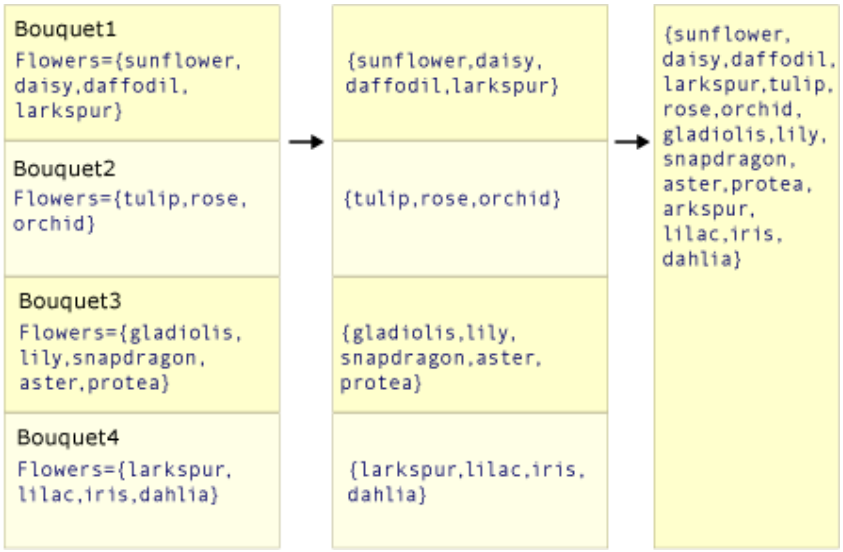
کار هر دو متد **Select()** و **SelectMany()** تولید مقدار (یا مقادیر) حاصله از مقادیر منبع است. **Select()** یک مقدار حاصله را برای هر مقدار منبعی تولید می‌کند. ازاینرو نتیجه کلی مجموعه‌ای است که دارای تعداد عناصر مشابهی با مجموعه‌ی منبع است. در مقابل، **SelectMany()** یک نتیجه‌ی کلی منحصر به فرد را تولید می‌کند که شامل زیرمجموعه‌های به هم چسبیده از هر یک از مقادیر منبع است. عمل انتقالی که به عنوان یک آرگومان به **SelectMany()** ارسال می‌شود باید دنباله‌ی قابل شمارشی از مقادیر را برای هر یک از مقادیر منبع برگرداند. پس از آن این دنباله‌های قابل شمارش توسط **SelectMany()** به یک دیگر الحاق می‌شوند تا دنباله‌ی بزرگ‌تری ایجاد شود.

دو طرحواره‌ی زیر تفاوت مفهومی مابین طرز عمل این متد را نشان می‌دهد. در هر حالت، فرض کنید تابع (انتقال) انتخابگر آرایه از گلها را از هر یک از مقادیر منبع انتخاب می‌کند.

این طرحواره نشان می‌دهد که چگونه متد **Select()** مجموعه‌ای را برمی‌گرداند که دارای تعداد عناصر مشابهی با مجموعه‌ی منبع است.



طرحواره‌ی زیر نشان می‌دهد که چگونه متد **SelectMany()** دنباله‌ی میانی از آرایه‌ها را به یک مقدار حاصل نهایی ملحق می‌کند که حاوی هر یک از مقادیر از هر کدام از آرایه‌های میانی است.



کد نمونه

مثال زیر رفتار متدهای **Select()** و **SelectMany()** را مقایسه می‌کند. این کد با گرفتن دو آیتم نخست هر لیست از اسامی گلها در مجموعه‌ی منبع یک دسته گل ایجاد می‌کند. در این مثال، مقدار یکتایی که تابع انتقال Select(TSource,TResult)(IEnumerable(TSource),Func(TSource,TResult)) استفاده می‌کند خودش مجموعه‌ای از مقادیر است. این امر نیاز به حلقه‌ی **foreach** دیگری دارد تا تک تک رشته‌ها را در هر یک از زیردنباله‌ها سرشماری کند.

کد نمونه: **C#**

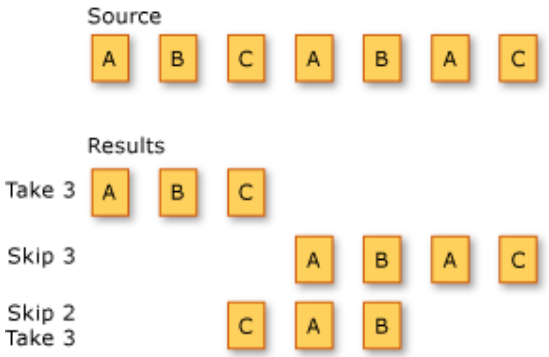
```
class Bouquet
{
    public List<string> Flowers { get; set; }
}

static void SelectVsSelectMany()
{
    List<Bouquet> bouquets = new List<Bouquet>() {
        new Bouquet { Flowers = new List<string> { "sunflower", "daisy",
"daffodil", "larkspur" }},
        new Bouquet{ Flowers = new List<string> { "tulip", "rose", "orchid" }},
        new Bouquet{ Flowers = new List<string> { "gladiolis", "lily",
"snapdragon", "aster", "protea" }},
        new Bouquet{ Flowers = new List<string> { "larkspur", "lilac", "iris",
"dahlia" }}
    };

    // *********** Select ***********
    IEnumerable<List<string>> query1 = bouquets.Select(bq => bq.Flowers);

    // ********* SelectMany *********
    IEnumerable<string> query2 = bouquets.SelectMany(bq => bq.Flowers);

    Console.WriteLine("Results by using Select():");
    // Note the extra foreach loop here.
    foreach (IEnumerable<String> collection in query1)
        foreach (string item in collection)
            Console.WriteLine(item);

    Console.WriteLine("\nResults by using SelectMany():");
    foreach (string item in query2)
        Console.WriteLine(item);

    /* This code produces the following output:

       Results by using Select():
        sunflower
        daisy
        daffodil
        larkspur
```

```
        tulip
        rose
        orchid
        gladiolis
        lily
        snapdragon
        aster
        protea
        larkspur
        lilac
        iris
        dahlia

      Results by using SelectMany():
        sunflower
        daisy
        daffodil
        larkspur
        tulip
        rose
        orchid
        gladiolis
        lily
        snapdragon
        aster
        protea
        larkspur
        lilac
        iris
        dahlia
    */

}
```

جزءبندی داده

جزءبندی در LINQ به عمل تقسیم دنباله‌ی ورودی به دو بخش بدون این که عناصر از نو آرایش شوند و پس از آن برگرداندن یکی از بخش‌ها اشاره دارد.

طرحواره‌ی زیر نتایج سه عمل جزءبندی متفاوت را بر روی دنباله‌ای از کاراکترها نشان می‌دهد. عملیات نخست سه عنصر نخست واقع در دنباله را برمی‌گرداند. عملیات دوم سه عنصر نخست را کنار گذاشته و عناصر باقیمانده را برمی‌گرداند. عملیات سوم دو عنصر نخست در دنباله را کنار گذاشته و سه عنصر بعدی را برمی‌گرداند.



متدهای عملگر پرس‌وجوی استانداردی که دنباله‌ها را تقسیم می‌کنند در جدول زیر لیست شده‌اند.

اسم عملگر	توضیح
Skip	عناصر را تا یک موقعیت مشخص شده در یک دنباله کنار می‌گذارد.
SkipWhile	عناصر را بر مبنای عمل یک گزاره تا زمانی که یک عنصر شرط را ارضا نکند کنار می‌گذارد.
Take	عناصر را تا یک موقعیت مشخص شده در یک دنباله می‌گیرد.
TakeWhile	عناصر را بر مبنای عمل یک گزاره تا زمانی که یک عنصر شرط را ارضا نکند کنار دریافت می‌کند.

کد نمونه‌ی زیر چگونگی استفاده از متد **Skip(TSource)** را برای کنار گذاشتن تعداد مشخصی از عناصر واقع در یک آرایه‌ی مرتب شده و برگرداندن عناصر باقی‌مانده نشان می‌دهد.

کد نمونه:

C#

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

IEnumerable<int> lowerGrades =
    grades.OrderByDescending(g => g).Skip(3);

Console.WriteLine("All grades except the top three are:");
foreach (int grade in lowerGrades)
    Console.WriteLine(grade);

/*
 This code produces the following output:

 All grades except the top three are:
 82
 70
 59
 56
*/
```

کد نمونه‌ی زیر چگونگی استفاده از متد **SkipWhile(TSource)(IEnumerable(TSource),Func(TSource,Boolean))** را برای کنار گذاشتن عناصر یک آرایه مادامی که یک شرط برقرار باشد نشان می‌دهد.

کد نمونه: **C#**

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

IEnumerable<int> lowerGrades =
    grades
    .OrderByDescending(grade => grade)
    .SkipWhile(grade => grade >= 80);

Console.WriteLine("All grades below 80:");
foreach (int grade in lowerGrades)
    Console.WriteLine(grade);

/*
 This code produces the following output:

 All grades below 80:
 70
 59
 56
*/
```

کد نمونه‌ی زیر چگونگی استفاده از متد **Take(TSource)** را برای برگرداندن عناصری از ابتدای یک دنباله نشان می‌دهد.

کد نمونه: **C#**

```
int[] grades = { 59, 82, 70, 56, 92, 98, 85 };

IEnumerable<int> topThreeGrades =
    grades.OrderByDescending(grade => grade).Take(3);

Console.WriteLine("The top three grades are:");
foreach (int grade in topThreeGrades)
    Console.WriteLine(grade);

/*
 This code produces the following output:

 The top three grades are:
 98
 92
 85
*/
```

کد نمونه‌ی زیر چگونگی استفاده از متد **TakeWhile(TSource)(IEnumerable(TSource),Func(TSource,Boolean))** را برای برگرداندن عناصری از ابتدای یک دنباله مادامی که یک شرط برقرار باشد نشان می‌دهد.

کد نمونه: **C#**

```
string[] fruits = { "apple", "banana", "mango", "orange",
                    "passionfruit", "grape" };

IEnumerable<string> query =
    fruits.TakeWhile(fruit => String.Compare("orange", fruit, true) != 0);

foreach (string fruit in query)
    Console.WriteLine(fruit);

/*
 This code produces the following output:

 apple
 banana
 mango
*/
```

عملیات‌های اتصال

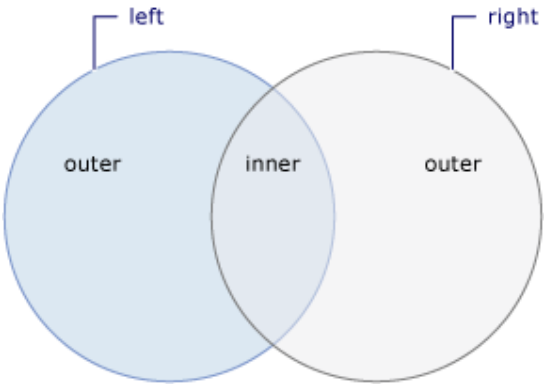
اتصال (یا پیوند) دو منبع داده رابطه‌ی اشیاء واقع در یک منبع داده با اشیائی است که یک مشخصه‌ی مشترک در منبع داده‌ی دیگر را به اشتراک می‌گذارند.

اتصال عملیات مهمی در پرس‌وجوهایی است که منابع داده‌ای را هدف قرار می‌دهند که روابط مابین آنها به طور مستقیم قابل پی‌گیری نیست. در برنامه‌نویسی شیءگرا، این حرف می‌تواند به معنای همبستگی و رابطه‌ای مابین اشیائی باشد که مدل‌سازی نشده است مانند جهت وارونه‌ی یک رابطه‌ی یک سویه. نمونه‌ای از یک رابطه‌ی یک سویه یک کلاس Customer است که دارای خاصیتی از نوع City است اما کلاس City دارای خاصیتی نیست که مجموعه‌ای از اشیاء Customer باشد. اگر دارای لیستی از اشیاء City هستید و قصد دارید تا همه‌ی مصرف کنندگان را در هر شهر پیدا کنید، می‌توانید از یک عمل اتصال (join) استفاده کنید تا آنها را پیدا کنید.

Join و GroupJoin متدهای اتصال ارائه شده در چارچوب LINQ هستند. این متدها اتصال‌های برابری را انجام می‌دهند یعنی اتصال‌هایی که دو منبع داده را بر اساس برابری کلیدهایشان مطابقت می‌دهند. (برای مقایسه، Transact-SQL از عملگرهای اتصال دیگری غیر از 'equals' پشتیبانی می‌کند، برای مثال عملگر 'less than'.) در

جملات پایگاه داده‌ی رابطه‌ای، متد Join یک اتصال درونی را پیاده‌سازی می‌کند، نوعی از اتصال که در آن تنها آن اشیائی که دارای جفت دیگری در مجموعه‌ی داده‌ی دیگر هستند برگردانده می‌شوند. متد GroupJoin هیچ گونه معادل مستقیمی در جملات پایگاه داده‌ی رابطه‌ای ندارد اما فرامجموعه‌ای از اتصال‌های درونی و بیرونی سمت چپ را پیاده‌سازی می‌کند. یک اتصالی خارجی سمت چپ اتصالی است که همه‌ی عناصر منبع داده‌ی نخست (سمت چپ) را برمی‌گرداند حتی اگر هیچ گونه عنصر همبسته‌ای در منبع داده‌ی دیگر نداشته باشد.

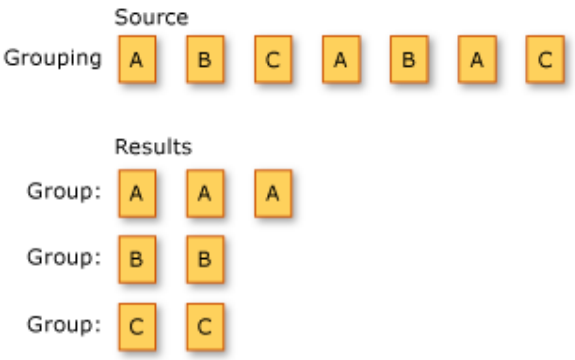
طرحواره‌ی زیر یک نمای مفهومی از دو مجموعه و عناصر میان این دو مجموعه که یا در یک اتصال درونی و یا در یک اتصال بیرونی چپ جای داده شده‌اند را نشان می‌دهد.



گروه‌بندی داده‌ها

گروه‌بندی به عمل قرار دادن داده در چند گروه طوری که عناصر واقع در هر گروه ویژگی مشترکی را به اشتراک بگذارند اشاره دارد.

طرحواره‌ی زیر نتیجه‌ی گروه‌بندی دنباله‌ای از کاراکترها را نشان می‌دهد. کلید مربوط به هر گروه، کاراکتر است.



GroupBy و ToLookup متدهای عملگر پرس‌وجوی استانداردی هستند که داده‌ها را گروه‌بندی می‌کنند. group...by یا group...by...into... معادل ترکیب نوشتاری عبارت پرس‌وجوی C# برای متد GroupBy است. C# برای متد ToLookup معادل ترکیب نوشتاری عبارت پرس‌وجو ندارد.

کد نمونه‌ی زیر از ضابطه‌ی **group by** استفاده می‌کند تا اعداد صحیح واقع در یک لیست را بر طبق زوج یا فرد بودنشان گروه‌بندی کند.

کد نمونه: **C#**

```
List<int> numbers = new List<int>() { 35, 44, 200, 84, 3987, 4, 199, 329, 446, 208
};

IEnumerable<IGrouping<int, int>> query = from number in numbers
                                         group number by number % 2;

foreach (var group in query)
{
    Console.WriteLine(group.Key == 0 ? "\nEven numbers:" : "\nOdd numbers:");
    foreach (int i in group)
        Console.WriteLine(i);
}

/* This code produces the following output:

    Odd numbers:
    35
    3987
    199
    329

    Even numbers:
    44
    200
    84
    4
    446
    208
*/
```

عملیات‌های تولید

تولید (یا زایش) به عمل ایجاد دنباله‌ی جدیدی از مقادیر اشاره دارد. متدهای عملگر پرس‌وجوی استانداردی که عمل تولید یا زایش را انجام می‌دهند در جدول زیر لیست شده‌اند. این متدها در C# دارای معادلی برای ترکیب نوشتاری عبارت پرس‌وجو نیستند.

متد	توضیح
DefaultIfEmpty	یک مجموعه‌ی تهی را با یک مجموعه‌ی یکتا که به طور پیش فرض مقداردار شده است جایگزین می‌کند.
Empty	یک مجموعه‌ی تهی را برمی‌گرداند.
Range	مجموعه‌ای را تولید می‌کند که حاوی دنباله‌ای از اعداد است.
Repeat	مجموعه‌ای را تولید می‌کند که حاوی یک مقدار تکراری است.

کد نمونه‌ی زیر چگونگی استفاده از متد **DefaultIfEmpty(TSource)(IEnumerable(TSource))** را برای آماده‌سازی یک مقدار پیش فرض در حالتی که دنباله‌ی منبع تهی باشد نشان می‌دهد. مثال زیر از یک دنباله‌ی ناتهی استفاده می‌کند.

کد نمونه: **C#**

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void DefaultIfEmptyEx1()
{
    List<Pet> pets =
        new List<Pet>{ new Pet { Name="Barley", Age=8 },
                       new Pet { Name="Boots", Age=4 },
                       new Pet { Name="Whiskers", Age=1 } };

    foreach (Pet pet in pets.DefaultIfEmpty())
        Console.WriteLine(pet.Name);
}

/*
 This code produces the following output:

 Barley
 Boots
 Whiskers
*/
```

ولی مثال زیر از یک دنباله‌ی تهی استفاده می‌کند.

کد نمونه: **C#**

```
List<int> numbers = new List<int>();

foreach (int number in numbers.DefaultIfEmpty())
    Console.WriteLine(number);

/*
 This code produces the following output:

 0
*/
```

کد نمونه‌ی زیر چگونگی استفاده از متد **Empty(TResult)()** را برای تولید یک IEnumerable(T) تهی نشان می‌دهد.

کد نمونه: **C#**

```
IEnumerable<decimal> empty = Enumerable.Empty<decimal>();
```

کد نمونه‌ی زیر یک کاربرد محتمل متد **Empty(TResult)()** را نشان می‌دهد. متد Aggregate به مجموعه‌ای از آرایه‌های رشته‌ای اعمال می‌شود. عناصر هر یک از آرایه‌های واقع در مجموعه تنها در صورتی به IEnumerable(T) حاصل اضافه می‌شوند که آرایه حاوی چهار یا تعداد بیشتری عنصر باشد. متد **Empty(TResult)** برای تولید مقدار آغازین (*seed value*) برای Aggregate به کار برده می‌شود زیرا اگر هیچ آرایه‌ای در مجموعه دارای چهار عنصر یا بیشتر از آن نباشد، تنها دنباله‌ی تهی برگردانده می‌شود.

کد نمونه: **C#**

```
string[] names1 = { "Hartono, Tommy" };
string[] names2 = { "Adams, Terry", "Andersen, Henriette Thaulow",
                    "Hedlund, Magnus", "Ito, Shu" };
string[] names3 = { "Solanki, Ajay", "Hoeing, Helge",
                    "Andersen, Henriette Thaulow",
                    "Potra, Cristina", "Iallo, Lucio" };

List<string[]> namesList =
    new List<string[]> { names1, names2, names3 };

// Only include arrays that have four or more elements
IEnumerable<string> allNames =
    namesList.Aggregate(Enumerable.Empty<string>(),
    (current, next) => next.Length > 3 ? current.Union(next) : current);

foreach (string name in allNames)
    Console.WriteLine(name);

/*
 This code produces the following output:

 Adams, Terry
```

```
 Andersen, Henriette Thaulow
 Hedlund, Magnus
 Ito, Shu
 Solanki, Ajay
 Hoeing, Helge
 Potra, Cristina
 Iallo, Lucio
*/
```

کد نمونه‌ی زیر چگونگی استفاده از متد **Range** را برای تولید دنباله‌ای از مقادیر نشان می‌دهد.

کد نمونه: **C#**

```
// Generate a sequence of integers from 1 to 10
// and then select their squares.
IEnumerable<int> squares = Enumerable.Range(1, 10).Select(x => x * x);

foreach (int num in squares)
    Console.WriteLine(num);

/*
 This code produces the following output:

 1
 4
 9
 16
 25
 36
 49
 64
 81
 100
*/
```

کد نمونه‌ی زیر چگونگی استفاده از متد **Repeat(TResult)** را برای تولید دنباله‌ای از یک مقدار تکراری نشان می‌دهد.

کد نمونه: **C#**

```
IEnumerable<string> strings =
    Enumerable.Repeat("I like programming.", 15);

foreach (String str in strings)
    Console.WriteLine(str);

/*
 This code produces the following output:

 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
```

```
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
 I like programming.
*/
```

عملیات‌های برابری

دو دنباله‌ای که عناصر متناظرشان برابر بوده و تعداد عناصر یکسانی را دارند به عنوان دنباله‌های برابر درنظرگرفته می‌شوند. متد SequenceEqual متد عملگر پرس‌وجوی استاندارد LINQ بوده و این متد در C# دارای معادل ترکیب نوشتاری عبارت پرس‌وجو نیست. متد SequenceEqual با مقایسه‌ی عناصر در یک وضعیت جفت‌گونه تشخیص می‌دهد که آیا دو دنباله برابر هستند یا نه.

کد نمونه‌ی زیر چگونگی استفاده از متد **SequenceEqual(TSource)(IEnumerable(TSource),IEnumerable(TSource))** را برای تشخیص برابری یا عدم برابری دو دنباله نشان می‌دهد. در این مثال دنباله‌ها برابرند.

کد نمونه: **C#**

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void SequenceEqualEx1()
{
    Pet pet1 = new Pet { Name = "Turbo", Age = 2 };
    Pet pet2 = new Pet { Name = "Peanut", Age = 8 };

    // Create two lists of pets.
    List<Pet> pets1 = new List<Pet> { pet1, pet2 };
    List<Pet> pets2 = new List<Pet> { pet1, pet2 };

    bool equal = pets1.SequenceEqual(pets2);

    Console.WriteLine(
        "The lists {0} equal.",
        equal ? "are" : "are not");
}

/*
 This code produces the following output:

 The lists are equal.
*/
```

کد نمونه‌ی دو دنباله‌ای را که برابر نیستند مقایسه می‌کند.

کد نمونه: **C#**

```
class Pet
{
    public string Name { get; set; }
    public int Age { get; set; }
}

public static void SequenceEqualEx2()
{
    Pet pet1 = new Pet() { Name = "Turbo", Age = 2 };
    Pet pet2 = new Pet() { Name = "Peanut", Age = 8 };

    // Create two lists of pets.
    List<Pet> pets1 = new List<Pet> { pet1, pet2 };
    List<Pet> pets2 =
        new List<Pet> { new Pet { Name = "Turbo", Age = 2 },
                        new Pet { Name = "Peanut", Age = 8 } };

    bool equal = pets1.SequenceEqual(pets2);

    Console.WriteLine("The lists {0} equal.", equal ? "are" : "are not");
}

/*
 This code produces the following output:

 The lists are not equal.
*/
```

عملیات‌های عنصر

عملیات‌های عنصر یک عنصر خاص و منحصر به فرد را از یک دنباله برمی‌گردانند. متدهای عملگر پرس‌وجوی استانداردی که عملیات‌های عنصر را انجام می‌دهند در جدول زیر لیست شده‌اند.

متد	توضیح
ElementAt	عنصر واقع در یک اندیس مشخص در یک مجموعه را برمی‌گرداند.
ElementAtOrDefault	عنصر واقع در یک اندیس مشخص در یک مجموعه را برمی‌گرداند و یا اگر اندیس خارج از دامنه باشد یک مقدار پیش فرض را برمی‌گرداند.
First	عنصر نخست یک مجموعه را برمی‌گرداند و یا اولین عنصری را که یک شرط را ارضا کند برمی‌گرداند.

FirstOrDefault	عنصر نخست یک مجموعه را برمی‌گرداند و یا اولین عنصری را که یک شرط را ارضا کند برمی‌گرداند. اگر چنین عنصری یافت نشود یک مقدار پیش فرض را برمی‌گرداند.
Last	آخرین عنصر یک مجموعه و یا آخرین عنصری را که یک شرط را ارضا کند برمی‌گرداند.
LastOrDefault	آخرین عنصر یک مجموعه و یا آخرین عنصری را که یک شرط را ارضا کند برمی‌گرداند. اگر چنین مقداری یافت نشود یک مقدار پیش فرض برگردانده می‌شود.
Single	تنها عنصر یک مجموعه و یا تنها عنصری را که یک شرط را ارضا کند برمی‌گرداند.
SingleOrDefault	تنها عنصر یک مجموعه و یا تنها عنصری را که یک شرط را ارضا کند برمی‌گرداند. درصورتی که چنین وجود نداشته باشد یا مجموعه دقیقاً شامل یک عنصر نباشد، یک مقدار پیش فرض برگردانده می‌شود.

تبدیل انواع داده در LINQ

متدهای تبدیل نوع اشیاء ورودی را تغییر می‌دهند. عملیات‌های تبدیل در پرس‌وجوهای LINQ در گستره‌ی وسیعی از برنامه‌ها سودمند است. اینها چند نمونه از این کاربردها هستند:

- متد Enumerable.AsEnumerable(TSource) می‌تواند برای پنهان کردن پیاده‌سازی سفارشی یک نوع از یک عملگر پرس‌وجوی استاندارد به کار برده شود.
- متد Enumerable.OfType(TResult) می‌تواند برای امکان‌پذیر ساختن مجموعه‌های غیرپارامتری شده برای پرس‌وجوی LINQ به کار برده شود.
- متدهای Enumerable.ToArray(TSource)، Enumerable.ToDictionary، Enumerable.ToList(TSource) و Enumerable.ToLookup می‌توانند برای تحمیل اجرای فوری پرس‌وجو به کار برده شوند به جای این که اجرای پرس‌وجو تا زمان حرکت بر روی نتایج پرس‌وجو در یک حلقه‌ی foreach به تعویق انداخته شود.

جدول زیر متدهای عملگر پرس‌وجوی استانداردی را که تبدیلات نوع داده را انجام می‌دهند لیست کرده است. متدهای تبدیلی که نامشان با "As" شروع می‌شود نوع استاتیک مجموعه‌ی منبع را تغییر می‌دهند اما آن را

سرشماری نمی‌کنند. متدهایی که نامشان با "To" شروع می‌شود مجموعه‌ی منبع را شمارش کرده و آیتم‌های آن را در نوع مجموعه‌ی متناظر جای می‌دهند.

متد	توضیح
AsEnumerable	ورودی را در حالی که به صورت IEnumerable(T) نوعدار شده است برمی‌گرداند.
AsQueryable	یک واسط (جنریک) IEnumerable را به یک واسط (جنریک) IQueryable تبدیل می‌کند.
Cast	عناصر یک مجموعه را به یک نوع مشخص قالب‌بندی می‌کند.
OfType	مقادیر را بسته به توانایی آنها برای قالب‌بندی شدن به یک نوع مشخص فیلتر می‌کند.
ToArray	یک مجموعه را به یک آرایه تبدیل می‌کند. این متد اجرای فوری پرس‌وجو را تحمیل می‌کند.
ToDictionary	بسته به عمل یک کلید انتخاب کننده، عناصر را در یک Dictionary(TKey,TValue) جای می‌دهد. این متد اجرای فوری پرس‌وجو را تحمیل می‌کند.
ToList	یک مجموعه را به یک List(T) تبدیل می‌کند. این متد اجرای فوری پرس‌وجو را تحمیل می‌کند.
ToLookup	بر مبنای عمل یک کلید انتخاب کننده، عناصر را در یک Lookup(TKey,TElement) (یک دیکشنری یک به چند) جای می‌دهد. این متد اجرای فوری پرس‌وجو را تحمیل می‌کند.

کد زیر یک متغیر دامنه را که صریحاً نوعدار شده است به کار می‌برد تا قبل از دسترسی به یک عضوی که تنها روی زیرمجموعه قابل دسترس است یک نوع را به یک زیرنوع قالب‌بندی کند.

کد نمونه: **C#**

```
class Plant
{
    public string Name { get; set; }
}

class CarnivorousPlant : Plant
{
    public string TrapType { get; set; }
}
```

```
static void Cast()
{
    Plant[] plants = new Plant[] {
        new CarnivorousPlant { Name = "Venus Fly Trap", TrapType = "Snap Trap" },
        new CarnivorousPlant { Name = "Pitcher Plant", TrapType = "Pitfall Trap"
},
        new CarnivorousPlant { Name = "Sundew", TrapType = "Flypaper Trap" },
        new CarnivorousPlant { Name = "Waterwheel Plant", TrapType = "Snap Trap" }
    };

    var query = from CarnivorousPlant cPlant in plants
                where cPlant.TrapType == "Snap Trap"
                select cPlant;

    foreach (Plant plant in query)
        Console.WriteLine(plant.Name);

    /* This code produces the following output:

        Venus Fly Trap
        Waterwheel Plant
    */
}
```

عملیات الحاق (یا زنجیربندی)

الحاق (یا زنجیربندی) به عمل چسباندن یک دنباله به یک دنباله‌ی دیگر اشاره دارد. طرحواره‌ی زیر یک عمل الحاق انجام شده بر روی دو دنباله‌ی کاراکتری را نشان می‌دهد.

متد Concat متد عملگر پرس‌وجوی استانداردی است که عمل الحاق را انجام می‌دهد. این متد دو دنباله را به هم می‌چسباند تا دنباله‌ی جدیدی را شکل دهد.

کد نمونه‌ی زیر چگونگی استفاده از متد **Concat(TSource)(IEnumerable(TSource),IEnumerable(TSource))** را برای الحاق دو دنباله نشان می‌دهد.

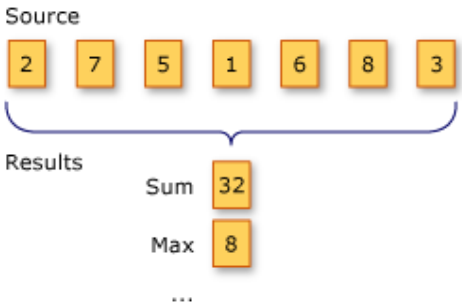
کد نمونه: **C#**

```
class Pet
{
    public string Name { get; set; }
```

```
    public int Age { get; set; }
}

static Pet[] GetCats()
{
    Pet[] cats = { new Pet { Name="Barley", Age=8 },
                   new Pet { Name="Boots", Age=4 },
                   new Pet { Name="Whiskers", Age=1 } };
    return cats;
}

static Pet[] GetDogs()
{
    Pet[] dogs = { new Pet { Name="Bounder", Age=3 },
                   new Pet { Name="Snoopy", Age=14 },
                   new Pet { Name="Fido", Age=9 } };
    return dogs;
}

public static void ConcatEx1()
{
    Pet[] cats = GetCats();
    Pet[] dogs = GetDogs();

    IEnumerable<string> query =
        cats.Select(cat => cat.Name).Concat(dogs.Select(dog => dog.Name));

    foreach (string name in query)
        Console.WriteLine(name);
}

// This code produces the following output:
//
// Barley
// Boots
// Whiskers
// Bounder
// Snoopy
// Fido
```

عملیات برافزودگی

یک عمل برافزودگی مقدار واحدی را از مجموعه‌ای از مقادیر محاسبه می‌کند. نمونه‌ای از یک عمل برافزودگی محاسبه‌ی میانگین دمای روزانه است. طرحواره‌ی زیر نتیجه‌ی دو عمل برافزودگی مختلف را بر روی دنباله‌ای از اعداد نشان می‌دهد. عملیات نخست اعداد را جمع می‌زند. عملیات دوم مقدار بیشینه‌ی دنباله را برمی‌گرداند.



متدهای عملگر پرس‌وجوی استانداردی که عملیات‌های برافزودگی را انجام می‌دهند در جدول زیر لیست شده‌اند.

متد	توضیح
Aggregate	یک عمل برافزودگی سفارشی را بر روی مقادیر یک مجموعه انجام می‌دهد.
Average	مقدار میانگین مجموعه‌ای از مقادیر را محاسبه می‌کند.
Count	تعداد عناصر واقع در یک مجموعه، به طور دلخواه تنها آن عناصری را که شرایط یک گزاره را ارضا می‌کنند برمی‌گرداند.
LongCount	تعداد عناصر واقع در یک مجموعه‌ی بزرگ، به طور دلخواه تنها آن عناصری را که شرایط یک گزاره را ارضا می‌کنند برمی‌گرداند.
Max	مقدار بیشینه را در بین مقادیر یک مجموعه تعیین می‌کند.
Min	مقدار کمینه را در بین مقادیر یک مجموعه تعیین می‌کند.
Sum	مجموع مقادیر واقع در یک مجموعه را محاسبه می‌کند.

نقل و انتقال داده با LINQ

کار LINQ تنها بازیابی داده نیست. LINQ ابزار قدرتمندی برای نقل و انتقال داده‌هاست. با استفاده از یک پرس‌وجوی LINQ می‌توانید یک دنباله‌ی منبع را به عنوان ورودی به کار برده و آن را به روش‌های متعددی ویرایش کنید تا یک دنباله‌ی خروجی جدید را ایجاد کنید. شما می‌توانید با مرتب‌سازی و گروه‌بندی، خود دنباله

را بدون ویرایش کردن خود عناصر تغییر دهید. اما شاید بتوان گفت قدرتمندترین ویژگی پرس‌وجوهای LINQ قابلیت ایجاد انواع جدید است. این کار در ضابطه‌ی **select** انجام می‌شود. برای مثال، می‌توانید فعالیت‌های زیر را انجام دهید:

- ادغام دنباله‌های ورودی متعدد در یک دنباله‌ی خروجی واحدی که دارای یک نوع جدید است.
- ایجاد دنباله‌های خروجی که عناصرشان تنها از یک یا چندین خاصیت هر یک از عناصر واقع در دنباله‌ی منبع تشکیل می‌شوند.
- ایجاد دنباله‌های خروجی که عناصرشان از نتایج عملیات‌های انجام شده بر روی داده‌ی منبع تشکیل می‌شوند.
- ایجاد دنباله‌های خروجی در یک قالب متفاوت. برای مثال، شما می‌توانید داده را از سطور (یا رکوردهای) SQL یا فایل‌های متن به XML تبدیل (یا منتقل) کنید.

این فقط چند نمونه ار فعالیت‌هایی هستند که می‌توانید انجام دهید. البته، این تبدیلات می‌توانند به روش‌های مختلف در پرس‌وجوی یکسانی ترکیب شوند. علاوه بر این، دنباله‌ی خروجی یک پرس‌وجو می‌تواند به عنوان دنباله‌ی ورودی برای یک پرس‌وجوی جدید مورد استفاده قرار گیرد.

متصل کردن ورودی‌ها در یک دنباله‌ی خروجی واحد

شما می‌توانید یک پرس‌وجوی LINQ را برای ایجاد یک دنباله‌ی خروجی که شامل عناصری از یک یا چند دنباله‌ی ورودی است مورد استفاده قرار دهید. مثال زیر چگونگی ترکیب دو ساختار داده‌ی درون حافظه‌ای را نشان می‌دهد، اما مفاهیم یکسانی به ترکیب داده از منابع XML یا SQL یا DataSet اعمال می‌شوند.

کد نمونه: **C#**

```
class Student
{
    public string First { get; set; }
    public string Last {get; set;}
    public int ID { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
```

```
    public List<int> Scores;
}

class Teacher
{
    public string First { get; set; }
    public string Last { get; set; }
    public int ID { get; set; }
    public string City { get; set; }
}
```

مثال زیر پرس‌وجو را نشان می‌دهد:

C# کد نمونه:

```
class DataTransformations
{
    static void Main()
    {
        // Create the first data source.
        List<Student> students = new List<Student>()
        {
            new Student {First="Svetlana",
                Last="Omelchenko",
                ID=111,
                Street="123 Main Street",
                City="Seattle",
                Scores= new List<int> {97, 92, 81, 60}},
            new Student {First="Claire",
                Last="O'Donnell",
                ID=112,
                Street="124 Main Street",
                City="Redmond",
                Scores= new List<int> {75, 84, 91, 39}},
            new Student {First="Sven",
                Last="Mortensen",
                ID=113,
                Street="125 Main Street",
                City="Lake City",
                Scores= new List<int> {88, 94, 65, 91}},
        };

        // Create the second data source.
        List<Teacher> teachers = new List<Teacher>()
        {
            new Teacher {First="Ann", Last="Beebe", ID=945, City =
"Seattle"},
            new Teacher {First="Alex", Last="Robinson", ID=956, City =
"Redmond"},
            new Teacher {First="Michiyo", Last="Sato", ID=972, City =
```

```
"Tacoma"}
        };

        // Create the query.
        var peopleInSeattle = (from student in students
                    where student.City == "Seattle"
                    select student.Last)
                    .Concat(from teacher in teachers
                            where teacher.City == "Seattle"
                            select teacher.Last);

        Console.WriteLine("The following students and teachers live in
Seattle:");
        // Execute the query.
        foreach (var person in peopleInSeattle)
        {
            Console.WriteLine(person);
        }

        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/* Output:
    The following students and teachers live in Seattle:
    Omelchenko
    Beebe
 */
```

انتخاب یک زیرمجموعه از هر یک از عناصر منبع

دو روش اصلی برای انتخاب یک زیرمجموعه از هر یک از عناصر واقع در دنباله‌ی منبع وجود دارد:

۱. برای انتخاب تنها یکی از اعضای عنصر منبع، از عملگر نقطه استفاده کنید. در مثال زیر، فرض کنید که یک شیء `Customer` حاوی چندین خاصیت عمومی (**public**) باشد از جمله یک رشته به نام `City`. این پرس‌وجو هرگاه اجرا شود یک دنباله‌ی خروجی از رشته تولید خواهد کرد.

کد نمونه:

```
var query = from cust in Customers
            select cust.City;
```

۲. برای ایجاد عناصری که حاوی بیشتر از یکی از خاصیت‌های عنصر منبع باشند می‌توانید یک مقداردهنده‌ی شیء را با یک شیء دارای نام یا یک نوع بدون نام و نامشخص به کار ببرید. مثال زیر کاربرد یک نوع نامشخص را برای کپسوله (نهان) کردن دو خاصیت هر یک از عناصر `Customer` نشان می‌دهد:

کد نمونه:

```
var query = from cust in Customer
            select new {Name = cust.Name, City = cust.City};
```

انتقال اشیاء درون حافظه‌ای به XML

پرس‌وجوهای LINQ انتقال داده مابین ساختارهای درون حافظه‌ای، پایگاه‌های داده‌ی SQL، DataSetهای ADO.NET و جریان‌ها یا اسناد XML را آسان می‌سازند. مثال زیر اشیاء واقع در یک ساختار داده‌ی درون حافظه‌ای را به عناصر XML منتقل می‌کند.

کد نمونه: **C#**

```
class XMLTransform
{
    static void Main()
    {
        // Create the data source by using a collection initializer.
        List<Student> students = new List<Student>()
        {
            new Student {First="Svetlana", Last="Omelchenko", ID=111,
Scores = new List<int>{97, 92, 81, 60}},
            new Student {First="Claire", Last="O'Donnell", ID=112, Scores
= new List<int>{75, 84, 91, 39}},
            new Student {First="Sven", Last="Mortensen", ID=113, Scores =
new List<int>{88, 94, 65, 91}},
        };

        // Create the query.
        var studentsToXML = new XElement("Root",
            from student in students
            let x = String.Format("{0},{1},{2},{3}", student.Scores[0],
                    student.Scores[1], student.Scores[2],
student.Scores[3])
            select new XElement("student",
                       new XElement("First", student.First),
                       new XElement("Last", student.Last),
                       new XElement("Scores", x)
                    ) // end "student"
                ); // end "Root"
```

```
        // Execute the query.
        Console.WriteLine(studentsToXML);

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

کد نمونه‌ی قبل، خروجی XML زیر را تولید می‌کند:

کد نمونه:

```
< Root>
  <student>
    <First>Svetlana</First>
    <Last>Omelchenko</Last>
    <Scores>97,92,81,60</Scores>
  </student>
  <student>
    <First>Claire</First>
    <Last>O'Donnell</Last>
    <Scores>75,84,91,39</Scores>
  </student>
  <student>
    <First>Sven</First>
    <Last>Mortensen</Last>
    <Scores>88,94,65,91</Scores>
  </student>
</Root>
```

انجام عملیات‌ها بر روی عناصر منبع

یک دنباله‌ی خروجی ممکن است شامل هیچ یک از عناصر یا خاصیت‌های عناصر دنباله‌ی منبع نباشد. در عوض خروجی می‌تواند دنباله‌ای از مقادیر باشد که با استفاده از عناصر منبع به عنوان آرگومان‌های ورودی محاسبه می‌شوند. پرس‌وجوی ساده‌ی زیر هرگاه اجرا شود دنباله‌ای از رشته‌ها را بیرون می‌دهد که مقادیرشان بیانگر یک محاسبه‌ی مبتنی بر دنباله‌ی منبعی از عناصری از نوع **double** است.

نکته:

در صورت انتقال پرس‌وجو به برخی از حوزه‌های (domain) دیگر فراخوانی متدها در عبارات پرس‌وجو

پشتیبانی نمی‌شود. برای مثال، شما نمی‌توانید یک متد #C معمول را در LINQ به SQL فراخوانی کنید زیرا SQL Server دارای هیچ زمینه‌ای برای آن نیست. اگرچه، شما می‌توانید رویه‌های ذخیره شده را به متدها نگاشت کرده و آنها را فراخوانی کنید.

کد نمونه: **C#**

```
class FormatQuery
{
    static void Main()
    {
        // Data source.
        double[] radii = { 1, 2, 3 };

        // Query.
        IEnumerable<string> query =
            from rad in radii
            select String.Format("Area = {0}", (rad * rad) * 3.14);

        // Query execution.
        foreach (string s in query)
            Console.WriteLine(s);

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/* Output:
    Area = 3.14
    Area = 12.56
    Area = 28.26
*/
```

روابط مابین نوعها در عملیات‌های پرس وجو

برای نوشتن پرس‌وجوها به شکل کارامد، لازم است تا بدانید که نوعهای متغیرها در یک عملیات پرس‌وجوی کامل چگونه باید یک دیگر ارتباط دارند. با آگاهی از این روابط از اتفاقات پشت صحنه‌ای که در هنگام نوعدار شدن ضمنی متغیرها با استفاده از **var** رخ می‌دهند آگاه خواهید شد.

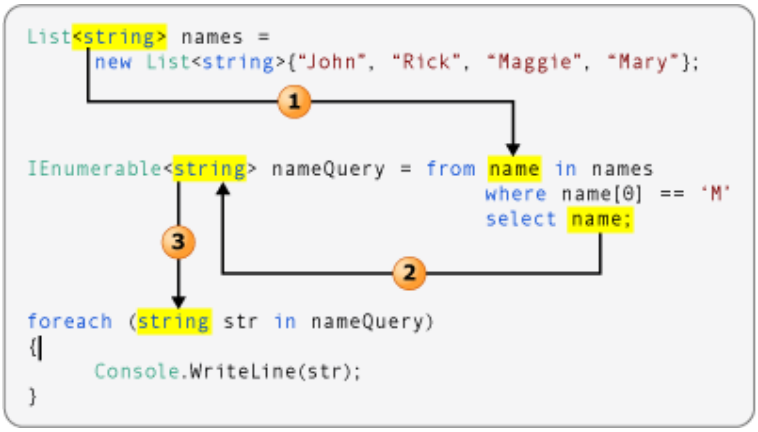
عملیات‌های پرس‌وجوی LINQ به شکل قدرتمندی در منبع داده، در خود پرس‌وجو و در زمان اجرای پرس‌وجو نوعدار می‌شوند (مدیریت نوع قوی). نوع متغیرها در پرس‌وجو باید با نوع عناصر واقع در منبع داده و

با نوع متغیر کنترلی در دستور **foreach** سازگار باشد. این مدیریت نوع قوی تضمین می‌کند که خطاهای نوع در زمان کامپایل شناسایی شده و قبل از این که کاربران با آنها برخورد کنند می‌توانند تصحیح شوند.

برای نشان دادن این روابط نوعی، اغلب مثال‌هایی که در پی می‌آیند از مدیریت نوع صریح برای همه‌ی متغیرها استفاده می‌کنند. مثال آخری نشان می‌دهد که چگونه مفاهیم یکسانی حتی هنگام استفاده از مدیریت نوع ضمنی با استفاده واژه کلیدی **var** قابل اعمال است.

پرس‌وجوهایی که داده‌ی منبع را منتقل نمی‌کنند

شکل زیر یک عملیات پرس‌وجوی LINQ به اشیاء را نشان می‌دهد که هیچ نقل و انتقالی روی داده انجام نمی‌دهد. منبع شامل دنباله‌ای از رشته‌هاست و خروجی پرس‌وجو نیز دنباله‌ای از رشته‌هاست.



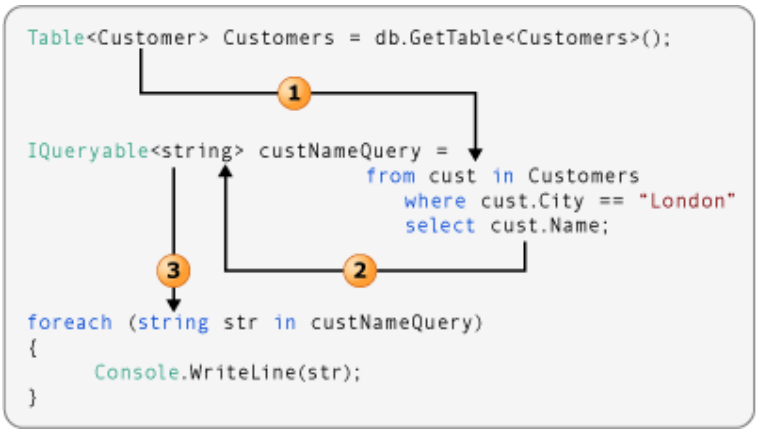
۳. آرگومان نوع منبع داده نوع متغیر دامنه را تعیین می‌کند.

۴. نوع شیئی که انتخاب می‌شود نوع متغیر پرس‌وجو را تعیین می‌کند. در اینجا `name` یک رشته است. ازاینرو متغیر پرس‌وجو یک **IEnumerable**<string> است.

۵. در یک دستور **foreach** روی متغیر پرس‌وجو حرکت می‌شود. از آن جایی که متغیر پرس‌وجو دنباله‌ای از رشته‌هاست، متغیر کنترلی حلقه نیز یک رشته است.

پرس‌وجوهایی که داده‌ی منبع را منتقل می‌کنند

شکل زیر یک عملیات پرس‌وجوی LINQ به SQL را نشان می‌دهد که یک انتقال (تبدیل) ساده را روی داده انجام می‌دهد. پرس‌وجو دنباله‌ای از اشیاء `Customer` را به عنوان ورودی گرفته و تنها خاصیت `Name` را در نتیجه گزینش می‌کند. از آن جایی که `Name` یک رشته است، پرس‌وجو دنباله‌ای از رشته‌ها را به عنوان خروجی تولید می‌کند.

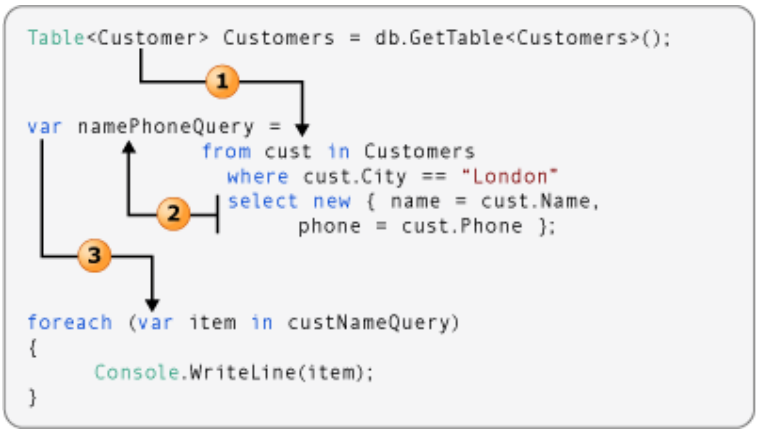


۱. آرگومان نوع منبع داده نوع متغیر دامنه را تعیین می‌کند.

۲. دستور **select** به جای شیء `Customer` کامل، خاصیت `Name` را برمی‌گرداند. از آن جایی که `Name` یک رشته است، نوع آرگومان `custNameQuery` رشته (**string**) است نه خود شیء `Customer`.

۳. از آن جایی که `custNameQuery` دنباله‌ای از رشته‌هاست، متغیر کنترلی حلقه‌ی **foreach** نیز باید یک **string** باشد.

شکل زیر انتقال داده‌ای را نشان می‌دهد که اندکی پیچیده‌تر است. دستور **select** نوع بدون نامی را برمی‌گرداند که تنها دو عضو از شیء `Customer` اصلی را گیر انداخته است.



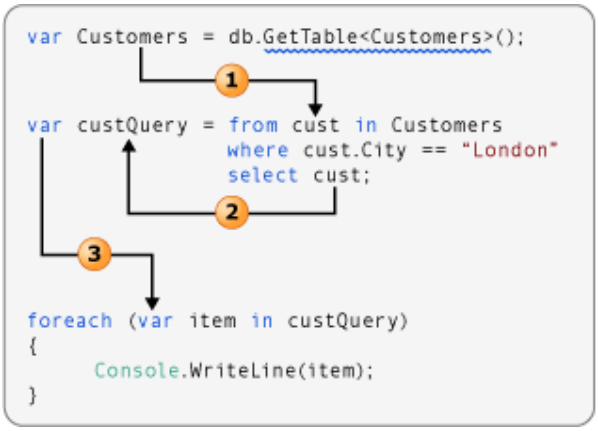
۱. آرگومان نوع منبع داده همواره نوع متغیر دامنه در پرس‌وجو است.

۲. از آن جایی که دستور **select** در اینجا یک نوع بدون نام تولید می‌کند، متغیر پرس وجو باید با استفاده از **var** به صورت ضمنی نوعدار شود.

۳. از آن جایی که نوع متغیر پرس‌وجو ضمنی است، متغیر کنترلی در حلقه‌ی **foreach** نیز باید ضمنی باشد.

اجازه دادن به کامپایلر برای استنتاج اطلاعات نوع

هرچند لازم است تا با روابط نوع در یک عملیات پرس‌وجو آشنا باشید، این حق انتخاب را دارید که کامپایلر اجازه دهید تا همه‌ی کارها را برای شما انجام دهد. واژه کلیدی **var** می‌تواند برای هرگونه متغیر محلی واقع در یک عملیات پرس‌وجو مورد استفاده قرار گیرد. شکل زیر دقیقاً معادل با مثال شماره دویی است که پیش از این بحث شد. تنها تفاوت این است که کامپایلر نوع قوی را برای هر یک از متغیرهای واقع در عملیات پرس‌وجو آماده خواهد کرد:



ترکیب نوشتاری پرس‌وجو در مقابل ترکیب نوشتاری متد

اغلب پرس‌وجوها در اسناد اولیه LINQ با استفاده از ترکیب نوشتاری پرس‌وجوی اعلانی معرفی شده در C# 3.0 به صورت عبارات پرس‌وجو نوشته می‌شوند. اگر، CLR دارای هیچ گونه نمادگذاری ترکیب نوشتاری پرس‌وجو برای خودش نیست. ازاینرو، در زمان کامپایل، عبارات پرس‌وجو به چیزهایی ترجمه می‌شوند که CLR آنها را می‌فهمد: فراخوان متدها. این متدها *عملگرهای پرس‌وجوی استاندارد* خوانده می‌شوند و دارای نام‌هایی چون **Where**، **Select**، **GroupBy**، **Join**، **Max**، **Average** و غیره هستند. شما می‌توانید به جای ترکیب نوشتاری پرس‌وجو، آنها را مستقیماً با استفاده از ترکیب نوشتاری متد فراخوانی کنید.

به طور کلی، توصیه می‌کنیم که از ترکیب نوشتاری پرس‌وجو استفاده کنید زیرا این ترکیب معمولاً ساده‌تر و خواناتر است؛ اگرچه، مابین ترکیب نوشتاری متد و ترکیب نوشتاری پرس‌وجو هیچ گونه تفاوت معناشناختی وجود ندارد. علاوه بر این، برخی از پرس‌وجوها مانند آنهایی که تعداد عناصری را که با یک شرایط مشخص مطابقت می‌کنند بازیابی می‌کنند یا آنهایی که عنصری را بازیابی می‌کنند که دارای مقدار بیشینه در یک دنباله‌ی منبع است، تنها می‌توانند به صورت فراخوان متدها بیان شوند. مستندات مرجع برای عملگرهای پرس‌وجوی استاندارد در فضای نام System.Linq به طور معمول از ترکیب نوشتاری متد استفاده می‌کنند. ازاینرو، حتی هنگام شروع به کار برای نوشتن پرس‌وجوهای LINQ، آشنایی با چگونگی استفاده از ترکیب نوشتاری متد در پرس‌وجوها و در خود عبارات پرس‌وجو سودمند خواهد بود.

متدهای بسط عملگر پرس‌وجوی استاندارد

مثال زیر یک عبارت پرس‌وجوی ساده و پرس‌وجوی معادل آن را که به صورت یک *پرس‌وجوی مبتنی بر متد* نوشته شده است نشان می‌دهد.

کد نمونه: **C#**

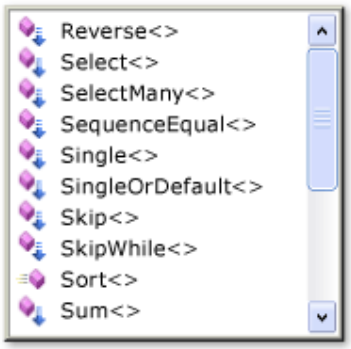
```
class QueryVMethodSyntax
{
    static void Main()
    {
        int[] numbers = { 5, 10, 8, 3, 6, 12};
```

```
        //Query syntax:
        IEnumerable<int> numQuery1 =
            from num in numbers
            where num % 2 == 0
            orderby num
            select num;

        //Method syntax:
        IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 == 0).OrderBy(n
=> n);

        foreach (int i in numQuery1)
        {
            Console.Write(i + " ");
        }
        Console.WriteLine(System.Environment.NewLine);
        foreach (int i in numQuery2)
        {
            Console.Write(i + " ");
        }

        // Keep the console open in debug mode.
        Console.WriteLine(System.Environment.NewLine);
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
/*
    Output:
    6 8 10 12
    6 8 10 12
 */
```

خروجی دو مثال یکسان است. می‌توانید ببینید که نوع متغیر پرس‌وجو در هر دو شکل یکسان است: IEnumerable(T).

برای درک پرس‌وجوی مبتنی متد بیایید آن را دقیق‌تر بررسی کنیم. در سمت راست عبارت متوجه خواهید شد که ضابطه‌ی `where` اکنون به صورت یک متد نمونه روی شیء `numbers` بیان می‌شود که همان طور که آن را بازفراخوانی خواهید کرد دارای نوع `IEnumerable<int>` است. اگر با واسط جنریک **IEnumerable**(T) آشنا باشید می‌دانید که این واسط دارای متد **Where** نیست. با این‌حال، اگر در IDE ویژوال استودیو لیست تکمیل IntelliSense را احضار کنید تنها یک متد **Where** را نخواهید دید بلکه بسیاری از متدهای دیگر نظیر **Select**، **SelectMany**، **Join** و **Orderby** را خواهید دید. اینها همگی عملگرهای پرس‌وجوی استاندارد هستند.

```
List<string> list = new List<string();
list.
```



هرچند به نظر می‌رسد که انگار واسط جنریک IEnumerable(T) بازتعریف شده است تا دربرگیرنده‌ی این متدهای افزوده گردد، در حقیقت این اصل قضیه نیست. عملگرهای پرس‌وجوی استاندارد به صورت نوع جدیدی از متد به نام متدهای بسط پیاده‌سازی می‌شوند. متدهای بسط یک نوع موجود را «بسط» می‌دهند؛ آنها می‌توانند فراخوانی شوند انگار که متدهای نمونه بر روی نوع بوده‌اند. عملگرهای پرس‌وجوی استاندارد واسط جنریک IEnumerable(T) را بسط می‌دهند و این امر دلیل این است که شما می‌توانید `numbers.Where(...)` را بنویسید.

برای آغاز به کار استفاده از LINQ، همه‌ی آن چیزی که واقعاً لازم است تا درباره‌ی متدهای بسط بدانید این است که چگونه آنها را با استفاده از احکام **using** به دامنه (یا میدان دید) برنامه‌تان برسانید. از نقطه نظر برنامه‌ی شما، متد بسط و متد نمونه‌ی عادی برابرند.

عبارات لامبدا

در مثال قبل دیدید که عبارت شرطی (`num % 2 == 0`) به صورت یک آرگومان برخط به متد **Where** ارسال شد: `Where(num => num % 2 == 0)`. این عبارت برخط یک عبارت لامبدا خوانده می‌شود. این یک روش سرراست و راحت برای نوشتن کدی است که در غیر این صورت باید به شکل نافرم‌تری به صورت یک متد بی‌نام یا یک نماینده‌ی جنریک یا یک درخت عبارت نوشته می‌شد. در C# عملگر لامبدا **=>** است که به صورت «رفتن به» خوانده می‌شود. `num` واقع در سمت چپ عملگر لامبدا متغیر ورودی است که متناظر با `num` در عبارت پرس‌وجوست. کامپایلر می‌تواند نوع `num` را استنتاج کند زیرا می‌داند که `numbers` یک نوع جنریک

IEnumerable(T) است. بدنه‌ی لامبدا عیناً همان عبارت واقع در ترکیب نوشتاری پرس‌وجو یا هرگونه عبارت یا دستور C# دیگر است؛ بدنه‌ی لامبدا می‌تواند حاوی فراخوانی متدها و دیگر دستورات منطقی پیچیده باشد. «مقدار برگشتی» عیناً نتیجه عبارت است.

هنگام استفاده از LINQ، لزومی ندارد که از عبارات لامبدا به شکل گسترده‌ای استفاده کنید. اگرچه، برخی از پرس‌وجوها تنها می‌توانند در ترکیب نوشتاری متد بیان شوند و برخی از آنها نیازمند عبارات لامبدا هستند. بعد از این که با لامبداها بیشتر آشنا شدید درخواهی یافت که آنها یک ابزار قدرتمند و انعطاف‌پذیر در جعبه ابزار LINQ هستند.

ویژگی‌هایی از C# 3.0 که از LINQ پشتیبانی می‌کنند

در این بخش با ساختمان‌های زبانی جدید در C# 3.0 آشنا می‌شوید. هرچند این ویژگی‌های جدید همگی تا حدی با پرس‌وجوهای LINQ به کار برده می‌شوند، محدود به LINQ نبوده و می‌توانند در هر زمینه‌ای که در آنجا آنها را سودمند یافتید به کار برده شوند.

عبارات پرس‌وجو

عبارات پرس‌وجو از یک ترکیب نوشتاری اعلانی مشابه با SQL یا XQuery استفاده می‌کنند تا روی مجموعه‌های IEnumerable پرس‌وجو انجام دهند. در زمان کامپایل ترکیب نوشتاری پرس‌وجو به یک سری فراخوانی متد به پیاده‌سازی مهیا کننده LINQ از متدهای بسط عملگر پرس‌وجوی استاندارد تبدیل می‌شود. برنامه‌ها با مشخص کردن فضای نام مناسب با یک حکم **using** عملگرهای پرس‌وجوی استانداردی را که در دامنه (حوزه دید) هستند کنترل می‌کند. عبارت پرس‌وجوی زیر آرایه‌ای از رشته‌ها را گرفته و آنها را بر طبق کاراکتر نخست رشته گروه‌بندی کرده و گروه‌ها را مرتب می‌کند.

کد نمونه:

```
var query = from str in stringArray
            group str by str[0] into stringGroup
            orderby stringGroup.Key
            select stringGroup;
```

متغیرهایی که نوعشان به صورت ضمنی مدیریت می‌شود (var)

هنگام اعلان و مقداردهی اولیه‌ی یک متغیر به جای تعیین نوع یک متغیر به صورت صریح، می‌توانید از تصریح کننده‌ی **var** استفاده کنید تا به کامپایلر دستور دهید تا نوع را استنتاج کرده و تخصیص دهد؛ این امر در زیر نشان داده شده است:

کد نمونه:

```
var number = 5;
var name = "Virginia";
var query = from str in stringArray
            where str[0] == 'm'
            select str;
```

متغیرهایی که به صورت **var** اعلان می‌شوند عیناً همانند متغیرهایی هستند که نوعشان به صورت صریح تعیین می‌شود. استفاده از **var** ایجاد نوعهای بدون نام را امکان‌پذیر می‌سازد، اما می‌تواند برای هرگونه متغیر محلی به کار برده شود. آرایه‌ها نیز می‌توانند به صورت ضمنی اعلان شوند.

مقداردهنده‌ی شیء و مجموعه

مقداردهنده‌های شیء و مجموعه مقداردهی اولیه‌ی اشیاء را بدون فراخوانی صریح یک سازنده برای شیء امکان‌پذیر می‌سازند. مقداردهنده‌ها هنگامی که داده‌ی منبع را به یک نوع داده‌ی جدید بیرون می‌دهند به طور معمول در عبارات پرس‌وجو به کار برده می‌شوند. با فرض یک کلاس به نام `Customer` با خاصیت‌های عمومی `Name` و `Phone`، مقداردهنده‌ی شیء می‌تواند به صورت کد نمونه‌ی زیر به کار برده شود:

کد نمونه:

```
Customer cust = new Customer {Name = "Mike" ; Phone ={ "555-1212 "};
```

نوعهای بدون نام

یک نوع بی‌نام توسط کامپایلر ایجاد می‌شود و اسم نوع تنها برای کامپایلر قابل دسترس است. نوعهای بدون نام روش سرراستی را برای گروه‌بندی موقت مجموعه‌ای از خاصیت‌ها در یک نتیجه پرس‌وجو در اختیار می‌گذارد

بدون این که نیازی به تعریف یک نوع دارای نام مجزا باشد. انواع بدون نام با یک عبارت جدید و یک مقداردهنده‌ی شیء مقداردهی اولیه می‌شوند؛ این امر در کد نمونه‌ی زیر نشان داده شده است:

کد نمونه:

```
select new {name = cust.Name, phone = cust.Phone};
```

متدهای بسط

یک متد بسط متد استاتیکی است که می‌تواند با یک نوع مرتبط شود طوری که می‌تواند فراخوانی شود انگار که یک متد نمونه بر روی نوع بوده است. این ویژگی شما را قادر می‌سازد تا در عمل متدهای جدیدی را به نوعهای موجود «اضافه» کنید بدون این واقعاً مجبور به تغییر آنها شوید. عملگرهای پرس‌وجوی استاندارد مجموعه‌ای از متدهای بسط هستند که عاملیت پرس‌وجوی LINQ را برای هر نوعی که واسط جنریک IEnumerable(T) را پیاده‌سازی کند در اختیار می‌گذارند.

عبارات لامبدا

یک عبارت لامبدا تابع درون خطی است که از عملگر **=>** استفاده می‌کند تا پارامترهای ورودی را از بدنه‌ی تابع جدا کند و در زمان کامپایل می‌تواند به یک نماینده یا یک درخت عبارت تبدیل شود. در برنامه‌نویسی LINQ، هنگام انجام فراخوان‌های متد مستقیم به عملگرهای پرس‌وجوی استاندارد با عبارات لامبدا مواجه خواهید شد.

خاصیت‌های پیاده‌سازی شده اتوماتیک

«خاصیت‌های پیاده‌سازی شده اتوماتیک»[1] اعلان خاصیت را موجز و مختصرتر می‌سازند. هرگاه خاصیتی را همانند کد نمونه زیر اعلان کنید، کامپایلر یک فیلد پشتیبان خصوصی و بدون نام ایجاد خواهد کرد که به جز از طریق تنظیم کننده و دریافت کننده‌ی خاصیت قابل دسترس نیست.

[1] Auto-Implemented Properties

کد نمونه:

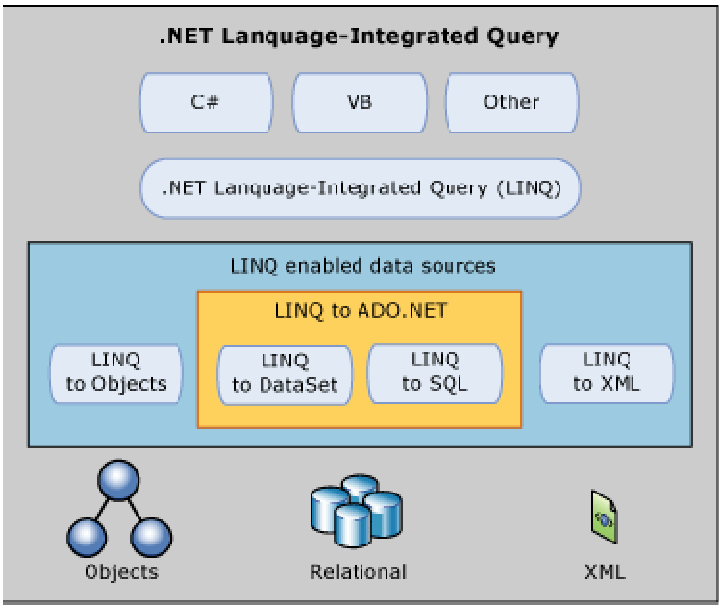
```
public string Name {get; set;}
```

LINQ به ADO.NET

LINQ مجموعه‌ای از عملگرهای پرس‌وجوی استاندارد با اهداف کلی را تعریف می‌کند که شما می‌توانید آنها را در زبان‌های برنامه‌نویسی NET Framework 3.0. مورد استفاده قرار دهید. این عملگرهای پرس‌وجوی استاندارد شما را قادر می‌سازند تا مجموعه‌های درون حافظه‌ای یا جداول واقع در یک پایگاه داده را گزینش، فیلتر و پیمایش کنید. توجه داشته باشید که پرس‌وجوهای LINQ در خود زبان برنامه‌نویسی بیان می‌شوند نه به صورت لیترال‌های رشته‌ای تعبیه شده در کد برنامه. این تغییر معناداری از شیوه‌ای است که اغلب برنامه‌ها روی نسخه‌های پیشین NET Framework. نوشته شده‌اند. نوشتن پرس‌وجوها از میان زبان برنامه‌نویسی شما مزایای کلیدی متعددی را در اختیار می‌گذارد. این کار پرس‌وجوها را با رفع نیاز به استفاده از یک زبان پرس‌وجوی مجزا ساده‌سازی می‌کند. و چنان چه از IDE ویژوال استودیو ۲۰۰۸ یا ۲۰۱۰ استفاده می‌کنید LINQ به شما اجازه می‌دهد تا از مزایای بررسی زمان کامپایل، مدیریت نوع استاتیک و IntelliSense استفاده کنید.

LINQ در میان زمینه‌های متعددی از دسترسی داده در NET Framework. جای داده شده است، از جمله مدل برنامه‌نویسی بی‌اتصال و الگوهای پایگاه داده‌ی SQL Server موجود. این بخش LINQ به ADO.NET، پیاده‌سازی ADO.NET از LINQ، را تشریح می‌کند.

نمودار زیر نگاه اجمالی بر چگونگی وابستگی LINQ به ADO.NET با زبان‌های برنامه‌نویسی سطح بالا، دیگر تکنولوژی‌های LINQ و منابع داده‌ی LINQ فعال می‌اندازد.



LINQ به ADO.NET از دو تکنولوژی LINQ وابسته تشکیل می‌شود: LINQ به DataSet و LINQ به SQL.

LINQ به DataSet

DataSet یکی از مؤلفه‌هایی است که به شکل وسیعی در ADO.NET به کار می‌رود و یک عنصر کلیدی مدل برنامه‌نویسی بی‌اتصال است که ADO.NET روی آن بنا شده است. اگرچه علیرغم این برجستگی و اهمیت، DataSet دارای قابلیت‌های پرس‌وجوی محدودی است.

LINQ به DataSet شما را قادر می‌سازد تا با استفاده از همان قابلیت‌های پرس‌وجویی که برای بسیاری از منابع داده‌ی دیگر در دسترس است امکانات پرس‌وجوی غنی‌تری را در DataSet ایجاد کنید.

LINQ به SQL

LINQ به SQL زیرساخت زمان اجرایی را برای مدیریت داده‌ی رابطه‌ای به صورت اشیاء در اختیار می‌گذارد. در LINQ به SQL، مدل داده‌ی یک پایگاه داده‌ی رابطه‌ای به یک مدل شیئی بیان شده در زبان برنامه‌نویسی مورد نظر توسعه دهنده نگاشته می‌شود. هرگاه برنامه را اجرا کنید، LINQ به SQL پرس‌وجوهای گردآوری شده در

مدل شیئی را به SQL ترجمه کرده و آنها را به منظور اجرا به پایگاه داده ارسال می‌کند. هرگاه پایگاه داده نتایج را برگشت دهد، LINQ به SQL آنها را به اشیائی بازپس می‌دهد که می‌توانید دست‌کاریشان کنید.

LINQ به SQL حاوی پشتیبانی لازم برای رویه‌های ذخیره شده و توابع تعریف شده توسط کاربر در پایگاه داده و نیز توارث در مدل شیئی است.

نگاه اجمالی بر LINQ به ADO.NET

امروزه بسیاری از توسعه دهندگان باید از دو (یا چند) زبان برنامه‌نویسی استفاده کنند: یک زبان سطح بالا برای نوشتن منطق عملی برنامه و لایه‌های نمایشی (زبان‌هایی چون C# یا Visual Basic) و یک زبان پرس‌وجو برای تعامل با پایگاه داده (زبان‌هایی چون Transact-SQL). این امر نیازمند این است که توسعه دهنده در زبان‌های متعددی متبحر باشد و باعث تضادهای زبانی در محیط توسعه نیز می‌شود. برای مثال، برنامه‌ای که یک API دسترسی به داده را برای اجرای یک پرس‌وجو در قبال یک پایگاه داده به کار می‌برد پرس‌وجو را با استفاده از علائم نقل قول به صورت یک لیترال رشته‌ای مشخص می‌کند. این رشته‌ی پرس‌وجو برای کامپایلر غیرقابل اطمینان است و هیچ گونه بررسی به منظور یافتن خطا بر روی آن انجام نمی‌شود، خطاهایی مانند ترکیب نوشتاری نامعتبر یا این که آیا ستون‌ها و ردیف‌هایی که به آنها را ارجاع می‌کند واقعاً وجود دارد یا نه. در آنجا هیچ گونه بررسی نوع برای پارامترهای پرس‌وجو وجود ندارد و هیچ گونه پشتیبانی از **IntelliSense** موجود نیست.

LINQ توسعه دهندگان را قادر می‌سازد تا پرس‌وجوهای مبتنی بر مجموعه را در کد برنامه‌شان شکل دهند بدون این که مجبور به استفاده از یک زبان پرس‌وجوی مجزا شوند. شما می‌توانید پرس‌وجوهای LINQ را در قبال منابع داده‌ی قابل شمارش (یعنی منبع داده‌ای که واسط IEnumerable را پیاده‌سازی کند) مانند ساختارهای داده‌ای درون حافظه‌ای، اسناد XML، پایگاه‌های داده‌ی SQL و اشیاء DataSet بنویسید. با وجود این که این منابع داده‌ی قابل شمارش به شیوه‌های مختلفی پیاده‌سازی می‌شوند، ترکیب نوشتاری و ساخت‌های زبانی یکسانی را ارائه می‌دهند. از آن جایی که پرس‌وجوها می‌توانند در خود زبان برنامه‌نویسی شکل یابند، لزومی ندارد که از زبان پرس‌وجوی دیگری استفاده کنید که به صورت لیترال‌هایی رشته‌ای تعبیه شده‌اند که نمی‌توانند توسط

کامپایلر درک یا بررسی شوند. جای دادن پرس‌وجوها در زبان‌های برنامه‌نویسی برنامه‌نویسان ویژوال استودیو را قادر می‌سازد تا با فراهم‌سازی بررسی ترکیب نوشتاری و نوع زمان کامپایل و پشتیبانی از **IntelliSense**، مثمرثمرتر واقع گردند. این ویژگی‌ها نیاز به اشکال‌زدایی و رفع خطای پرس‌وجو را کاهش می‌دهد.

انتقال داده از جداول SQL به اشیاء درون حافظه‌ای اغلب اوقات خسته کننده و مستعد خطاست. تأمین کننده‌ی LINQ پیاده‌سازی شده توسط LINQ به DataSet و LINQ به SQL داده‌ی منبع را به مجموعه‌های شیئی مبتنی بر IEnumerable تبدیل می‌کند. برنامه‌نویسان همواره داده را به صورت یک مجموعه‌ی IEnumerable مشاهده می‌کنند هم زمانی که پرس‌وجو می‌کنید و هم زمانی که بروزرسانی انجام می‌دهید. پشتیبان کامل **IntelliSense** برای نوشتن پرس‌وجوها در قبال این مجموعه‌ها فراهم شده است.

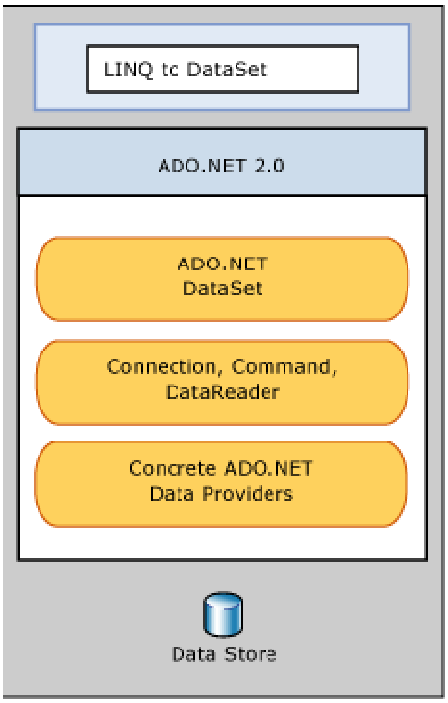
LINQ به DataSet

LINQ به DataSet پرس‌وجو بر روی داده‌ی نهان شده در یک شیء DataSet را ساده‌تر و سریع‌تر می‌سازد. بخصوص، LINQ به DataSet با تواناسازی توسعه دهندگان برای نوشتن پرس‌وجوها از خود زبان برنامه‌نویسی به جای استفاده از یک زبان پرس‌وجوی مجزا، فرایند پرس‌وجو را ساده‌سازی می‌کند. این امر به طور خاص برای توسعه دهندگان ویژوال استودیو سودمند است کسانی که اکنون می‌توانند از مزایای بررسی ترکیب نوشتاری زمان کامپایل، مدیریت نوع استاتیک و پشتیبانی **IntelliSense** ارائه شده توسط ویژوال استودیو در پرس‌وجوهایشان بهره‌برداری کنند.

LINQ به DataSet می‌تواند برای پرس‌وجو بر روی داده‌هایی که از یک یا چند منبع داده گردآوری شده‌اند نیز به کار برده شود. این امر سناریوهای بسیاری را امکان‌پذیر می‌سازد که نیاز به انعطاف‌پذیری در چگونگی نمایش و مدیریت داده دارند سناریوهایی چون پرس‌وجو از داده‌هایی که به صورت محلی انباشته شده‌اند و نهان‌سازی لایه‌ی میانی در برنامه‌های وب. بخصوص، برنامه‌های گزارش‌گیری عمومی، آنالیز و آگهی‌گیری تجاری نیازمند این شیوه از دست‌کاری هستند.

عاملیت LINQ به DataSet عمدتاً از طریق متدهای بسط واقع در کلاس‌های DataRowExtensions و DataTableExtensions ارائه می‌شوند. LINQ به DataSet روی معماری ADO.NET 2.0 موجود ساخته شده و از

آن استفاده می‌کند و به معنای جایگزین ADO.NET 2.0 در کد برنامه نیست. کد ADO.NET 2.0 موجود به عمل کردن در یک برنامه‌ی LINQ به DataSet ادامه خواهد داد. رابطه‌ی LINQ به DataSet به ADO.NET 2.0 و مخزن داده در نمودار زیر نشان داده می‌شود.



DataSet یکی از مؤلفه‌های ADO.NET است که به شکل وسیعی مورد استفاده قرار می‌گیرد. این شیء یک عنصر کلیدی مدل برنامه‌نویسی بی‌اتصال است که ADO.NET روی آن پایه‌ریزی شده است و این شیء (DataSet) شما را قادر می‌سازد تا به صریحاً داده را از منابع داده مختلف نهان سازید. برای لایه نمایشی، DataSet به شکل سفت و محکمی با کنترل‌های GUI برای انقیاد داده یکپارچه شده است. برای لایه‌ی میانی، DataSet نهانگاهی را ارائه می‌دهد که شکل رابطه‌ای داده را ابقا می‌کند و حاوی سرویس‌های پرس‌وجوی ساده و سریع و سرویس‌های کاوش (navigation) سلسله مراتبی است. یک تکنیک رایج به کار رفته برای کاهش دادن تعداد درخواست‌ها روی یک پایگاه داده استفاده از DataSet برای نهان‌سازی در لایه‌ی میانی است. برای مثال، یک برنامه‌ی ASP.NET Web داده‌گرا را در نظر بگیرید. اغلب اوقات بخش معناداری از داده‌ی برنامه به صورت پی‌درپی تغییر نمی‌کند و در طی جلسات و یا برای کاربران مختلف مشترک است. این داده می‌تواند روی سرور وب در حافظه نگهداری شود که تعداد درخواست‌ها در قبال پایگاه داده را کاهش داده و سرعت تعاملات کاربر

را بالا می‌برد. دیگر جنبه‌ی مفید DataSet این است که به یک برنامه اجازه می‌دهد تا زیرمجموعه‌هایی از داده را از یک یا چند منبع داده به فضای برنامه بیاورد. پس از این کار برنامه می‌تواند داده را درون حافظه دستکاری کند درحالی که شکل رابطه‌ای آن را حفظ می‌کند.

علیرغم برجستگی و اهمیتش، DataSet دارای امکانات پرس‌وجوی محدودی است. متد Select می‌تواند برای فیلتر کردن و مرتب‌سازی به کار برده شود و متدهای GetChildRows و GetParentRow می‌توانند برای کاوش سلسله مراتبی به کار برده شوند. با این حال برای هرچیز پیچیده‌تری توسعه دهندگان باید یک پرس‌وجوی سفارشی بنویسند. این امر می‌تواند منجر به برنامه‌هایی گردد که به شکل بدی اجرا شده و نگهداری و پشتیبانی از آنها مشکل است.

پرس‌وجو از اشیاء DataSet با استفاده از LINQ به DataSet

قبل از این که با استفاده از LINQ به DataSet شروع به پرس‌وجو از یک شیء DataSet نمایید باید DataSet را (با داده) پر کنید. روش‌های متعددی برای بارگذاری داده به یک DataSet وجود دارد، روش‌هایی چون استفاده از کلاس DataAdapter یا LINQ به SQL. بعد از این که داده به یک شیء DataSet بارگذاری شد می‌توانید شروع به پرس‌وجو از آن نمایید. تدوین پرس‌وجوها با استفاده از LINQ به DataSet مشابه استفاده از LINQ در قبال دیگر منابع داده‌ی LINQ فعال است. پرس‌وجوها می‌توانند در یک DataSet در قبال جداول منحصر به فرد انجام شوند یا با استفاده از عملگرهای پرس‌وجوی استاندارد Join و GroupJoin در قبال بیش از یک جدول انجام شوند.

پرس‌وجوهای LINQ هم در قبال اشیاء DataSet نوعدار و هم بدون نوع پشتیبانی می‌شوند. اگر الگوی DataSet در زمان طراحی برنامه شناخته شده باشد، یک DataSet نوعدار توصیه می‌شود. در یک DataSet نوعدار، جداول و ردیف‌ها برای هریک از ستون‌ها دارای اعضای نوعدار هستند که این امر پرس‌وجوها را ساده‌تر و خواناتر می‌سازد.

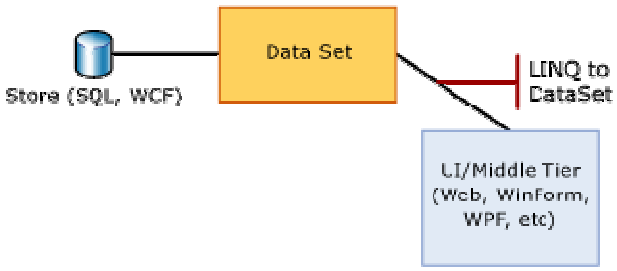
علاوه بر عملگرهای پرس‌وجوی استاندارد پیاده‌سازی شده در **System.Core.dll**، LINQ به DataSet بسط‌های مختص DataSet متعددی را اضافه می‌کند که آن را قادر می‌سازد تا به شکل ساده‌تری روی مجموعه‌ای از اشیاء DataRow پرس‌وجو انجام دهد. این بسط‌های مختص DataSet شامل عملگرهایی برای مقایسه‌ی دنباله‌ای از ردیف‌ها و نیز متدهایی که دسترسی لازم به مقادیر ستون یک DataRow را در اختیار می‌گذارند است.

برنامه‌های N–لایه و LINQ به DataSet

برنامه‌های داده‌ی N–لایه برنامه‌های داده مرکزی هستند که به لایه‌های منطقی متعدد مجزا شده‌اند. یک برنامه‌ی N–لایه‌ی معمول شامل یک لایه‌ی نمایشی، یک لایه‌ی میانی و یک لایه‌ی داده است. تفکیک مؤلفه‌های برنامه به لایه‌های مجزا قابلیت نگهداری و مقیاس‌پذیری برنامه را افزایش می‌دهد.

در برنامه‌های N–لایه، DataSet اغلب اوقات در لایه‌ی میانی برای نهان کردن اطلاعات برای یک برنامه‌ی وب به کار برده می‌شود. عاملیت پرس‌وجوی LINQ به DataSet از طریق متدهای بسط پیاده‌سازی شده و DataSet مربوط به ADO.NET موجود را بسط می‌دهد.

نمودار زیر نشان می‌دهد که چگونه LINQ به DataSet به شیء DataSet وابسته است و با یک برنامه‌ی n–لایه مناسبت دارد:



بارگذاری داده به یک شیء DataSet

یک شیء DataSet قبل از این که بتوانید بوسیله LINQ به DataSet روی آن پرس‌وجو انجام دهید باید در ابتدا با داده پر شود. راههای مختلفی برای پر کردن DataSet از داده وجود دارد. برای مثال، می‌توانید LINQ به SQL را به کار ببرید تا روی پایگاه داده پرس‌وجو انجام داده و نتیجه را به DataSet بارگذاری کنید.

دیگر روش رایج برای بارگذاری داده به یک DataSet استفاده از کلاس DataAdapter برای بازیابی داده از پایگاه داده است. این امر در مثال زیر نشان داده شده است.

مثال

این مثال یک DataAdapter را به کار می‌برد تا روی پایگاه داده‌ی AdventureWorks به منظور اطلاعات فروش از سال ۲۰۰۲ پرس‌وجو انجام داده و نتایج را به یک DataSet بارگذاری کند. بعد از این که DataSet پر از داده شد، شما می‌توانید با استفاده از LINQ به DataSet پرس‌وجوهایی را در قبال آن بنویسید.

کد نمونه: **C#**

```
try
{
    // Create a new adapter and give it a query to fetch sales order, contact,
    // address, and product information for sales in the year 2002. Point
connection
    // information to the configuration setting "AdventureWorks".
    string connectionString = "Data Source=localhost;Initial
Catalog=AdventureWorks;"
        + "Integrated Security=true;";


    SqlDataAdapter da = new SqlDataAdapter(
        "SELECT SalesOrderID, ContactID, OrderDate, OnlineOrderFlag, " +
        "TotalDue, SalesOrderNumber, Status, ShipToAddressID, BillToAddressID " +
        "FROM Sales.SalesOrderHeader " +
        "WHERE DATEPART(YEAR, OrderDate) = @year; " +

        "SELECT d.SalesOrderID, d.SalesOrderDetailID, d.OrderQty, " +
        "d.ProductID, d.UnitPrice " +
        "FROM Sales.SalesOrderDetail d " +
        "INNER JOIN Sales.SalesOrderHeader h " +
        "ON d.SalesOrderID = h.SalesOrderID  " +
        "WHERE DATEPART(YEAR, OrderDate) = @year; " +

        "SELECT p.ProductID, p.Name, p.ProductNumber, p.MakeFlag, " +
        "p.Color, p.ListPrice, p.Size, p.Class, p.Style, p.Weight  " +
        "FROM Production.Product p; " +

        "SELECT DISTINCT a.AddressID, a.AddressLine1, a.AddressLine2, " +
        "a.City, a.StateProvinceID, a.PostalCode " +
        "FROM Person.Address a " +
        "INNER JOIN Sales.SalesOrderHeader h " +
        "ON  a.AddressID = h.ShipToAddressID OR a.AddressID = h.BillToAddressID "
+
        "WHERE DATEPART(YEAR, OrderDate) = @year; " +

        "SELECT DISTINCT c.ContactID, c.Title, c.FirstName, " +
        "c.LastName, c.EmailAddress, c.Phone " +
```

```
        "FROM Person.Contact c " +
        "INNER JOIN Sales.SalesOrderHeader h " +
        "ON c.ContactID = h.ContactID " +
        "WHERE DATEPART(YEAR, OrderDate) = @year;",
    connectionString);

    // Add table mappings.
    da.SelectCommand.Parameters.AddWithValue("@year", 2002);
    da.TableMappings.Add("Table", "SalesOrderHeader");
    da.TableMappings.Add("Table1", "SalesOrderDetail");
    da.TableMappings.Add("Table2", "Product");
    da.TableMappings.Add("Table3", "Address");
    da.TableMappings.Add("Table4", "Contact");

    // Fill the DataSet.
    da.Fill(ds);

    // Add data relations.
    DataTable orderHeader = ds.Tables["SalesOrderHeader"];
    DataTable orderDetail = ds.Tables["SalesOrderDetail"];
    DataRelation order = new DataRelation("SalesOrderHeaderDetail",
                            orderHeader.Columns["SalesOrderID"],
                            orderDetail.Columns["SalesOrderID"], true);
    ds.Relations.Add(order);

    DataTable contact = ds.Tables["Contact"];
    DataTable orderHeader2 = ds.Tables["SalesOrderHeader"];
    DataRelation orderContact = new DataRelation("SalesOrderContact",
                                    contact.Columns["ContactID"],
                                    orderHeader2.Columns["ContactID"], true);
    ds.Relations.Add(orderContact);
}
catch (SqlException ex)
{
    Console.WriteLine("SQL exception occurred: " + ex.Message);
}
```

چگونگی ایجاد یک پروژه LINQ به DataSet در ویژوال استودیو

انواع مختلف پروژه‌های LINQ نیازمند فضاهای نام وارد شده‌ی بخصوص (در ویژوال بیسیک) یا احکام **using** (در C#) و ارجاعات بخصوص هستند. حداقل شرط لازم یک ارجاع به System.Core.dll و یک حکم **using** برای فضای نام System.Linq است. به طور پیش فرض، اگر یک پروژه‌ی تازه‌ی Visual C# 2008 را ایجاد کنید اینها آماده خواهند شد. LINQ به DataSet نیازمند یک ارجاع به System.Data.dll و System.Data.DataSetExtensions.dll و یک حکم **using** است.

اگر در حال بروزرسانی یک پروژه از یک نسخه‌ی پیشین ویژوال استودیو (مانند ۲۰۰۵) هستید، ممکن است لازم شود تا این ارجاعات مرتبط با LINQ را به طور دستی مهیا کنید. ضمناً ممکن است که لازم شود تا به طور دستی پروژه را تنظیم کنید تا .NET Framework نسخه‌ی 3.0 را هدف‌گیری کند.

نکته:

اگر در حال ساخت برنامه از طریق خط فرمان هستید، باید به طور دستی به DLLهای وابسته واقع در مسیر *drive*:\Program Files\Reference Assemblies\Microsoft\Framework\v3.5 ارجاع کنید.

برای هدف‌گیری .NET Framework 3.5

۱. در ویژوال استودیو ۲۰۰۸، یک پروژه جدید C# را ایجاد کنید.

۲. برای یک پروژه‌ی C#، منوی **Project** را کلیک کرده و سپس **Properties** را کلیک کنید.

۳. در صفحه‌ی خاصیت **Application**، مورد .NET Framework 3.5 را در لیست آبشاری **Target Framework** انتخاب کنید.

۴. روی منوی **Project**، مورد **Add Reference** را کلیک کرده، برگه‌ی .NET را کلیک کنید، سپس **System.Core** را یافته و **OK** را کلیک کنید.

۵. یک حکم **using** برای فضای نام **System.Linq** به پروژه یا فایل کد منبع‌تان اضافه کنید.

برای فعال کردن عاملیت LINQ به DataSet

۱. در صورت لزوم، مراحل بخش قبل را برای افزودن یک ارجاع به **System.Core.dll** و یک حکم **using** برای فضای نام **System.Linq** پی‌گیری نمایید.

۲. در C#، منوی **Project** را کلیک کرده و پس از آن **Add Reference** را کلیک کنید.

۳. در جعبه‌ی محاوره‌ای **Add Reference**، برگه‌ی **.NET** را کلیک کنید. **System.Data** و **System.Data.DataSetExtensions** را یافته آنها را کلیک کرده و سپس دکمه‌ی **OK** را کلیک کنید.

۴. یک حکم **using** برای **System.Data** به پروژه یا فایل کد منبع‌تان اضافه کنید.

۵. یک ارجاع به **System.Data.DataSetExtensions.dll** برای عاملیت LINQ به DataSet اضافه کنید. در صورتی که از قبل وجود نداشته باشد یک ارجاع به **System.Data.dll** اضافه کنید.

۶. به طور دلخواه، بسته به چگونگی اتصال به پایگاه داده یک حکم **using** برای **System.Data.Common** یا **System.Data.SqlClient** اضافه کنید.

پرس‌وجوها در LINQ به DataSet

یک عملیات پرس‌وجوی LINQ از سه عمل تشکیل می‌شود: بدست آوردن منبع یا منابع داده، ایجاد پرس‌وجو و اجرای پرس‌وجو.

منابع داده‌ای که واسط جنریک IEnumerable(T) را پیاده‌سازی می‌کنند می‌توانند از طریق LINQ پرس‌وجو شوند. فراخوانی AsEnumerable روی یک DataSet شیئی را برمی‌گرداند که واسط جنریک IEnumerable(T) را پیاده‌سازی می‌کند؛ این واسط جنریک به عنوان منبع داده برای پرس‌وجوهای LINQ به DataSet عمل می‌کند.

در پرس‌وجو، در واقع اطلاعاتی را تعیین می‌کنید که قصد بازیابی آنها را از منبع داده دارید. یک پرس‌وجو می‌تواند چگونگی مرتب شدن، گروه‌بندی شدن و شکل یافتن اطلاعات برگشتی را نیز تعیین می‌کند. در LINQ، یک پرس‌وجو در یک متغیر ذخیره می‌شود. اگر پرس‌وجو طراحی می‌شود تا دنباله‌ای از مقادیر را برگشت دهد، خود متغیر پرس‌وجو باید یک نوع قابل شمارش باشد. این متغیر پرس‌وجو هیچ عملی را اتخاذ نکرده و هیچ گونه داده‌ای را برگشت نمی‌دهد؛ این متغیر تنها اطلاعات پرس‌وجو را ذخیره می‌کند. بعد از این که پرس‌وجویی را ایجاد کردید باید آن را اجرا کنید تا همه‌ی داده‌ها را بازیابی کنید.

در پرس‌وجویی که دنباله‌ای از مقادیر را برگشت می‌دهد، متغیر پرس‌وجو خودش هرگز نتایج پرس‌وجو را نگهداری نمی‌کند و تنها فرامین پرس‌وجو را ذخیره می‌کند. اجرای پرس‌وجو تا زمانی که در یک حلقه‌ی **foreach** روی نتایج متغیر پرس‌وجو حرکت شود به تعویق می‌افتد. این امر اجرای معوق خوانده می‌شود؛ یعنی اجرای پرس‌وجو در برخی از زمان‌های بعد از ایجاد پرس‌وجو رخ می‌دهد. این حرف به معنای این است که شما می‌توانید یک پرس‌وجو را هر زمان و هرقدر که بخواهید اجرا کنید. برای مثال این امر زمانی که دارای پایگاه داده‌ای هستید که درحال بروزرسانی توسط برنامه‌های دیگر است مفید است. در برنامه‌ی خود، می‌توانید

پرس‌وجویی را ایجاد کنید که آخرین اطلاعات را بازیابی کرده و به صورت مکرر پرس‌وجو را اجرا کند در حالی که در هر بار اطلاعات بروز شده را برمی‌گرداند.

در تقابل با پرس‌وجوهای معوق، که دنباله‌ای از مقادیر را برمی‌گردانند، پرس‌وجوهایی که یک مقدار یکتا را برمی‌گردانند به صورت فوری اجرا می‌شوند. چند نمونه از پرس‌وجوهای یکتا **Count**، **Max**، **Average** و **First** هستند. اینها بلافاصله اجرا می‌شوند زیرا برای محاسبه‌ی نتیجه‌ی یکتا نتایج پرس‌وجو مورد نیازند. برای مثال، برای یافتن میانگین نتایج پرس‌وجو بایستی پرس‌وجو اجرا شود تا این که تابع میانگین‌یابی دارای داده‌ی ورودی باشد تا با آن کار کند. شما می‌توانید متدهای ToList(TSource) یا ToArray(TSource) را بر روی یک پرس‌وجو به کار ببرید تا بر اجرای فوری پرس‌وجویی که نتیجه‌ی یکتایی را تولید نمی‌کند تأکید کنید. زمانی که قصد نهان کردن نتایج پرس‌وجو را دارید این تکنیک برای تحمیل اجرای فوری می‌تواند مفید باشد.

پرس‌وجوها

پرس‌وجوهای LINQ به DataSet می‌توانند با دو ترکیب نوشتاری متفاوت تدوین (فرموله) شوند: ترکیب نوشتاری عبارت پرس‌وجو و ترکیب نوشتاری پرس‌وجوی مبتنی بر متد.

ترکیب نوشتاری عبارت پرس‌وجو

عبارات پرس‌وجو یک ترکیب نوشتاری پرس‌وجوی اعلانی هستند. این ترکیب نوشتاری یک توسعه دهنده را قادر می‌سازند تا پرس‌وجوها را در C# یا Visual Basic در قالب مشابهی با SQL بنویسد. با استفاده از ترکیب نوشتاری عبارت پرس‌وجو، شما حتی می‌توانید با کمترین کد عملیات‌های فیلترینگ، مرتب‌سازی و گروه‌بندی پیچیده را بر روی منابع داده انجام دهید.

ترکیب نوشتاری عبارت پرس‌وجو یکی از مشخصه‌های تازه در C# 3.0 و Visual Basic 2008 است. اگرچه، CLR نمی‌تواند خود ترکیب نوشتاری عبارت پرس‌وجو را بخواند. ازاینرو، در زمان کامپایل، عبارات پرس‌وجو به چیزی ترجمه می‌شوند که CLR آن را می‌فهمد: فراخوانی متدها. این متدها با عنوان *عملگرهای پرس‌وجوی*

استاندارد شناخته می‌شوند. به عنوان یک توسعه دهنده، به جای استفاده از ترکیب نوشتاری پرس‌وجو شما دارای گزینه‌ی فراخوانی مستقیم آنها با استفاده از ترکیب نوشتاری متد هستید.

مثال از متد Select استفاده می‌کند تا همه‌ی رکوردهای جدول *Product* را برگردانده و اسامی محصولات را نمایش دهد.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable products = ds.Tables["Product"];

IEnumerable<DataRow> query =
    from product in products.AsEnumerable()
    select product;

Console.WriteLine("Product Names:");
foreach (DataRow p in query)
{
    Console.WriteLine(p.Field<string>("Name"));
}
```

ترکیب نوشتاری پرس‌وجوی مبتنی بر متد

روش دیگر برای فرموله کردن پرس‌وجوهای LINQ به DataSet استفاده از پرس‌وجوهای مبتنی بر متد است. این ترکیب نوشتاری پرس‌وجوی مبتنی بر متد دنباله‌ای از فراخوانی‌های متد مستقیم به متدهای عملگر LINQ است در حالی که عبارات لامبدا به عنوان پارامتر ارسال می‌شوند.

مثال از متد Select استفاده می‌کند تا همه‌ی رکوردهای جدول *Product* را برگردانده و اسامی محصولات را نمایش دهد.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable products = ds.Tables["Product"];
```

```
var query = products.AsEnumerable().
    Select(product => new
                {
                    ProductName = product.Field<string>("Name"),
                    ProductNumber = product.Field<string>("ProductNumber"),
                    Price = product.Field<decimal>("ListPrice")
                });

Console.WriteLine("Product Info:");
foreach (var productInfo in query)
{
    Console.WriteLine("Product name: {0} Product number: {1} List price: ${2} ",
        productInfo.ProductName, productInfo.ProductNumber, productInfo.Price);
}
```

ترکیب پرس‌وجوها

همان طور که پیش از این اشاره شد، زمانی که پرس‌وجو طراحی می‌شود تا دنباله‌ای از مقادیر را برگشت دهد خود متغیر پرس‌وجو تنها فرامین پرس‌وجو را ذخیره می‌کند. اگر پرس‌وجو شامل متدی نباشد که باعث اجرای فوری شود، اجرای واقعی پرس‌وجو تا زمانی که در یک حلقه‌ی **foreach** روی نتایج متغیر پرس‌وجو حرکت کنید به تعویق می‌افتد. اجرای معوق پرس‌وجوهای متعدد را قادر می‌سازد تا با هم ترکیب شوند یا یک پرس‌وجو بسط داده شود. هرگاه یک پرس‌وجو بسط داده می‌شود، این پرس‌وجو اصلاح می‌شود تا شامل عملیات‌های جدید شود و اجرای نهایی تغییرات را بازتاب خواهد داد. در مثال زیر، پرس‌وجوی نخست تمامی محصولات را برمی‌گرداند. پرس‌وجوی دوم با استفاده از **Where** پرس‌وجوی نخست را بسط می‌دهد تا تمامی محصولات با اندازه‌ی "L" را برگرداند.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable products = ds.Tables["Product"];

IEnumerable<DataRow> productsQuery =
    from product in products.AsEnumerable()
    select product;

IEnumerable<DataRow> largeProducts =
    productsQuery.Where(p => p.Field<string>("Size") == "L");
```

```
Console.WriteLine("Products of size 'L':");
foreach (DataRow product in largeProducts)
{
    Console.WriteLine(product.Field<string>("Name"));
}
```

بعد از این که یک پرس‌وجو اجرا شد، هیچ گونه پرس‌وجوی دیگری نمی‌تواند با آن ترکیب شود و همه‌ی پرس‌وجوهای بعدی از عملگرهای LINQ درون حافظه‌ای استفاده خواهند کرد. هرگاه در یک دستور **foreach** روی نتایج متغیر پرس‌وجو حرکت کنید یا با یک فراخوان به یکی از عملگرهای تبدیل LINQ که باعث اجرای فوری می‌شود پرس‌وجو اجرا می‌شود. این عملگرها شامل اینها هستند: ToList(TSource)، ToArray(TSource)، ToLookup و ToDictionary.

در مثال زیر، پرس‌وجوی نخست تمامی محصولات مرتب شده به واسطه‌ی لیست قیمتشان را برمی‌گرداند. متد ToArray(TSource) برای تحمیل اجرای فوری پرس‌وجو به کار برده شده است:

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable products = ds.Tables["Product"];

IEnumerable<DataRow> query =
    from product in products.AsEnumerable()
    orderby product.Field<Decimal>("ListPrice") descending
    select product;

// Force immediate execution of the query.
IEnumerable<DataRow> productsArray = query.ToArray();

Console.WriteLine("Every price from highest to lowest:");
foreach (DataRow prod in productsArray)
{
    Console.WriteLine(prod.Field<Decimal>("ListPrice"));
}
```

پرس‌وجو از اشیاء DataSet

بعد از این که یک شیء DataSet با داده پر شد، می‌توانید شروع به پرس‌وجو از آن کنید. فرموله کردن پرس‌وجوها با LINQ به DataSet مشابه استفاده از LINQ در قبال دیگر منابع LINQ فعال است. هرچند به یاد داشته باشید که هنگام استفاده از پرس‌وجوهای LINQ روی یک شیء DataSet به جای فهرستی از یک نوع سفارشی، شما در حال پرس‌وجو از فهرستی از اشیاء DataRow هستید. این حرف به معنای این است که شما می‌توانید هر یک از اعضای کلاس DataRow را در پرس‌وجوهای LINQ مورد استفاده قرار دهید. این امر به شما اجازه می‌دهد تا پرس‌وجوهای پربار و پیچیده‌ای را ایجاد کنید.

همانند دیگر پیاده‌سازی‌های LINQ، پرس‌وجوهای LINQ به DataSet را می‌توانید به دو روش مختلف ایجاد کنید: ترکیب نوشتاری عبارت پرس‌وجو و ترکیب نوشتاری پرس‌وجوی مبتنی بر متد. شما می‌توانید ترکیب نوشتاری عبارت پرس‌وجو یا ترکیب نوشتاری پرس‌وجوی مبتنی بر متد را به کار ببرید تا در قبال جداول منحصربه فرد واقع در یک DataSet، چندین جدول واقع در یک DataSet یا جداول واقع در یک DataSet نوعدار پرس‌وجوهایی را صورت دهید.

پرس‌وجوها از یک جدول واحد

پرس‌وجوهای LINQ روی منابع داده‌ای که واسط جنریک IEnumerable(T) یا واسط IQueryable را پیاده‌سازی می‌کنند کار می‌کنند. کلاس DataTable هیچ یک از این دو واسط را پیاده‌سازی نمی‌کند، ازاینرو اگر در ضابطه‌ی **From** یک پرس‌وجوی LINQ قصد استفاده از DataTable به عنوان یک منبع را دارید باید متد AsEnumerable را فراخوان کنید.

مثال زیر تمامی سفارشات برخط را از جدول SalesOrderHeader گرفته و ID سفارش، تاریخ سفارش و تعداد سفارش را به خروجی کنسول می‌دهد.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
```

```
FillDataSet(ds);

DataTable orders = ds.Tables["SalesOrderHeader"];

var query =
    from order in orders.AsEnumerable()
    where order.Field<bool>("OnlineOrderFlag") == true
    select new
           {
               SalesOrderID = order.Field<int>("SalesOrderID"),
               OrderDate = order.Field<DateTime>("OrderDate"),
               SalesOrderNumber = order.Field<string>("SalesOrderNumber")
           };

foreach (var onlineOrder in query)
{
    Console.WriteLine("Order ID: {0} Order date: {1:d} Order number: {2}",
        onlineOrder.SalesOrderID,
        onlineOrder.OrderDate,
        onlineOrder.SalesOrderNumber);
}
```

متغیر محلی پرس‌وجو با یک عبارت پرس‌وجو مقداردهی اولیه می‌شود که این عبارت پرس‌وجو با اعمال یک یا چند عملگر پرس‌وجو از عملگرهای پرس‌وجوی استاندارد یا در مورد LINQ به DataSet از عملگرهای مختص کلاس DataSet روی یک یا چند منبع اطلاعاتی عمل می‌کند. عبارت پرس‌وجو در مثال قبل از دو عملگر پرس‌وجوی استاندارد استفاده می‌کند: **Where** و **Select**.

ضابطه‌ی **Where** دنباله را بر اساس صحت یا عدم صحت یک شرط فیلتر می‌کند، که در این مورد *OnlineOrderFlag* به **true** تنظیم شده است. عملگر **Select** شیء قابل شمارشی را تخصیص داده و برگشت می‌دهد که آرگومان‌های ارسالی به عملگر را می‌گیرد. در مثال قبل، یک نوع بدون نام همراه با سه خاصیت ایجاد شده است: *SalesOrderID*، *OrderDate* و *SalesOrderNumber*. مقادیر این سه خاصیت با مقادیر ستون‌های *SalesOrderID*، *OrderDate* و *SalesOrderNumber* از جدول *SalesOrderHeader* تنظیم می‌شوند.

پس از آن حلقه‌ی **foreach** شیء قابل شمارش برگشت داده شده توسط **Select** را سرشماری کرده و نتایج پرس‌وجو را بدست می‌دهد. از آن جایی پرس‌وجو یک نوع Enumerable است، یعنی واسط IEnumerable(T) را پیاده‌سازی می‌کند، ارزیابی پرس‌وجو تا زمانی که با استفاده از حلقه‌ی **foreach** روی نتایج متغیر پرس‌وجو

حرکت شود به تعویق می‌افتد. ارزیابی پرس‌وجوی معوق به پرس‌وجوها اجازه می‌دهد تا همانند مقادیری که می‌توانند چندین بار ارزیابی شده و هربار به صورت بالقوه نتایج متفاوتی را به بار می‌دهند نگهداری شوند.

متد Field دسترسی لازم به مقادیر ستون یک DataRow را در اختیار می‌گذارد و متد SetField مقادیر ستون‌ها را در یک DataRow تنظیم می‌کند. هم متد Field و هم متد SetField انواع تهی‌پذیر را مدیریت می‌کنند از اینرو لزومی ندارد که به طور صریح به منظور یافتن مقادیر تهی بررسی انجام دهید. هردو متد، متدهای جنریک نیز هستند که به معنای این است که لزومی ندارد تا نوع برگشتی را قالب‌بندی (تبدیل نوع صریح) کنید. شما می‌توانید از اکسسور ستون از پیش موجود در DataRow استفاده کنید (مثلاً `o["OrderDate"]`) اما انجام با این کار نیازمند این خواهید بود تا شیء برگشتی را به نوع درخور قالب‌بندی (تبدیل نوع صریح) کنید. اگر ستون مقدار تهی را بپذیرد لازم است تا با استفاده از متد IsNull بررسی کنید که آیا مقدار تهی است یا نه.

توجه داشته باشید که نوع داده‌ی مشخص شده در پارامتر جنریک *T* متد Field و متد SetField باید با نوع مقدار درزیر قرار کرفته مطابقت کند وإلا یک استثنای InvalidCastException پرتاب خواهد شد. اسم ستون مشخص شده نیز باید با اسم یک ستون واقع در DataSet مطابقت کند وإلا یک استثنای ArgumentException پرتاب خواهد شد. در هردو حالت، استثناء هنگام اجرا شدن پرس‌وجو در زمان اجرا پرتاب می‌شود.

پرس‌وجو از جداول متقاطع

علاوه بر پرس‌وجو از یک جدول واحد، شما می‌توانید در LINQ به DataSet پرس‌وجوهایی را از چندین جدول متقاطع انجام دهید. این کار با استفاده از یک پیوند (یا یک عملیات *join*) انجام می‌گیرد. یک پیوند رابطه‌ی اشیاء واقع در یک منبع داده با اشیائی است که یک ویژگی مشترک را در منبع داده‌ی دیگر به اشتراک می‌گذارند، مانند یک محصول یا ID محصول. در برنامه‌نویسی شیءگرا، کاوش و هدایت روابط مابین اشیاء نسبتاً راحت است زیرا هر شیئی دارای عضوی است که به شیء دیگری ارجاع می‌کند. هرچند در جداول پایگاه داده‌ی بیرونی، کاوش و هدایت کردن روابط به این سرراستی نیست. جداول پایگاه داده حاوی روابط توکار نیستند. در این موارد، عملیات **join** می‌تواند برای مطابقت دادن عناصر از هر یک از منابع به کار برده شود. برای مثال، با فرض دو جدول که حاوی اطلاعات محصول و اطلاعات فروش هستند، شما می‌توانید یک

عملیات اتصال (**join**) را به کار ببرید تا اطلاعات فروش و محصول را به منظور یافتن سفارشات فروش یکسان مطابقت دهید.

چارچوب LINQ دو عملگر اتصال در اختیار می‌گذارد، **Join** و **GroupJoin**. این عملگرها *اتصال‌های برابری* را انجام می‌دهند: یعنی اتصال‌هایی که دو منبع داده را تنها زمانی مطابقت می‌دهد که کلیدهای‌شان برابر باشد. (در مقابل، Transact-SQL از عملگرهایی اتصالی غیر از **equals**، مانند عملگر **less than** پشتیبانی می‌کند.)

در جملات پایگاه داده‌ی رابطه‌ای، **Join** یک اتصال درونی را پیاده‌سازی می‌کند. پیوند درونی نوعی از پیوند است که در آن تنها آن اشیائی که دارای یک تطابق در مجموعه داده‌ی مقابلند برگشت داده می‌شوند.

عملگرهای **GroupJoin** هیچ گونه معادل مستقیمی در جملات پایگاه داده‌ی رابطه‌ای ندارند؛ آنها یک فرامجموعه از پیوندهای درونی و پیوندهای بیرونی چپی را پیاده‌سازی می‌کنند. یک پیوند بیرونی چپی پیوندی است که هریک از عناصر مجموعه‌ی نخست (چپی) را برمی‌گرداند حتی اگر دارای هیچ گونه عنصر همبسته‌ای در مجموعه‌ی دوم نباشد.

مثال

مثال زیر یک اتصال سنتی مابین جداول *SalesOrderHeader* و *SalesOrderDetail* از پایگاه داده‌ی AdventureWorks انجام می‌دهد تا سفارشات آنلاین را از ماه آگوست بدست آورد.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable orders = ds.Tables["SalesOrderHeader"];
DataTable details = ds.Tables["SalesOrderDetail"];

var query =
    from order in orders.AsEnumerable()
    join detail in details.AsEnumerable()
    on order.Field<int>("SalesOrderID") equals
        detail.Field<int>("SalesOrderID")
    where order.Field<bool>("OnlineOrderFlag") == true
    && order.Field<DateTime>("OrderDate").Month == 8
```

```
    select new
           {
               SalesOrderID =
                   order.Field<int>("SalesOrderID"),
               SalesOrderDetailID =
                   detail.Field<int>("SalesOrderDetailID"),
               OrderDate =
                   order.Field<DateTime>("OrderDate"),
               ProductID =
                   detail.Field<int>("ProductID")
           };


foreach (var order in query)
{
    Console.WriteLine("{0}\t{1}\t{2:d}\t{3}",
        order.SalesOrderID,
        order.SalesOrderDetailID,
        order.OrderDate,
        order.ProductID);
}
```

پرس‌وجو از اشیاء DataSet نوعدار

اگر الگوی DataSet در زمان طراحی برنامه شناخته شده باشد، به شما توصیه می‌کنم که هنگام استفاده از LINQ به DataSet از یک DataSet نوعدار استفاده کنید. یک DataSet نوعدار کلاسی است که از یک DataSet مشتق می‌شود. بدین سبب، این شیء تمامی متدها، رویدادها و خاصیت‌های یک DataSet را به ارث می‌برد. علاوه بر اینها، یک DataSet نوعدار متدها، خاصیت‌ها و رویدادهایی را که به شکل پرقدرتی نوعدار شده‌اند در اختیار می‌گذارد. این حرف به معنای این است که شما می‌توانید به جای استفاده از متدهای مبتنی بر مجموعه، بوسیله‌ی اسم به جداول و ستون‌ها دسترسی پیدا کنید. این کار پرس‌وجوها را ساده‌تر و خواناتر می‌کند.

LINQ به DataSet از پرس‌وجو کردن از یک DataSet نوعدار نیز پشتیبانی می‌کند. با یک DataSet نوعدار، لزومی ندارد که برای دسترسی به داده‌ی ستون‌ها از متدهای جنریک Field یا SetField استفاده کنید. در زمان کامپایل اسامی خاصیت‌ها قابل دسترسی هستند زیرا اطلاعات نوع ضمیمه‌ی DataSet شده است. LINQ به DataSet دسترسی به مقادیر ستون را همانند نوع درست در اختیار می‌گذارند، طوری که خطاهای عدم تطابق به جای زمان اجرا در هنگام کامپایل شدن کد گیر می‌افتند.

قبل از این که بتوانید شروع به پرس‌وجو کردن از یک DataSet نوعدار کنید، باید با استفاده از DataSet Designer کلاس را در ویژوال استودیو ۲۰۰۸ تولید کنید. بخش «چگونگی ایجاد یک DataSet نوعدار» را ملاحظه کنید.

مثال

مثال زیر پرس‌وجویی را نشان می‌دهد که روی یک DataSet نوعدار انجام شده است:

کد نمونه: **C#**

```
var query = from o in orders
            where o.OnlineOrderFlag == true
            select new { o.SalesOrderID,
                         o.OrderDate,
                         o.SalesOrderNumber };

foreach(var order in query)
{
    Console.WriteLine("{0}\t{1:d}\t{2}",
order.SalesOrderID,
order.OrderDate,
order.SalesOrderNumber);
}
```

چگونگی ایجاد یک DataSet نوعدار

شما می‌توانید با استفاده از **Data Source Configuration Wizard** ابزار DataSet Designer یک DataSet نوعدار ایجاد کنید.

نکته:

شما می‌توانید با فراخوان سازنده‌ی DataSet نمونه‌ای از یک DataSet را ایجاد کنید. به طور دلخواه یک آرگومان نام را مشخص کنید. اگر نامی برای DataSet تعیین نکنید، اسم آن به "NewDataSet" تنظیم خواهد شد.

شما می‌توانید یک DataSet جدید را بر اساس یک DataSet موجود ایجاد کنید. DataSet جدید می‌تواند یک کپی دقیق از DataSet موجود باشد؛ یک کلونی از DataSet که ساختار یا الگوی رابطه‌ای را کپی می‌کند اما

شامل هیچ داده‌ای از DataSet موجود نیست؛ یا اگر از متد GetChanges استفاده شود زیرمجموعه‌ای از DataSet که تنها شامل ردیف‌های (رکوردهای) تغییر یافته‌ای از DataSet موجود است.

کد نمونه‌ی زیر چگونگی ایجاد نمونه‌ای از یک DataSet را نشان می‌دهد.

کد نمونه: **C#**

```
DataSet customerOrders = new DataSet("CustomerOrders");
```

ایجاد اشیاء DataSet نوعدار با Data Source Configuration Wizard یا با DataSet Designer

برای ایجاد یک DataSet بوسیله‌ی Data Source Configuration Wizard

۱. روی منوی **Data**، مورد **Add New Data Source** را کلیک کنید تا **Data Source Configuration Wizard** راه‌اندازی شود.

۲. **Database** را روی صفحه‌ی **Choose a Data Source Type** انتخاب کنید.

۳. ویزارد را تکمیل کنید که پس از اتمام کار یک DataSet نوعدار به پروژه‌ی شما اضافه خواهد شد.

برای ایجاد یک DataSet بوسیله‌ی DataSet Designer

۱. روی منوی **Project**، مورد **Add New Item** را کلیک کنید.

۲. از جعبه‌ی محاوره‌ای **Add New Item**، مورد DataSet را انتخاب کنید.

۳. یک نام برای DataSet تایپ کنید.

۴. **Add** را کلیک کنید.

DataSet به پروژه اضافه شده و در **DataSet Designer** باز می‌شود.

۵. گزینه‌هایی را از برگه‌ی **DataSet** متلق به **Toolbox** به روی طراح بکشانید.

—یا—

گزینه‌هایی را از یک اتصال فعال واقع در **Server Explorer/Database Explorer** به روی **DataSet Sesigner** بکشانید (درگ کنید).

مقایسه اشیاء DataRow

LINQ چند عملگر مجموعه را برای مقایسه‌ی عناصر منبع به منظور آگاهی از برابری یا عدم برابری آنها تعریف می‌کند. LINQ عملگرهای مجموعه‌ی زیر را در اختیار می‌گذارد:

- **Distinct**
- **Union**
- **Intersect**
- **Except**

این عملگرها با فراخوانی متدهای GetHashCode و Equals بر روی هر یک از مجموعه‌های عناصر، عناصر منبع را مقایسه می‌کنند. در مورد یک DataRow، این عملگرها یک مقایسه‌ی ارجاعی را انجام می‌دهند که این کار معمولاً رفتار ایده‌آلی برای عملیات‌های مجموعه‌ای روی داده‌های جدولی نیست. برای عملیات‌هایی که روی مجموعه‌ها انجام می‌شوند، به طور معمول می‌خواهید تعیین کنید که آیا مقادیر عنصر مساوی هستند و ارجاعات عنصری نیستند. ازاینرو، کلاس DataRowComparer به LINQ به DataSet اضافه شده است. این کلاس می‌تواند برای مقایسه‌ی مقادیر ردیف‌ها (رکوردها) به کار برده شود.

این کلاس نمی‌تواند به طور مستقیم نمونه‌سازی شود؛ درعوض، بایستی خاصیت Default برای برگرداندن نمونه‌ای از DataRowComparer به کار برده شود. پس از آن متد Equals(DataRow, DataRow) فراخوان می‌شود و دو شیء DataRow که با هم مقایسه خواهند شد به صورت پارامترهای ورودی ارسال می‌شوند. متد Equals(DataRow, DataRow) در صورتی که مجموعه‌ی درخواست شده‌ی مقادیر ستون در هردو شیء DataRow برابر باشد مقدار **true** را برمی‌گرداند؛ در غیر این صورت، مقدار برگشتی **false** خواهد شد.

مثال

این مثال عملگر **Intersect** را برای برگرداندن تماسهای ظاهر شده در هر دو جدول به می‌برد.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);
```

```
DataTable contactTable = ds.Tables["Contact"];

// Create two tables.
IEnumerable<DataRow> query1 = from contact in contactTable.AsEnumerable()
                              where contact.Field<string>("Title") == "Ms."
                              select contact;

IEnumerable<DataRow> query2 = from contact in contactTable.AsEnumerable()
                              where contact.Field<string>("FirstName") == "Sandra"
                              select contact;


DataTable contacts1 = query1.CopyToDataTable();
DataTable contacts2 = query2.CopyToDataTable();

// Find the intersection of the two tables.
var contacts = contacts1.AsEnumerable().Intersect(contacts2.AsEnumerable(),
                                                  DataRowComparer.Default);

Console.WriteLine("Intersection of contacts tables");
foreach (DataRow row in contacts)
{
    Console.WriteLine("Id: {0} {1} {2} {3}",
        row["ContactID"], row["Title"], row["FirstName"], row["LastName"]);
}
```

ایجاد یک DataTable از یک پرس‌وجو

انقیاد داده یکی از کاربردهای رایج شیء DataTable است. متد CopyToDataTable نتایج یک پرس‌وجو را گرفته و داده را به یک DataTable کپی می‌کند که این شیء (یعنی DataTable) پس از آن می‌تواند برای انقیاد داده به کار برده شود. هرگاه عملیات‌ها بر روی داده‌ها انجام شدند، شیء DataTable جدید با شیء DataTable منبع ادغام می‌شود.

متد CopyToDataTable فرایند زیر را برای ایجاد یک DataTable از یک پرس‌وجو مورد استفاده قرار می‌دهد:

۱. متد CopyToDataTable یک DataTable را از روی جدول منبع (یک شیء DataTable که واسط جنریک IQueryable(T) را پیاده‌سازی می‌کند) تکثیر می‌کند. منبع IEnumerable معمولاً از یک عبارت LINQ به DataSet یا از یک پرس‌وجوی متد نشأت می‌گیرد.

۲. الگوی DataTable تکثیر شده از ستون‌های اولین شیء DataRow شمارش شده واقع در جدول منبع ساخته شده و اسم جدول تکثیر شده اسم جدول منبع همراه با واژه‌ی "query" الحاق شده به آن است.

۳. برای هر یک از ردیف‌های (رکوردهای) واقع در جدول منبع، محتوای ردیف به یک شیء DataRow جدید کپی می‌شود، که پس از آن در جدول تکثیر شده درج می‌شود. خاصیت‌های RowState و RowError در طی عملیات کپی ابقا می‌شوند. اگر اشیاء DataRow واقع در منبع از جداول متفاوتی باشند یک استثنای ArgumentException پرتاب می‌شود.

۴. بعد از این که تمامی اشیاء DataRow در جدول قابل پرس‌وجوی ورودی کپی شد شیء DataTable تکثیر شده برگردانده می‌شود. اگر دنباله‌ی منبع حاوی هیچ یک از اشیاء DataRow نباشد، متد یک شیء DataTable تهی را برمی‌گرداند.

توجه داشته باشید که فراخوانی متد CopyToDataTable باعث خواهد شد که پرس‌وجوی مقید شده به جدول منبع اجرا شود.

هرگاه متد CopyToDataTable یا با یک نوع ارجاعی تهی یا یک نوع مقداری تهی‌پذیر واقع در یک ستون واقع در جدول منبع مواجه شود، مقدار را با فیلد Value جایگزین می‌کند. با این روش، مقادیر تهی در شیء DataTable برگشتی به درستی مدیریت می‌شوند.

مثال

مثال زیر جدول SalesOrderHeader را به منظور یافتن سفارشات بعد از ۸ آگوست ۲۰۰۱ جستار کرده و متد CopyToDataTable را برای ایجاد یک DataTable از آن پرس‌وجو مورد استفاده قرار می‌دهد. پس از آن این DataTable به یک شیء BindingSource که به عنوان پراکسی برای یک DataGridView عمل می‌کند مقید می‌شود.

کد نمونه: **C#**

```
// Bind the System.Windows.Forms.DataGridView object
// to the System.Windows.Forms.BindingSource object.
dataGridView.DataSource = bindingSource;

// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);
```

```
DataTable orders = ds.Tables["SalesOrderHeader"];

// Query the SalesOrderHeader table for orders placed
// after August 8, 2001.
IEnumerable<DataRow> query =
    from order in orders.AsEnumerable()
    where order.Field<DateTime>("OrderDate") > new DateTime(2001, 8, 1)
    select order;

// Create a table from the query.
DataTable boundTable = query.CopyToDataTable<DataRow>();

// Bind the table to a System.Windows.Forms.BindingSource object,
// which acts as a proxy for a System.Windows.Forms.DataGridView object.
bindingSource.DataSource = boundTable;
```

متدهای جنریک Field و SetField

LINQ به DataSet متدهای بسط را برای کلاس DataRow به منظور دسترسی به مقادیر ستون در اختیار می‌گذارد: متد Field و متد SetField. این متدها برای توسعه دهندگان دسترسی راحت‌تری را به مقادیر ستون به خصوص درباره‌ی مقادیر تهی ارائه می‌دهند. شیء DataSet فیلد Value را برای ارائه‌ی مقادیر تهی به کار می‌برد در حالی که LINQ از نوع تهی‌پذیر معرفی شده در NET Framework 2.0. استفاده می‌کند. استفاده از اکسسور ستون از پیش موجود در DataRow نیازمند این است که شیء برگشتی را به نوع درخور قالب‌بندی کنید. اگر یک فیلد بخصوص در یک DataRow بتواند تهی شود، بایستی صریحاً برای یک مقدار تهی بررسی انجام دهید زیرا برگرداندن Value و قالب‌بندی ضمنی آن به نوع دیگر یک استثنای InvalidCastException پرتاب می‌کند. در مثال زیر، اگر متد IsNull برای بررسی یک مقدار تهی به کار برده نمی‌شد، در صورتی که تصریح کننده Value را برگشت می‌داد و سعی بر قالب‌بندی (تبدیل نوع صریح) آن به یک String می‌نمود یک استثنا پرتاب می‌شد.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable products = ds.Tables["Product"];

var query =
    from product in products.AsEnumerable()
    where !product.IsNull("Color") &&
        (string)product["Color"] == "Red"
    select new
           {
               Name = product["Name"],
               ProductNumber = product["ProductNumber"],
```

```
                ListPrice = product["ListPrice"]
            };

foreach (var product in query)
{
    Console.WriteLine("Name: {0}", product.Name);
    Console.WriteLine("Product number: {0}", product.ProductNumber);
    Console.WriteLine("List price: ${0}", product.ListPrice);
    Console.WriteLine("");
}
```

متد Field دسترسی لازم به مقادیر ستون یک DataRow را در اختیار می‌گذارد و متد SetField مقادیر ستون را در یک DataRow تنظیم می‌کند. هم متد Field و هم متد SetField هردو انواع تهی‌پذیر را مدیریت می‌کنند، ازاینرو لزومی ندارد که همانند مثال قبل صریحاً به منظور مقادیر تهی بررسی انجام دهید. هردو متد، متدهای جنریک هستند ازاینرو لزومی ندارد که نوع برگشتی را هم قالب‌بندی (تبدیل نوع صریح) کنید.

مثال زیر از متد Field استفاده می‌کند.

کد نمونه: **C#**

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable products = ds.Tables["Product"];

var query =
    from product in products.AsEnumerable()
    where product.Field<string>("Color") == "Red"
    select new
           {
               Name = product.Field<string>("Name"),
               ProductNumber = product.Field<string>("ProductNumber"),
               ListPrice = product.Field<Decimal>("ListPrice")
           };

foreach (var product in query)
{
    Console.WriteLine("Name: {0}", product.Name);
    Console.WriteLine("Product number: {0}", product.ProductNumber);
    Console.WriteLine("List price: ${0}", product.ListPrice);
    Console.WriteLine("");
}
```

توجه داشته باشید که نوع داده‌ی مشخص شده در پارامتر جنریک T متد Field و متد SetField باید با نوع مقدار متضمن جور باشد. در غیر این صورت، یک استثنای InvalidCastException پرتاب خواهد شد. اسم ستون مشخص شده نیز باید با اسم یک ستون واقع در DataSet مطابقت کند، وإلا یک استثنای ArgumentException

پرتاب خواهد شد. در هردو حالت، استثنا در زمان اجرا، در طی سرشماری داده، هنگام اجرا شدن پرس‌وجو پرتاب می‌شود.

متد SetField خودش هیچ گونه تبدیل داده‌ای را انجام نمی‌دهد. اگرچه این حرف به معنای این نیست که تبدیل نوعی رخ نخواهد داد. متد SetField رفتار ADO.NET 2.0 کلاس DataRow را ارائه می‌دهد. تبدیل نوع می‌تواند توسط شیء DataRow انجام شود و مقدار تبدیل شده پس از آن به شیء DataRow ذخیره خواهد شد.

انقیاد داده و LINQ به DataSet

انقیاد داده فرایندی است که یک ارتباط مابین UI (رابط کاربری) برنامه و منطق حرفه‌ای آن برقرار می‌کند. اگر انقیاد دارای تنظیمات درستی باشد و داده گزارشات درخور و مناسبی را ارائه دهد، هنگامی که داده مقدارش را تغییر دهد، عنصری که به داده مقید شده است به طور خودکار تغییرات را بازتاب خواهد داد. شیء DataSet یک نمایش درون حافظه‌ای از داده است که علیرغم منبع داده‌ای که دربردارد مدل برنامه‌نویسی رابطه‌ای پایداری را در اختیار می‌گذارد. DataView مربوط به ADO.NET 2.0 شما را قادر می‌سازد تا داده‌ی ذخیره شده در یک DataTable را مرتب کرده و فیلتر نمایید. این قابلیت اغلب اوقات در برنامه‌های انقیاد داده به کار برده می‌شود. با استفاده از یک DataView، شما می‌توانید داده‌ی واقع در یک جدول را با ترتیبات مرتب‌سازی مختلفی ارائه دهید و می‌توانید بوسیله‌ی وضعیت ردیف یا براساس یک عبارت فیلتر داده را فیلتر کنید.

LINQ به DataSet به توسعه دهنگان اجازه می‌دهد تا با استفاده از LINQ پرس‌وجوهای پیچیده و قدرتمندی را بر روی یک DataSet انجام دهند. اگرچه، یک پرس‌وجوی LINQ به DataSet فهرستی از اشیاء DataRow را برمی‌گرداند که به راحتی در یک سناریوی انقیاد به کار برده نمی‌شود. برای راحت‌تر ساختن انقیاد، می‌توانید یک DataView را از یک پرس‌وجوی LINQ به DataSet ایجاد کنید. این DataView از مرتب‌سازی و فیلترینگ مشخص شده در پرس‌وجو استفاده می‌کند اما برای انقیاد داده گزینه‌ی مناسب‌تری است. LINQ به DataSet با ارائه‌ی مرتب‌سازی و فیلترینگ مبتنی بر عبارت LINQ، که اجازه‌ی عملیات‌های مرتب‌سازی و فیلترینگ قدرتمند و پیچیده‌تری از مرتب‌سازی و فیلترینگ مبتنی بر رشته را می‌دهد، عاملیت DataView را بسط و گسترش می‌دهد.

توجه داشته باشید که DataView خودش پرس‌وجو را نمایش می‌دهد و یک دیدگاه بر فراز پرس‌وجو نیست. DataView با ارائه‌ی یک مدل انقیاد داده‌ی ساده به یک کنترل UI مانند یک DataGrid یا یک DataGridView مقید می‌شود. یک DataView با ارائه‌ی یک دیدگاه پیش فرض از آن جدول می‌تواند از یک DataTable نیز ایجاد شود.

ایجاد یک شیء DataView

در LINQ به DataSet، دو روش برای ایجاد یک DataView وجود دارد. شما می‌توانید یک DataView را از یک پرس‌وجوی LINQ به DataSet بر روی یک DataTable ایجاد کنید یا می‌توانید آن را از یک DataTable نوعدار یا بدون نوع ایجاد کنید. در هردو حالت، DataView را با استفاده از یکی از متدهای بسط AsDataView ایجاد خواهید کرد؛ DataView به طور مستقیم در بافت LINQ به DataSet قابل ساخت نیست.

بعد از این که DataView ایجاد شد، شما می‌توانید آن را به یک کنترل UI واقع در یک برنامه‌ی Windows Forms یا ASP.NET مقید کنید یا تنظیمات مرتب‌سازی و فیلترینگ را تغییر دهید.

DataView یک شاخص (index) ایجاد می‌کند که به شکل معناداری کارایی عملیات‌هایی چون مرتب‌سازی و فیلترینگ را که می‌توانند از شاخص استفاده کنند افزایش می‌دهد. شاخص مربوط به یک DataView هم زمانی که DataView ایجاد می‌شود و هم زمانی که هر کدام از اطلاعات مرتب‌سازی یا فیلترینگ تغییر کند ساخته می‌شود. ایجاد یک DataView و پس از آن تنظیم اطلاعات مرتب‌سازی و فیلترینگ در زمانهای آتی باعث می‌شود تا شاخص دستکم دو مرتبه ایجاد گردد: یک مرتبه زمانی که DataView ایجاد شود و دوباره زمانی که هر یک از عملیات‌های مرتب‌سازی یا فیلترینگ تغییر داده شوند.

ایجاد DataView از یک پرس‌وجوی LINQ به DataSet

یک شیء DataView می‌تواند از نتایج یک پرس‌وجوی LINQ به DataSet ایجاد شود جایی که نتایج تصویری از اشیاء DataRow هستند. شیء DataView که به تازگی ایجاد شده است اطلاعات مرتب‌سازی و فیلترینگ را از پرس‌وجویی که از آن ایجاد شده است به ارث می‌برد.

ایجاد یک شیء DataView از پرس‌وجویی که انواع بدون نام را برمی‌گرداند یا پرس‌وجوهایی که عملیات‌های اتصال (**join**) را انجام می‌دهند پشتیبانی نمی‌شود.

تنها عملگرهای پرس‌وجوی زیر در یک پرس‌وجوی به کار رفته برای ایجاد DataView پشتیبانی می‌شوند:

- Cast(TResult)
- OrderBy
- OrderByDescending
- Select``2(EnumerableRowCollection(UMP), Expression(Func(UMP, UMP)))
- ThenBy
- ThenByDescending
- Where(TRow)

مثال زیر یک شیء DataView را از سفارشات آنلاین مرتب شده به واسطه‌ی هزینه‌ی نهایی ایجاد می‌کند:

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderHeader"];

EnumerableRowCollection<DataRow> query =
    from order in orders.AsEnumerable()
    where order.Field<bool>("OnlineOrderFlag") == true
    orderby order.Field<decimal>("TotalDue")
    select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
```

شما می‌توانید بعد از این که یک شیء DataView از یک پرس‌وجو ایجاد شد خاصیت‌های مبتنی بر رشته‌ی RowFilter و Sort را هم برای فیلتر و مرتب‌سازی این شیء (DataView) استفاده کنید. توجه داشته باشید که این کار اطلاعات مرتب‌سازی و فیلترینگ ارث رسیده از پرس‌وجو را پاک خواهد کرد. مثال زیر از یک پرس‌وجوی LINQ به DataSet که با نام خانوادگی‌هایی که با 'S' شروع می‌شوند فیلتر می‌شود یک شیء DataView را ایجاد می‌کند. خاصیت مبتنی بر رشته‌ی Sort تنظیم می‌شود تا نام خانوادگی‌ها را به ترتیب صعودی مرتب کرده و پس از آن اسامی نخست را به ترتیب نزولی مرتب کند:

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

EnumerableRowCollection<DataRow> query = from contact in contacts.AsEnumerable()
```

```
                                        where
contact.Field<string>("LastName").StartsWith("S")
                                        select contact;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;

view.Sort = "LastName desc, FirstName asc";
```

ایجاد یک شیء DataView از یک شیء DataTable

علاوه بر ایجاد شدن از پرس‌وجوی LINQ به DataSet، یک شیء DataView می‌تواند با استفاده از متد AsDataView از یک شیء DataTable ایجاد گردد.

مثال زیر یک شیء DataView را از جدول SalesOrderDetail ایجاد کرده و آن را به عنوان منبع داده‌ی یک شیء BindingSource تنظیم می‌کند. این شیء به عنوان یک پراکسی برای یک کنترل DataGridView عمل می‌کند.

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderDetail"];

DataView view = orders.AsDataView();
bindingSource1.DataSource = view;

dataGridView1.AutoResizeColumns();
```

بعد از این که شیء DataView از یک شیء DataTable ایجاد شد فیلترینگ و مرتب‌سازی می‌توانند روی آن تنظیم شوند. مثال زیر یک شیء DataView را از جدول Contact ایجاد می‌کند و خاصیت Sort را تنظیم می‌کند تا نام خانوادگی‌ها را به ترتیب صعودی مرتب کرده و پس از آن اسامی نخست را به ترتیب نزولی مرتب کند:

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

DataView view = contacts.AsDataView();

view.Sort = "LastName desc, FirstName asc";

bindingSource1.DataSource = view;
```

```
dataGridView1.AutoResizeColumns();
```

اگرچه، در اینجا فقدان کارایی وجود دارد که با تنظیم خاصیت Sort یا RowFilter بعد از ایجاد شدن DataView از یک پرس‌وجو حاصل می‌شود زیرا شیء DataView شاخصی را برای پشتیبانی از عملیات‌های مرتب‌سازی و فیلترینگ ایجاد می‌کند. تنظیم خاصیت Sort یا RowFilter در حالی که سرباری را به برنامه‌ی شما تحمیل کرده و از میزان کارایی آن می‌کاهد شاخص مربوط به داده را بازسازی می‌کند. هرگاه امکان‌پذیر باشد، بهتر است که هنگامی که نخست DataView را ایجاد کرده و متعاقباً از ویرایش آن اجتناب می‌کنید اطلاعات مرتب‌سازی و فیلترینگ را مشخص کنید.

فیلترینگ با شیء DataView

توانایی فیلتر داده‌ها با استفاده از معیارهای بخصوص و پس از آن ارائه‌ی داده به یک مشتری از طریق یک کنترل UI جنبه‌ی مهمی از فرایند انقیاد داده است. شیء DataView برای فیلتر داده و برگرداندن زیرمجموعه‌ای از رکوردهای داده‌ای که معیارهای فیلتر بخصوصی را ارضا می‌کنند چندین راه متعدد را در اختیار می‌گذارد. علاوه بر قابلیت‌های فیلترینگ مبتنی بر رشته، DataView قابلیت استفاده از عبارات LINQ را برای معیارهای فیلترینگ ارائه می‌دهد. عبارات LINQ اجازه‌ی عملیات‌های فیلترینگ پیچیده‌تر و قدرتمندتری را از فیلترینگ مبتنی بر رشته می‌دهند.

دو روش برای فیلتر داده با استفاده از یک شیء DataView وجود دارد:

- ایجاد یک DataView از یک پرس‌وجوی LINQ به DataSet همراه با ضابطه‌ی **Where**.
- استفاده از قابلیت‌های فیلترینگ موجود مبتنی بر رشته‌ی DataView.

ایجاد شیء DataView از یک پرس‌وجو با اطلاعات فیلترینگ

یک شیء DataView می‌تواند از یک پرس‌وجوی LINQ به DataSet ایجاد شود. اگر آن پرس‌وجو شامل یک ضابطه‌ی **Where** باشد، DataView با اطلاعات فیلترینگ از پرس‌وجو ایجاد می‌شود. عبارت در ضابطه‌ی

Where به کار برده می‌شود تا تعیین کند که کدام رکوردهای داده ضمیمه‌ی DataView خواهد شد و زیربنای فیلتر است.

فیلترهای مبتنی بر عبارت فیلترینگ پیچیده‌تر و قدرتمندتری را از فیلترهای مبتنی بر رشته‌ی ساده‌تر در اختیار می‌گذارند. فیلترهای مبتنی بر رشته و فیلترهای مبتنی بر عبارت به طور متقابل مانعه‌الجمع هستند. هرگاه خاصیت مبتنی بر رشته‌ی RowFilter بعد از ایجاد شدن یک DataView از یک پرس‌وجو تنظیم شود، فیلتر مبتنی بر عبارت استنتاج شده از پرس‌وجو حذف می‌شود.

مثال

مثال زیر جدول SalesOrderDetail را برای یافتن سفارشات بزرگ‌تر از ۲ و کوچک‌تر از ۶ مورد پرس‌وجو قرار می‌دهد؛ یک شیء DataView را از آن پرس‌وجو ایجاد می‌کند؛ و DataView را به یک BindingSource مقید می‌کند:

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderDetail"];

EnumerableRowCollection<DataRow> query = from order in orders.AsEnumerable()
                                         where order.Field<Int16>("OrderQty") > 2
&& order.Field<Int16>("OrderQty") < 6
                                         select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
```

مثال

مثال زیر یک DataView را از یک پرس‌وجو برای یافتن سفارشات جای گرفته بعد از ۶ ژوئن سال ۲۰۰۱ ایجاد می‌کند:

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderHeader"];

EnumerableRowCollection<DataRow> query = from order in orders.AsEnumerable()
                                         where order.Field<DateTime>("OrderDate")
```

```
> new DateTime(2002, 6, 1)
                                        select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
```

مثال

عمل فیلترینگ می‌تواند با عمل مرتب‌سازی نیز ترکیب شود. مثال زیر یک شیء DataView را از یک پرس‌وجو برای یافتن تماس‌هایی که نام خانوادگی‌شان با "S" شروع شده و بواسطه‌ی نام خانوادگی و پس از آن نام نخست مرتب می‌شوند، ایجاد می‌کند:

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

EnumerableRowCollection<DataRow> query = from contact in contacts.AsEnumerable()
                                         where
contact.Field<string>("LastName").StartsWith("S")
                                         orderby
contact.Field<string>("LastName"), contact.Field<string>("FirstName")
                                         select contact;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
dataGridView1.AutoResizeColumns();
```

مثال

مثال زیر الگوریتم SoundEx را برای یافتن تماس‌هایی که نام خانوادگی‌شان مشابه "Zhu" است مورد استفاده قرار می‌دهد. الگوریتم SoundEx در متد SoundEx پیاده‌سازی می‌شود.

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

string soundExCode = SoundEx("Zhu");

EnumerableRowCollection<DataRow> query = from contact in contacts.AsEnumerable()
                                         where
SoundEx(contact.Field<string>("LastName")) == soundExCode
                                         select contact;
```

```
DataView view = query.AsDataView();

bindingSource1.DataSource = view;
dataGridView1.AutoResizeColumns();
```

SoundEx یک الگوریتم آواشناختی به کار رفته برای شاخص‌دار کردن اسامی به واسطه‌ی صدا به همان گونه که در انگلیسی تلفظ می‌شوند، است. متد SoundEx چهار کد کاراکتری را برای یک اسم برمی‌گرداند که از یک حرف انگلیسی که با ۳ عدد پی گرفته می‌شود تشکیل شده است. این حرف، حرف نخست اسم بوده و اعداد حروف بی‌صدای باقیمانده‌ی واقع در اسم را به رمز در می‌آورد. اسامی با صدای مشابه کد SoundEx مشابهی را به اشتراک می‌گذارند. پیاده‌سازی SoundEx به کار رفته در متد SoundEx مثال قبل در اینجا نشان داده شده است:

کد نمونه: **C#**

```
static private string SoundEx(string word)
{
    // The length of the returned code.
    int length = 4;

    // Value to return.
    string value = "";

    // The size of the word to process.
    int size = word.Length;

    // The word must be at least two characters in length.
    if (size > 1)
    {
        // Convert the word to uppercase characters.
        word = word.ToUpper(System.Globalization.CultureInfo.InvariantCulture);

        // Convert the word to a character array.
        char[] chars = word.ToCharArray();

        // Buffer to hold the character codes.
        StringBuilder buffer = new StringBuilder();
        buffer.Length = 0;

        // The current and previous character codes.
        int prevCode = 0;
        int currCode = 0;

        // Add the first character to the buffer.
        buffer.Append(chars[0]);

  // Loop through all the characters and convert them to the proper character code.
        for (int i = 1; i < size; i++)
        {
```

```
            switch (chars[i])
            {
                case 'A':
                case 'E':
                case 'I':
                case 'O':
                case 'U':
                case 'H':
                case 'W':
                case 'Y':
                    currCode = 0;
                    break;
                case 'B':
                case 'F':
                case 'P':
                case 'V':
                    currCode = 1;
                    break;
                case 'C':
                case 'G':
                case 'J':
                case 'K':
                case 'Q':
                case 'S':
                case 'X':
                case 'Z':
                    currCode = 2;
                    break;
                case 'D':
                case 'T':
                    currCode = 3;
                    break;
                case 'L':
                    currCode = 4;
                    break;
                case 'M':
                case 'N':
                    currCode = 5;
                    break;
                case 'R':
                    currCode = 6;
                    break;
            }

    // Check if the current code is the same as the previous code.
            if (currCode != prevCode)
            {
    // Check to see if the current code is 0 (a vowel); do not process vowels.
                if (currCode != 0)
                    buffer.Append(currCode);
            }
            // Set the previous character code.
            prevCode = currCode;

            // If the buffer size meets the length limit, exit the loop.
            if (buffer.Length == length)
                break;
        }
        // Pad the buffer, if required.
```

```
        size = buffer.Length;
        if (size < length)
            buffer.Append('0', (length - size));

        // Set the value to return.
        value = buffer.ToString();
    }
    // Return the value.
    return value;
}
```

استفاده از خاصیت RowFilter

قابلیت موجود فیلترینگ مبتنی بر رشته‌ی DataView هنوز هم در بافت LINQ به DataSet کار می‌کند.

مثال زیر یک شیء DataView را از جدول Contact ایجاد کرده و پس از آن خاصیت RowFilter را تنظیم می‌کند تا رکوردهایی را برگرداند که نام خانوادگی تماس "Zhu" باشد:

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

DataView view = contacts.AsDataView();

view.RowFilter = "LastName='Zhu'";

bindingSource1.DataSource = view;
dataGridView1.AutoResizeColumns();
```

بعد از این که یک DataView از یک DataTable و یا از یک پرس‌وجوی LINQ به DataSet ایجاد شد، شما می‌توانید خاصیت RowFilter را به کار ببرید تا زیرمجموعه‌هایی از رکوردها را بر اساس مقادیر ستون‌هایشان مشخص کنید. فیلترهای مبتنی بر رشته و فیلترهای مبتنی بر عبارت به طور متقابل مانعه‌الجمع هستند. تنظیم خاصیت RowFilter عبارت فیلتر استنتاج شده از پرس‌وجوی LINQ به DataSet را پاک خواهد کرد و عبارت فیلتر نمی‌تواند بازنشانی شود.

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

EnumerableRowCollection<DataRow> query = from contact in contacts.AsEnumerable()
                                         where contact.Field<string>("LastName")
```

```
== "Hernandez"
                                        select contact;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
dataGridView1.AutoResizeColumns();

view.RowFilter = "LastName='Zhu'";
```

اگر قصد دارید تا نتایج یک پرس‌وجوی بخصوص بر روی داده را برگردانید، در مقابل اجتناب از نمای پویایی از یک زیرمجموعه‌ای از داده، شما می‌توانید غیر از تنظیم خاصیت RowFilter از متدهای Find یا FindRows شیء DataView استفاده کنید. بهترین کاربرد خاصیت RowFilter در یک برنامه مقید به داده است جایی که در آن یک کنترل مقید نتایج فیلتر شده را نمایش می‌دهد. تنظیم خاصیت RowFilter درحالی که سرباری را به برنامه تحمیل کرده و از میزان کارایی آن می‌کاهد شاخص مربوط به داده را بازسازی می‌کند. متدهای Find و FindRows بدون این که نیاز به بازسازی شاخص باشد از شاخص فعلی استفاده می‌کنند. اگر تنها یک مرتبه قصد فراخوانی Find یا FindRows را دارید، در این صورت باید از DataView موجود استفاده کنید. اگر قصد داشته باشید تا متدهای Find یا FindRows را چندین بار فراخوانی کنید، باید یک DataView جدید را به منظور بازسازی شاخص بر روی ستونی که قصد جستجو بر روی آن را دارید ایجاد کنید و پس از آن متدهای Find یا FindRows را فراخوان کنید.

پاک کردن فیلتر

بعد از این که فیلترینگ با استفاده از خاصیت RowFilter تنظیم شد فیلتر بر روی یک شیء DataView می‌تواند پاک شود. فیلتر بر روی یک DataView می‌تواند به دو طریق مختلف پاک شود:

- تنظیم خاصیت RowFilter به **null**.
- تنظیم خاصیت RowFilter به یک رشته‌ی تهی.

مثال

مثال زیر یک شیء DataView را از یک پرس‌وجو ایجاد کرده و پس از آن با تنظیم خاصیت RowFilter به **null**، فیلتر را پاک می‌کند:

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderHeader"];

EnumerableRowCollection<DataRow> query = from order in orders.AsEnumerable()
                                         where order.Field<DateTime>("OrderDate")
> new DateTime(2002, 11, 20)
                                            && order.Field<Decimal>("TotalDue") <
new Decimal(60.00)
                                         select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;

view.RowFilter = null;
```

مثال

مثال زیر یک شیء DataView را از یک جدول ایجاد کرده و خاصیت RowFilter را تنظیم نموده و پس از آن با تنظیم خاصیت RowFilter به یک رشته‌ی تهی، فیلتر را پاک می‌کند:

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

DataView view = contacts.AsDataView();

view.RowFilter = "LastName='Zhu'";


bindingSource1.DataSource = view;
dataGridView1.AutoResizeColumns();

// Clear the row filter.
view.RowFilter = "";
```

مرتب کردن با DataView

توانایی مرتب‌سازی داده بر اساس معیارهای بخصوص و پس از آن ارائه‌ی داده به یک مشتری از طریق یک کنترل UI یکی از جنبه‌های مهم انقیاد داده است. DataView برای مرتب‌سازی داده و برگرداندن رکوردهای داده‌ی مرتب شده به واسطه‌ی معیارهای مرتب‌سازی بخصوص چندین شیوه را در اختیار می‌گذارد. علاوه بر قابلیت‌های مرتب‌سازی مبتنی بر رشته‌اش، DataView شما را قادر می‌سازد تا عبارات LINQ را هم برای معیارهای مرتب‌سازی به کار ببرید. عبارات LINQ اجازه‌ی عملیات‌های مرتب‌سازی قدرتمندتر و پیچیده‌تری از عملیات‌های مرتب‌سازی مبتنی بر رشته را می‌دهند. این بخش هردو رویکرد را برای مرتب‌سازی با استفاده DataView تشریح می‌کند.

ایجاد DataView از یک پرس‌وجو با اطلاعات مرتب‌سازی

یک شیء DataView می‌تواند از یک پرس‌وجوی LINQ به DataSet ایجاد شود. اگر آن پرس‌وجو شامل یکی از ضابطه‌های OrderBy، OrderByDescending، ThenBy یا ThenByDescending باشد، عبارات واقع در این ضابطه‌ها به عنوان مبنای مرتب‌سازی داده در DataView به کار برده می‌شوند. برای مثال، اگر پرس‌وجو شامل ضابطه‌های **OrderBy...** و **Then By...** باشد، DataView حاصل داده را بواسطه‌ی دو ستون مشخص شده مرتب خواهد کرد.

مرتب‌سازی مبتنی بر عبارت مرتب‌سازی پیچده‌تر و قدرتمندتری را از مرتب‌سازی مبتنی بر رشته‌ی ساده‌تر در اختیار می‌گذارد. توجه داشته باشید که مرتب‌سازی‌های مبتنی بر رشته و مبتنی بر عبارت متقابلاً مانعه‌الجمع هستند. اگر خاصیت مبتنی بر رشته‌ی Sort بعد از ایجاد شدن یک DataView از یک پرس‌وجو تنظیم شود، فیلتر مبتنی بر عبارت استنتاج شده از پرس‌وجو پاک می‌شود و نمی‌تواند بازنشانی شود.

شاخص مربوط به یک DataView هم زمان ایجاد DataView و هم زمانی که یکی از اطلاعات مرتب‌سازی یا فیلترینگ تغییر داده می‌شود ایجاد می‌شود. با فراهم‌سازی معیارهای مرتب‌سازی در پرس‌وجوی LINQ به

DataSet که DataView از آن ایجاد می‌شود و تغییر ندادن طلاعات مرتب‌سازی در زمان آتی می‌توانید بهترین کارایی را بدست آورید.

مثال

مثال زیر از جدول SalesOrderHeader پرس‌وجو کرده و رکوردهای برگشتی را به واسطه‌ی تاریخ مرتب می‌کند؛ یک شیء DataView از آن پرس‌وجو ایجاد می‌کند و پس از آن DataView را به یک BindingSource مقید می‌کند.

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderHeader"];

EnumerableRowCollection<DataRow> query = from order in orders.AsEnumerable()
                                         orderby
order.Field<DateTime>("OrderDate")
                                         select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
```

مثال

مثال زیر از جدول SalesOrderHeader پرس‌وجو کرده و رکوردهای برگشتی را به واسطه‌ی مقدار هزینه مرتب می‌کند؛ یک شیء DataView از آن پرس‌وجو ایجاد می‌کند و پس از آن DataView را به یک BindingSource مقید می‌کند.

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderHeader"];

EnumerableRowCollection<DataRow> query =
    from order in orders.AsEnumerable()
    orderby order.Field<decimal>("TotalDue")
    select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
```

مثال

مثال زیر از جدول SalesOrderDetail پرس‌وجو کرده و رکوردهای برگشتی را به واسطه‌ی میزان سفارش و بعد از آن به واسطه‌ی ID سفارش فروش مرتب می‌کند؛ یک شیء DataView از آن پرس‌وجو ایجاد می‌کند و پس از آن DataView را به یک BindingSource مقید می‌کند.

کد نمونه: **C#**

```
DataTable orders = dataSet.Tables["SalesOrderDetail"];

EnumerableRowCollection<DataRow> query = from order in orders.AsEnumerable()
                                         orderby order.Field<Int16>("OrderQty"),
order.Field<int>("SalesOrderID")
                                         select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;
```

استفاده از خاصیت مبتنی بر رشته‌ی Sort

قابلیت مرتب‌سازی مبتنی بر رشته‌ی DataView هنوز هم با LINQ به DataSet کار می‌کند. بعد از این که یک شیء DataView از یک پرس‌وجوی LINQ به DataSet ایجاد شد، شما می‌توانید خاصیت مبتنی بر رشته‌ی Sort را برای مرتب‌سازی بر روی DataView به کار ببرید.

قابلیت‌های مرتب‌سازی مبتنی بر رشته و مبتنی بر عبارت با یکدیگر جمع‌پذیر نیستند. تنظیم خاصیت Sort مرتب‌سازی مبتنی بر عبارت به ارث رسیده از پرس‌وجویی که DataView از آن ایجاد شده بود را پاک خواهد کرد.

مثال

مثال زیر یک شیء DataView از جدول Contact ایجاد کرده و رکوردها را به واسطه‌ی نام خانودگی‌شان به ترتیب نزولی و پس از آن به واسطه‌ی نام نخست‌شان به ترتیب صعودی مرتب می‌کند:

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

DataView view = contacts.AsDataView();

view.Sort = "LastName desc, FirstName asc";

bindingSource1.DataSource = view;
dataGridView1.AutoResizeColumns();
```

مثال

مثال زیر جدول Contact را برای یافتن نامهای خانوادگی که با حرف "S" شروع می‌شوند مورد پرس‌وجو قرار می‌دهد. یک DataView از آن پرس‌وجو ایجاد می‌شود و به یک شیء BindingSource مقید می‌شود.

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

EnumerableRowCollection<DataRow> query = from contact in contacts.AsEnumerable()
                                         where
contact.Field<string>("LastName").StartsWith("S")
                                         select contact;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;

view.Sort = "LastName desc, FirstName asc";
```

تسویه‌ی خاصیت مبتنی بر رشته‌ی Sort

اطلاعات مرتب‌سازی بر روی یک DataView می‌تواند بعد از تنظیم شدنش با استفاده از خاصیت Sort پاک شود. دو روش برای پاک کردن اطلاعات مرتب‌سازی در DataView وجود دارد:

- تنظیم خاصیت Sort به **null**.
- تنظیم خاصیت Sort به یک رشته‌ی تهی.

مثال

مثال زیر یک شیء DataView از یک پرس‌وجو ایجاد کرده و اطلاعات مرتب‌سازی را با تنظیم خاصیت Sort به یک رشته‌ی تهی تسویه می‌کند:

کد نمونه: C#

```
DataTable orders = dataSet.Tables["SalesOrderHeader"];

EnumerableRowCollection<DataRow> query = from order in orders.AsEnumerable()
                                         orderby order.Field<decimal>("TotalDue")
                                         select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;

view.Sort = "";
```

مثال

مثال زیر یک شیء DataView از جدول Contact ایجاد کرده و خاصیت Sort را تنظیم می‌کند تا رکوردها را به واسطه‌ی نام خانوادگی‌شان به ترتیب نزولی مرتب کند. پس از آن اطلاعات مرتب‌سازی با تنظیم خاصیت Sort به **null** پاک می‌شوند:

کد نمونه: C#

```
DataTable contacts = dataSet.Tables["Contact"];

DataView view = contacts.AsDataView();

view.Sort = "LastName desc";

bindingSource1.DataSource = view;
dataGridView1.AutoResizeColumns();

// Clear the sort.
view.Sort = null;
```

کارایی کلاس DataView

این بخش درباره‌ی مزایای عملکردی استفاده از متدهای Find و FindRows کلاس DataView و نهان‌سازی یک DataView در یک برنامه‌ی Web بحث می‌کند.

متدهای Find و FindRows

DataView یک شاخص را ایجاد می‌کند. یک شاخص شامل کلیدهایی است که از یک یا چند ستون واقع در جدول یا دیدگاه ساخته می‌شوند. این کلیدها در ساختاری ذخیره می‌شوند که DataView را قادر می‌سازد تا رکورد (ردیف) یا رکوردهای مرتبط با مقادیر کلید را به سرعت و با کارایی بالا پیدا کند. عملیات‌هایی چون فیلترینگ و مرتب‌سازی که از شاخص استفاده می‌کنند افزایش کارایی قابل توجهی را دریافت می‌کنند. شاخص مربوط به یک DataView هم زمان ایجاد DataView و هم زمانی که هرکدام از اطلاعات مرتب‌سازی یا فیلترینگ تغییر داده می‌شوند ایجاد می‌شود. ایجاد یک DataView و پس از آن تنظیم اطلاعات مرتب‌سازی یا فیلترینگ باعث می‌شود تا شاخص دست‌کم دو مرتبه ایجاد شود: یک مرتبه زمانی که DataView ایجاد می‌شود و دوباره زمانی که هر یک از خاصیت‌های مرتب‌سازی یا فیلتر تغییر داده می‌شوند.

اگر در مقابل ارائه‌ی یک دیدگاه دینامیک از یک زیرمجموعه از داده، قصد برگرداندن نتایج یک پرس‌وجوی بخصوص بر روی داده را دارید، می‌توانید غیر از تنظیم خاصیت RowFilter از متدهای Find یا FindRows استفاده کنید. خاصیت RowFilter بهترین کاربرد را در یک برنامه‌ی مقید به داده دارد جایی که در آن یک کنترل مقید نتایج فیلتر شده را نمایش می‌دهد. تنظیم خاصیت RowFilter با تحمیل سربار به برنامه‌ی شما و کاستن از میزان کارایی شاخص (ایندکس) مربوط به داده را بازسازی می‌کند. متدهای Find و FindRows بدون نیاز به بازسازی شاخص (ایندکس) از شاخص (ایندکس) فعلی استفاده می‌کنند. اگر قصد دارید که متدهای Find یا FindRows را تنها یک مرتبه فراخوان کنید، در این صورت باید از DataView موجود استفاده کنید. اگر قصد فراخوانی چندباره‌ی متدهای Find یا FindRows را دارید، باید یک DataView جدید ایجاد کنید تا شاخص (ایندکس) را بر روی ستونی که قصد جستجو بر روی آن را دارید بازسای کنید و سپس متدهای Find یا FindRows را فراخوان کنید.

مثال زیر متد Find را برای یافتن تماس‌هایی با نام خانوادگی "Zhu" به کار می‌برد.

کد نمونه: **C#**

```
DataTable contacts = dataSet.Tables["Contact"];

EnumerableRowCollection<DataRow> query = from contact in contacts.AsEnumerable()
```

```
                                        orderby contact.Field<string>("LastName")
                                        select contact;

DataView view = query.AsDataView();

// Find a contact with the last name of Zhu.
int found = view.Find("Zhu");
```

مثال زیر متد FindRows را برای یافتن تمامی محصولات با رنگ قرمز به کار می‌برد.

کد نمونه: **C#**

```
DataTable products = dataSet.Tables["Product"];

EnumerableRowCollection<DataRow> query = from product in products.AsEnumerable()
                                        orderby
product.Field<Decimal>("ListPrice"), product.Field<string>("Color")
                                        select product;

DataView view = query.AsDataView();

view.Sort = "Color";

object[] criteria = new object[] { "Red"};

DataRowView[] foundRowsView = view.FindRows(criteria);
```

ASP.NET

ASP.NET دارای یک مکانیزم نهان‌سازی (caching) است که به شما اجازه می‌دهد تا اشیائی را که برای ایجاد درون حافظه‌ای نیاز به منابع سرور گسترده دارند ذخیره کنید. نهان‌سازی این گونه از منابع به شکل معناداری می‌تواند عملکرد برنامه‌ی شما را بهبود بخشد. نهان‌سازی توسط کلاس Cache پیاده‌سازی می‌شود همراه با نمونه‌های نهان‌گاهی که برای هر یک از برنامه‌ها تدارک دیده شده است. از آن جایی که ایجاد یک شیء DataView جدید می‌تواند منبعبَر باشد، ممکن است بخواهید تا از این قابلیت نهان‌سازی در برنامه‌های Web خود استفاده کنید طوری که دیگر لزومی نداشته باشد که هربار که صفحه‌ی Web تازه‌سازی می‌شود DataView بازسازی شود.

در مثال زیر، DataView نهان می‌شود طوری که داده لزومی ندارد که داده هنگام بازسازی صفحه ازنومرتب شود.

کد نمونه: **C#**

```
if (Cache["ordersView"] == null)
{
   // Fill the DataSet.
   DataSet dataSet = FillDataSet();

   DataTable orders = dataSet.Tables["SalesOrderHeader"];

   EnumerableRowCollection<DataRow> query =
                        from order in orders.AsEnumerable()
                        where order.Field<bool>("OnlineOrderFlag") == true
                        orderby order.Field<decimal>("TotalDue")
                        select order;

   DataView view = query.AsDataView();
   Cache.Insert("ordersView", view);
}

DataView ordersView = (DataView)Cache["ordersView"];

GridView1.DataSource = ordersView;
GridView1.DataBind();
```

چگونگی مقید کردن یک شیء DataView به یک کنترل DataGridView

کنترل DataGridView روش قدرتمند و انعطاف‌پذیری را برای نمایش داده در یک قالب جدولی ارائه می‌دهد. کنترل DataGridView از مدل انقیاد داده‌ی استاندارد Windows Forms پشتیبانی می‌کند ازاینرو به DataView و گستره‌ی وسیعی از منابع داده مقید خواهد شد. اگرچه در اغلب وضعیت‌ها، شما به یک مؤلفه‌ی BindingSource مقید خواهید کرد که جزییات تعامل با منبع داده را مدیریت خواهد کرد.

برای متصل کردن یک کنترل DataGridView به یک DataView

۱. متدی را برای مدیریت جزییات بازیابی داده از یک پایگاه داده پیاده‌سازی کنید. کد نمونه‌ی زیر یک متد **GetData** را پیاده‌سازی می‌کند که این متد یک مؤلفه‌ی SqlDataAdapter را مقداردهی اولیه کرده و آن را برای پر کردن یک شیء DataSet مورد استفاده قرار می‌دهد. اطمینان حاصل کنید که متغیر *connectionString* به مقداری تنظیم می‌شود که برای پایگاه داده‌ی شما مناسب است.

کد نمونه: **C#**

```
private void GetData()
{
    try
    {
        // Initialize the DataSet.
        dataSet = new DataSet();
```

```
        dataSet.Locale = CultureInfo.InvariantCulture;

   // Create the connection string for the AdventureWorks sample database.
        string connectionString = "Data Source=localhost;Initial
Catalog=AdventureWorks;"
            + "Integrated Security=true;";

        // Create the command strings for querying the Contact table.
        string contactSelectCommand = "SELECT ContactID, Title, FirstName,
LastName, EmailAddress, Phone FROM Person.Contact";

        // Create the contacts data adapter.
        contactsDataAdapter = new SqlDataAdapter(
            contactSelectCommand,
            connectionString);

   // Create a command builder to generate SQL update, insert, and
   // delete commands based on the contacts select command. These are used to
        // update the database.
        SqlCommandBuilder contactsCommandBuilder = new
SqlCommandBuilder(contactsDataAdapter);

        // Fill the data set with the contact information.
        contactsDataAdapter.Fill(dataSet, "Contact");

    }
    catch (SqlException ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

۲. در گرداننده‌ی رویداد Load فرمتان، کنترل DataGridView را به مؤلفه‌ی BindingSource مقید کنید و متد **GetData** را برای بازیابی داده از پایگاه داده فراخوان کنید. DataView از پرس‌وجوی LINQ به DataSet روی جدول (شیء DataTable) Contact ایجاد می‌شود و پس از آن به مؤلفه‌ی BindingSource مقید می‌شود.

C# کد نمونه:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Connect to the database and fill the DataSet.
    GetData();

    contactDataGridView.DataSource = contactBindingSource;

    // Create a LinqDataView from a LINQ to DataSet query and bind it
    // to the Windows forms control.
    EnumerableRowCollection<DataRow> contactQuery = from row in
dataSet.Tables["Contact"].AsEnumerable()
                                                    where
```

```
row.Field<string>("EmailAddress") != null
                                                    orderby
row.Field<string>("LastName")
                                                    select row;

    contactView = contactQuery.AsDataView();

    // Bind the DataGridView to the BindingSource.
    contactBindingSource.DataSource = contactView;
    contactDataGridView.AutoResizeColumns();
}
```

LINQ به اشیاء

عبارت «LINQ به اشیاء» به کاربرد مستقیم پرس‌وجوهای LINQ با هر گونه IEnumerable یا IEnumerable(T) بدون استفاده از یک مهیا کننده‌ی LINQ میانی یا APIای نظیر LINQ به SQL یا LINQ به XML اشاره دارد. شما می‌توانید LINQ را برای پرس وجو از هر گونه مجموعه قابل شمارشی نظیر List(T)، Array یا Dictionary(TKey, TValue) به کار ببرید. مجموعه می‌تواند تعریف شده توسط کاربر باشد و یا توسط یک API مربوط به .NET Framework برگشت داده شده باشد.

در برداشت نخست، LINQ به اشیاء بیانگر یک رویکرد جدید به مجموعه‌هاست. در روش قدیمی، شما بایستی حلقه‌های **foreach** پیچیده‌ای می‌نوشتید که چگونگی بازیابی داده از یک مجموعه را مشخص می‌کردند. در رویکرد LINQ، شما کد اعلانی را می‌نویسید که توصیف کننده‌ی آن چیزی است که می‌خواهید بازیابی‌اش کنید.

علاوه بر این، پرس‌وجوهای LINQ نسبت به حلقه‌های **foreach** سنتی، سه مزیت عمده دارند:

۳. آنها موجز و مختصر و خواناترند به ویژه هنگام فیلترینگ (پالایش) شروط متعدد.

۴. آنها قابلیت‌های فیلترینگ (پالایش)، مرتب‌سازی و گروه‌بندی قدرتمندی را همراه با کمترین کدنویسی در اختیار می‌گذارند.

۵. آنها می‌توانند با کمترین یا هیچ تغییری به منابع داده‌ی دیگر حمل شوند.

به طور کلی، استفاده از LINQ به جای تکنیک‌های بازیابی سنتی اطلاعات مزایا و منافع بیشتری را در اختیارتان می‌گذارد.

چگونگی پرس وجو از یک ArrayList با LINQ

هنگام استفاده از LINQ برای پرس‌وجو از مجموعه‌های غیرجنریکی چون ArrayList، بایستی نوع متغیر دامنه را به طور صریح اعلان کنید تا نوع مشخصی از اشیاء را در مجموعه بازتاب دهد. برای مثال، اگر یک ArrayList از اشیاء `Student` دارید، ضابطه‌ی **from** (در C#) یا **From** (در ویژوال بیسیک) باید به این شکل باشد:

کد نمونه:

```
// C#
var query = from Student s in arrList
...
```

با تعیین نوع متغیر دامنه، شما هر یک از اقلام واقع در ArrayList را به یک `Student` قالب‌بندی می‌کنید. استفاده از یک متغیر دامنه‌ای که به شکل صریح نوعدار شده است معادل با فراخوانی متد Cast(TResult) است. اگر قالب‌بندی مشخص شده نتواند انجام گیرد متد Cast(TResult) یک استثنا پرتاب خواهد کرد. Cast(TResult) و OfType(TResult) دو متد عملگر پرس وجوی استاندارد هستند که روی انواع IEnumerable غیرجنریک عمل می‌کنند.

مثال

مثال زیر یک پرس‌وجوی ساده را بر روی یک ArrayList نشان می‌دهد. توجه داشته باشید که این مثال هنگامی که کد متد Add را فراخوانی می‌کند از مقداراولیه دهنده‌های شیء استفاده می‌کند، اما این امر یک نیازمندی ضروری نیست.

کد نمونه: **C#**

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

namespace NonGenericLINQ
{
    public class Student
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int[] Scores { get; set; }
    }

    class Program
```

```
    {
        static void Main(string[] args)
        {
            ArrayList arrList = new ArrayList();
            arrList.Add(
                new Student
                    {
                        FirstName = "Svetlana", LastName = "Omelchenko", Scores =
new int[] { 98, 92, 81, 60 }
                    });
            arrList.Add(
                new Student
                    {
                        FirstName = "Claire", LastName = "O'Donnell", Scores = new
int[] { 75, 84, 91, 39 }
                    });
            arrList.Add(
                new Student
                    {
                        FirstName = "Sven", LastName = "Mortensen", Scores = new
int[] { 88, 94, 65, 91 }
                    });
            arrList.Add(
                new Student
                    {
                        FirstName = "Cesar", LastName = "Garcia", Scores = new
int[] { 97, 89, 85, 82 }
                    });

            var query = from Student student in arrList
                        where student.Scores[0] > 95
                        select student;

            foreach (Student s in query)
                Console.WriteLine(s.LastName + ": " + s.Scores[0]);

            // Keep the console window open in debug mode.
            Console.WriteLine("Press any key to exit.");
            Console.ReadKey();
        }
    }
}
/* Output:
    Omelchenko: 98
    Garcia: 97
*/
```

LINQ و رشته ها

LINQ می‌تواند برای پرس وجو و انتقال رشته‌ها و مجموعه‌ای از رشته‌ها مورد استفاده قرار گیرد. به ویژه LINQ می‌تواند با داده‌ی نیمه ساخت یافته در فایل‌های داده سودمند باشد. پرس‌وجوهای LINQ می‌توانند با توابع رشته‌ای سنتی و عبارات منظم ترکیب شوند. برای مثال، شما می‌توانید متد Split را به کار ببرید تا آرایه‌ای از

رشته‌ها را ایجاد کنید که بعد از آن می‌توانید آن را با استفاده از LINQ مورد پرس وجو قرار داده یا تغییرش دهید. شما می‌توانید متد IsMatch را در ضابطه‌ی **where** یک پرس وجوی LINQ مورد استفاده قرار دهید. و می‌توانید LINQ را برای پرس‌وجو یا تغییر دادن نتایج MatchCollection برگشت داده شده توسط یک عبارت منظم به کار ببرید.

شما می‌توانید بلوک‌های متنی را با تقسیم آنها با استفاده از متد Split در یک آرایه‌ی قابل پرس وجو از رشته‌های کوچک‌تر، مورد پرس وجو قرار داده، آنالیز کرده و تغییرشان دهید. شما می‌توانید متن منبع را به واژه‌ها، جملات، پاراگراف‌ها، صفحات و یا هر معیار دیگری تقسیم کرده و هرگونه تقسیم اضافی را در صورت نیاز در پرس‌وجوی خود صورت دهید.

انواع مختلف فایل‌های متنی از یک سری از سطور تشکیل می‌شوند که اغلب اوقات با قالب‌بندی مشابهی چون فایل‌های مجزا شده توسط کاما یا tab همراهند و یا از سطور با طول ثابت تشکیل می‌شوند. بعد از این که یک چنین فایلی را به میان حافظه خواندید، شما می‌توانید LINQ را برای پرس‌وجو و/یا تغییر سطور به کار ببرید. ضمناً پرس‌وجوهای LINQ عمل ترکیب داده از منابع متعدد را ساده‌تر می‌کند.

چگونگی شمارش تعداد رخدادهای یک کلمه در یک رشته

این مثال چگونگی استفاده از LINQ برای شمارش تعداد رخدادهای یک کلمه مشخص در یک رشته را نشان می‌دهد. توجه داشته باشید که برای انجام شمارش، نخست متد Split فراخوانده می‌شود تا آرایه‌ای از کلمات را ایجاد کند. استفاده از متد Split هزینه‌ی عملکردی را در پی دارد. اگر تنها عملی که بر روی رشته انجام می‌شود شمارش کلمات در آن است، بهتر است که به جای Split به فکر استفاده از متدهای Matches یا IndexOf باشید. اگرچه، میزان کارایی مسئله‌ی بحرانی نباشد یا شما قبلاً جمله را تقسیم کرده‌اید تا انواع دیگری از پرس‌وجو را بر روی آن انجام دهید در این صورت به نظر می‌رسد بهتر باشد که برای شمارش کلمات یا عبارات از LINQ استفاده کنید.

مثال

کد نمونه: **C#**

```
class CountWords
{
    static void Main()
    {
        string text = @"Historically, the world of data and the world of objects"
+
         @" have not been well integrated. Programmers work in C# or Visual Basic"
+
         @" and also in SQL or XQuery. On the one side are concepts such as
classes," +
         @" objects, fields, inheritance, and .NET Framework APIs. On the other
side" +
         @" are tables, columns, rows, nodes, and separate languages for dealing
with" +
         @" them. Data types often require translation between the two worlds;
there are" +
         @" different standard functions. Because the object world has no notion
of query, a" +
          @" query can only be represented as a string without compile-time type
checking or" +
          @" IntelliSense support in the IDE. Transferring data from SQL tables or
XML trees to" +
          @" objects in memory is often tedious and error-prone.";

        string searchTerm = "data";

        //Convert the string into an array of words
        string[] source = text.Split(new char[] { '.', '?', '!', ' ', ';', ':',
',' }, StringSplitOptions.RemoveEmptyEntries);

        // Create and execute the query. It executes immediately
        // because a singleton value is produced.
        // Use ToLowerInvariant to match "data" and "Data"
        var matchQuery = from word in source
                         where word.ToLowerInvariant() ==
searchTerm.ToLowerInvariant()
                         select word;

        // Count the matches.
        int wordCount = matchQuery.Count();
        Console.WriteLine("{0} occurrences(s) of the search term \"{1}\" were
found.", wordCount, searchTerm);

        // Keep console window open in debug mode
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
/* Output:
   3 occurrences(s) of the search term "data" were found.
*/
```

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework. نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی پرس‌وجو برای جملاتی که حاوی مجموعه‌ی معینی از کلمات هستند

این مثال چگونگی پیدا کردن جملات واقع در یک فایل متنی را که حاوی تطابقاتی برای هر مجموعه‌ی مشخصی از کلمات است نشان می‌دهد. با وجود این که آرایه‌ی عبارات جستجو در این مثال از نقطه نظر کدنویسی طاقت فرساست، در زمان اجرا نیز می‌تواند به صورت پویا پر شود. در این مثال، پرس‌وجو جملاتی را که حاوی کلمات Historically، data و integrated هستند برمی‌گرداند.

مثال

کد نمونه: **C#**

```
class FindSentences
{
    static void Main()
    {
        string text = @"Historically, the world of data and the world of objects "
+
        @"have not been well integrated. Programmers work in C# or Visual Basic "
+
        @"and also in SQL or XQuery. On the one side are concepts such as classes,
" +
        @"objects, fields, inheritance, and .NET Framework APIs. On the other side
" +
     @"are tables, columns, rows, nodes, and separate languages for dealing with "
+
```

```
     @"them. Data types often require translation between the two worlds; there
are " +
     @"different standard functions. Because the object world has no notion of
query, a " +
     @"query can only be represented as a string without compile-time type
checking or " +
    @"IntelliSense support in the IDE. Transferring data from SQL tables or XML
trees to " +
    @"objects in memory is often tedious and error-prone.";

        // Split the text block into an array of sentences.
        string[] sentences = text.Split(new char[] { '.', '?', '!' });

        // Define the search terms. This list could also be dynamically populated
at runtime.
        string[] wordsToMatch = { "Historically", "data", "integrated" };

        // Find sentences that contain all the terms in the wordsToMatch array.
        // Note that the number of terms to match is not specified at compile
time.
        var sentenceQuery = from sentence in sentences
                            let w = sentence.Split(new char[] { '.', '?', '!', '
', ';', ':', ',' },

StringSplitOptions.RemoveEmptyEntries)
                            where w.Distinct().Intersect(wordsToMatch).Count() ==
wordsToMatch.Count()
                            select sentence;

        // Execute the query. Note that you can explicitly type
        // the iteration variable here even though sentenceQuery
        // was implicitly typed.
        foreach (string str in sentenceQuery)
        {
            Console.WriteLine(str);
        }

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
/* Output:
Historically, the world of data and the world of objects have not been well
integrated
*/
```

پرس‌وجو در ابتدا با تقسیم متن به جملات و پس از آن تقسیم جملات به آرایه‌ای از رشته‌هایی که کلمات را نگه می‌دارند عمل می‌کند. برای هر یک از این آرایه‌ها، متد Distinct همه‌ی کلمات تکراری را حذف می‌کند و پس از آن پرس‌وجو یک عملیات Intersect (فصل مشترک) را بر روی آرایه‌ی کلمات و آرایه‌ی

wordstoMatch انجام می‌دهد. اگر شمارگان فصل مشترک با شمارگان آرایه‌ی wordsToMatch یکسان باشد، تمامی کلماتی که در کلمات و جمله‌ی اصلی پیدا شده بودند برگردانده می‌شوند.

در فراخوان به متد LINQ، علائم نقطه‌گذاری به عنوان جداکننده به کار برده می‌شوند تا این که از رشته حذف شوند. اگر این کار را انجام ندهید، برای مثال می‌توانید دارای یک رشته‌ی "Historically," باشید که با رشته‌ی "Historically" واقع در آرایه‌ی wordsToMatch مطابقت نخواهد کرد. شما ممکن است بسته به انواع نشانه‌گذاری یافت شده در متن منبع لازم شود تا از جداکننده‌های بیشتری استفاده کنید.

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی پرس وجو برای یافتن کاراکترهای واقع در یک رشته

از آن جایی که کلاس String واسط جنریک IEnumerable(T) را پیاده‌سازی می‌کند، هر رشته‌ای می‌تواند به صورت دنباله‌ای از کاراکترها مورد پرس‌وجو قرار گیرد. هرچند، این کار یکی از کاربردهای متداول LINQ نیست. برای یافتن تطابقات الگوی پیچیده‌تر از کلاس Regex استفاده کنید.

مثال

مثال زیر یک رشته را مورد پرس وجو قرار می‌دهد تا تعداد ارقام عددی را که دربردارد تشخیص دهد. توجه داشته باشید بعد از این که برای بار نخست اجرا شد «مجدداً استفاده می‌شود». این امر به این خاطر امکان‌پذیر است که خود پرس‌وجو نمی‌تواند هیچ گونه نتیجه‌ی واقعی را ذخیره کند.

کد نمونه: **C#**

```
class QueryAString
{
    static void Main()
    {
        string aString = "ABCDE99F-J74-12-89A";

        // Select only those characters that are numbers
        IEnumerable<char> stringQuery =
          from ch in aString
          where Char.IsDigit(ch)
          select ch;

        // Execute the query
        foreach (char c in stringQuery)
            Console.Write(c + " ");

        // Call the Count method on the existing query.
        int count = stringQuery.Count();
        Console.WriteLine("Count = {0}", count);

        // Select all characters before the first '-'
        IEnumerable<char> stringQuery2 = aString.TakeWhile(c => c != '-');

        // Execute the second query
        foreach (char c in stringQuery2)
            Console.Write(c);

        Console.WriteLine(System.Environment.NewLine + "Press any key to exit");
        Console.ReadKey();

    }
}
/* Output:
  Output: 9 9 7 4 1 2 8 9
  Count = 8
  ABCDE99F
*/
```

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.

- این کد را به پروژه خود کپی کنید.

- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.

- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی ترکیب کردن پرس‌وجوهای LINQ با عبارات منظم

این مثال چگونگی استفاده از کلاس Regex برای ایجاد یک عبارت منظم به منظور یافتن تطبیق‌های پیچیده‌تر در رشته‌های متنی را نشان می‌دهد. پرس‌وجوی LINQ پالایش فایل‌هایی را که می‌خواهید با عبارات منظم جستجو کنید و شکل دادن به نتایج را آسان‌تر می‌سازد.

مثال

کد نمونه: **C#**

```
class QueryWithRegEx
{
    public static void Main()
    {
        // Modify this path as necessary.
        string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";

        // Take a snapshot of the file system.
        IEnumerable<System.IO.FileInfo> fileList = GetFiles(startFolder);

        // Create the regular expression to find all things "Visual".
        System.Text.RegularExpressions.Regex searchTerm =
            new System.Text.RegularExpressions.Regex(@"Visual
(Basic|C#|C\+\+|J#|SourceSafe|Studio)");

        // Search the contents of each .htm file.
        // Remove the where clause to find even more matches!
        // This query produces a list of files where a match
        // was found, and a list of the matches in that file.
        // Note: Explicit typing of "Match" in select clause.
```

```
        // This is required because MatchCollection is not a
        // generic IEnumerable collection.
        var queryMatchingFiles =
            from file in fileList
            where file.Extension == ".htm"
            let fileText = System.IO.File.ReadAllText(file.FullName)
            let matches = searchTerm.Matches(fileText)
            where searchTerm.Matches(fileText).Count > 0
            select new
            {
                name = file.FullName,
                matches = from System.Text.RegularExpressions.Match match in
matches
                          select match.Value
            };

        // Execute the query.
        Console.WriteLine("The term \"{0}\" was found in:",
searchTerm.ToString());


        foreach (var v in queryMatchingFiles)
        {
            // Trim the path a bit, then write
            // the file name in which a match was found.
            string s = v.name.Substring(startFolder.Length - 1);
            Console.WriteLine(s);

            // For this file, write out all the matching strings
            foreach (var v2 in v.matches)
            {
                Console.WriteLine("  " + v2);
            }
        }

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

    // This method assumes that the application has discovery
    // permissions for all folders under the specified path.
    static IEnumerable<System.IO.FileInfo> GetFiles(string path)
    {
        if (!System.IO.Directory.Exists(path))
            throw new System.IO.DirectoryNotFoundException();

        string[] fileNames = null;
        List<System.IO.FileInfo> files = new List<System.IO.FileInfo>();

        fileNames = System.IO.Directory.GetFiles(path, "*.*",
System.IO.SearchOption.AllDirectories);
        foreach (string name in fileNames)
        {
            files.Add(new System.IO.FileInfo(name));
        }
        return files;
    }
}
```

توجه داشته باشید که شما می‌توانید شیء MatchCollection را هم که توسط یک جستجوی **RexEx** برگشت داده شده است مورد پرس‌وجو قرار دهید. در این مثال تنها مقدار هر یک از تطبیق‌ها در نتایج تولید می‌شود. هرچند، استفاده از LINQ برای انجام تمامی انواع عملیات‌های فیلترینگ (پالایش)، مرتب‌سازی و گروه‌بندی بر روی آن مجموعه نیز امکان‌پذیر است. از آن جایی که MatchCollection یک مجموعه IEnumerable غیرجنریک است، شما باید به طور صریح نوع متغیر دامنه را در پرس‌وجو بیان کنید.

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید **F5** را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی ترکیب کردن و مقایسه مجموعه رشته‌ها باهم

این مثال چگونگی ترکیب کردن فایل‌هایی را نشان می‌دهد که از یک سری سطور متنی تشکیل شده‌اند و سپس نتایج را مرتب‌سازی می‌کند. به خصوص، این مثال چگونگی انجام یک عملیات ساده‌ی الحاق، یک اتحاد و یک اشتراک را بر روی دو مجموعه از خطوط متنی نشان می‌دهد.

برپایی پروژه و فایل‌های متنی

۱. این اسامی را به یک فایل متنی که names1.txt نامیده می‌شود کپی کرده و آن را در پوشه‌ی محلول خود ذخیره کنید:

کد نمونه:

```
Bankov, Peter
Holm, Michael
Garcia, Hugo
Potra, Cristina
Noriega, Fabricio
Aw, Kam Foo
Beebe, Ann
Toyoshima, Tim
Guy, Wey Yuan
Garcia, Debra
```

۲. این اسامی را به یک فایل متنی که names2.txt نامیده می‌شود کپی کرده و آن را در پوشه‌ی محلول خود ذخیره کنید. توجه داشته باشید که این دو فایل در برخی از اسامی مشترکند.

کد نمونه:

```
Liu, Jinghao
Bankov, Peter
Holm, Michael
Garcia, Hugo
Beebe, Ann
Gilchrist, Beth
Myrcha, Jacek
Giakoumakis, Leo
McLin, Nkenge
El Yassir, Mehdi
```

مثال

کد نمونه: **C#**

```
class MergeStrings
{
    static void Main(string[] args)
    {
        //Put text files in your solution folder
        string[] fileA = System.IO.File.ReadAllLines(@"../../../names1.txt");
        string[] fileB = System.IO.File.ReadAllLines(@"../../../names2.txt");

        //Simple concatenation and sort. Duplicates are preserved.
        IEnumerable<string> concatQuery =
            fileA.Concat(fileB).OrderBy(s => s);
```

```
            // Pass the query variable to another function for execution.
            OutputQueryResults(concatQuery, "Simple concatenate and sort.
Duplicates are preserved:");

            // Concatenate and remove duplicate names based on
            // default string comparer.
            IEnumerable<string> uniqueNamesQuery =
                fileA.Union(fileB).OrderBy(s => s);
            OutputQueryResults(uniqueNamesQuery, "Union removes duplicate
names:");

            // Find the names that occur in both files (based on
            // default string comparer).
            IEnumerable<string> commonNamesQuery =
                fileA.Intersect(fileB);
            OutputQueryResults(commonNamesQuery, "Merge based on intersect:");

            // Find the matching fields in each list. Merge the two
            // results by using Concat, and then
            // sort using the default string comparer.
            string nameMatch = "Garcia";

            IEnumerable<String> tempQuery1 =
                from name in fileA
                let n = name.Split(',')
                where n[0] == nameMatch
                select name;

            IEnumerable<string> tempQuery2 =
                from name2 in fileB
                let n2 = name2.Split(',')
                where n2[0] == nameMatch
                select name2;

            IEnumerable<string> nameMatchQuery =
                tempQuery1.Concat(tempQuery2).OrderBy(s => s);
            OutputQueryResults(nameMatchQuery, String.Format("Concat based on
partial name match \"{0}\":", nameMatch));

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

        static void OutputQueryResults(IEnumerable<string> query, string message)
        {
            Console.WriteLine(System.Environment.NewLine + message);
            foreach (string item in query)
            {
                Console.WriteLine(item);
            }
            Console.WriteLine("{0} total names in list", query.Count());
        }
    }
    /* Output:
        Simple concatenate and sort. Duplicates are preserved:
        Aw, Kam Foo
        Bankov, Peter
```

```
        Bankov, Peter
        Beebe, Ann
        Beebe, Ann
        El Yassir, Mehdi
        Garcia, Debra
        Garcia, Hugo
        Garcia, Hugo
        Giakoumakis, Leo
        Gilchrist, Beth
        Guy, Wey Yuan
        Holm, Michael
        Holm, Michael
        Liu, Jinghao
        McLin, Nkenge
        Myrcha, Jacek
        Noriega, Fabricio
        Potra, Cristina
        Toyoshima, Tim
        20 total names in list

        Union removes duplicate names:
        Aw, Kam Foo
        Bankov, Peter
        Beebe, Ann
        El Yassir, Mehdi
        Garcia, Debra
        Garcia, Hugo
        Giakoumakis, Leo
        Gilchrist, Beth
        Guy, Wey Yuan
        Holm, Michael
        Liu, Jinghao
        McLin, Nkenge
        Myrcha, Jacek
        Noriega, Fabricio
        Potra, Cristina
        Toyoshima, Tim
        16 total names in list

        Merge based on intersect:
        Bankov, Peter
        Holm, Michael
        Garcia, Hugo
        Beebe, Ann
        4 total names in list

        Concat based on partial name match "Garcia":
        Garcia, Debra
        Garcia, Hugo
        Garcia, Hugo
        3 total names in list
*/
```

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.

- این کد را به پروژه خود کپی کنید.

- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.

- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی یافتن تفاوت مجموعه‌ای مابین دو لیست

این مثال چگونگی استفاده از LINQ را برای مقایسه دو لیست نشان داده و آن خطوطی را که در names1.txt هستند اما در names1.txt خیر، به خروجی می‌دهد.

ایجاد فایل‌های داده

فایل‌های names1.txt و names2.txt را به صورتی که در بخش «چگونگی ترکیب کردن و مقایسه‌ی مجموعه‌ی رشته‌ها با هم» نشان داده شده است به پوشه‌ی محلول‌تان کپی کنید.

کپی

کد نمونه: **C#**

```
class CompareLists
{
    static void Main()
    {
        // Create the IEnumerable data sources.
        string[] names1 = System.IO.File.ReadAllLines(@"../../../names1.txt");
        string[] names2 = System.IO.File.ReadAllLines(@"../../../names2.txt");

        // Create the query. Note that method syntax must be used here.
        IEnumerable<string> differenceQuery =
          names1.Except(names2);

        // Execute the query.
        Console.WriteLine("The following lines are in names1.txt but not
```

```
names2.txt");
        foreach (string s in differenceQuery)
            Console.WriteLine(s);

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
/* Output:
     The following lines are in names1.txt but not names2.txt
    Potra, Cristina
    Noriega, Fabricio
    Aw, Kam Foo
    Toyoshima, Tim
    Guy, Wey Yuan
    Garcia, Debra
     */
```

برخی انوع عملیات‌های پرس‌وجو هم در C# و هم در ویژوال بیسیک، مانند Except، Distinct، Union و Concat(TSource) تنها می‌توانند به شکل ترکیب نوشتاری مبتنی بر متد بیان شوند.

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی مرتب‌سازی یا پالایش داده‌ی متنی برحسب هر کلمه یا فیلدی

مثال زیر چگونگی مرتب‌سازی خطوطی از متن ساخت یافته مانند مقادیری که با کاما از هم جدا شده‌اند را برحسب هر فیلد واقع در خط نشان می‌دهد. این فیلد می‌تواند به صورت پویا در زمان اجرا تعیین شود. فرض کنید فیلدها در فایل scores.csv بیانگر شماره شناسایی یک دانش آموزند که با ۴ نمره‌ی امتحانی پی گرفته می‌شود.

ایجاد فایل های داده

۱. این خطوط را به فایلی که scores.csv نامیده خواهد شد کپی کرده و آن را در همان پوشه‌ی پروژه خود ذخیره کنید. این فایل بیانگر داده‌ی صفحه گسترده است. ستون ID دانش آموز است و ستون ۲ تا ۵ نمرات امتحانی هستند.

کد نمونه:

```
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
```

مثال

کد نمونه: **C#**

```
public class SortLines
{
    static void Main()
    {
        // Create an IEnumerable data source
        string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");

        // Change this to any value from 0 to 4.
        int sortField = 1;

        Console.WriteLine("Sorted highest to lowest by field [{0}]:", sortField);
```

```
        // Demonstrates how to return query from a method.
        // The query is executed here.
        foreach (string str in RunQuery(scores, sortField))
        {
            Console.WriteLine(str);
        }

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

    // Returns the query variable, not query results!
    static IEnumerable<string> RunQuery(IEnumerable<string> source, int num)
    {
        // Split the string and sort on field[num]
        var scoreQuery = from line in source
                         let fields = line.Split(',')
                         orderby fields[num] descending
                         select line;

        return scoreQuery;
    }
}
/* Output (if sortField == 1):
   Sorted highest to lowest by field [1]:
    116, 99, 86, 90, 94
    120, 99, 82, 81, 79
    111, 97, 92, 81, 60
    114, 97, 89, 85, 82
    121, 96, 85, 91, 60
    122, 94, 92, 91, 91
    117, 93, 92, 80, 87
    118, 92, 90, 83, 78
    113, 88, 94, 65, 91
    112, 75, 84, 91, 39
    119, 68, 79, 88, 92
    115, 35, 72, 91, 70
 */
```

این مثال چگونگی برگرداندن یک متغیر پرس‌وجو از یک تابع (در ویژوال بیسیک) و یا متد (در سی شارپ) را نیز نشان می‌دهد.

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports**

(برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.

- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی مرتب سازی مجدد فیلدهای یک فایل قالب دار

یک فایل که مقادیر آن با کاما از هم منفک شده‌اند (CSV) فایلی است که اغلب اوقات برای ذخیره داده‌های صفحه گسترده یا انواع داده‌ی جدولی دیگری که برحسب ردیف‌ها و ستون‌ها بیان می‌شوند به کار برده می‌شود. با استفاده از متد Split برای مجزا کردن فیلدها، پرس‌وجو و دستکاری فایل‌های CSV با استفاده از LINQ کار بسیار ساده‌ای است. درواقع، تکنیک مشابهی می‌تواند برای مرتب‌سازی مجدد بخش‌هایی از هر سطر ساخت یافته از متن مورد استفاده واقع شود؛ این تکنیک محدود به فایل‌های CSV نمی‌شود.

در مثال زیر فرض بر این است که سه ستون بیانگر «نام»، «نام خانوادگی» و «شماره شناسایی» دانشجویان هستند. فیلدها برحسب ترتیب الفبایی نام خانوادگی دانشجویان مرتب شده‌اند. پرس‌وجو دنباله‌ی جدیدی تولید می‌کند که در آن ستون شماره شناسایی اول از همه ظاهر می‌شود که این ستون با ستون دومی پی گرفته می‌شود که ترکیبی از نام و نام خانوادگی دانشجوست. خطوط برحسب فیلد شماره شناسایی ازنومرتب می‌شوند. نتایج در فایل جدیدی ذخیره شده و داده‌ی اصلی بدون تغییر باقی می‌ماند.

ایجاد فایل داده

- پروژه جدید ویژوال بیسیک و یا سی شارپی را ایجاد کرده و خطوط زیر را به یک فایل متنی که spreadsheet1.csv نامیده خواهد شد کپی کنید. این فایل را در همان پوشه‌ی پروژه خود ذخیره کنید.

کد نمونه:

```
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
```

مثال

C# کد نمونه:

```
class CSVFiles
{
    static void Main(string[] args)
    {
        // Create the IEnumerable data source
        string[] lines =
System.IO.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 x = line.Split(',')
            orderby x[2]
            select x[2] + ", " + (x[1] + " " + x[0]);

        // Execute the query and write out the new file. Note that WriteAllLines
        // takes a string[], so ToArray is called on the query.
        System.IO.File.WriteAllLines(@"../../../spreadsheet2.csv",
query.ToArray());

        Console.WriteLine("Spreadsheet2.csv written to disk. Press any key to
exit");
        Console.ReadKey();
    }
}
/* 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
 */
```

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی پر کردن مجموعه‌های شیئی از طریق منابع متعدد

این مثال چگونگی ادغام داده‌ها از انواع منابع داده مختلف در دنباله‌ای از انواع جدید را نشان می‌دهد. نمونه‌های واقع در کد زیر رشته‌ها را با آرایه‌های صحیح ادغام می‌کند. اگرچه، مفهوم یکسانی به هرگونه منابع داده دیگر اعمال می‌شود، از جمله ترکیب اشیاء درون حافظه‌ای (شامل نتایج ناشی از پرس‌وجوهای LINQ به SQL، مجموعه داده‌های ADO.NET و اسناد XML).

نکته:

سعی نکنید تا داده‌های درون حافظه‌ای یا داده‌ی واقع در سیستم فایل را با داده‌ای که هنوز در یک پایگاه داده قرار دارد ادغام کنید. یک چنین اتصالات حوزه متقاطعی به خاطر روش‌های متفاوتی که در آنها عملیات‌های اتصال می‌توانند برای پرس‌وجوهای پایگاه داده و دیگر انواع منابع تعریف شوند می‌تواند نتایج تعریف نشده‌ای را در پی داشته باشند. علاوه بر این، این خطر وجود دارد که یک چنین عملیاتی بتواند در صورتی که داده‌ی واقع در پایگاه داده به اندازه کافی بزرگ باشد منجر به یک استثنای خارج از حد حافظه گردد. برای ادغام داده از یک پایگاه داده با داده‌ی درون حافظه‌ای، نخست **ToList** یا **ToArray** را روی پرس‌وجوی پایگاه داده

فراخوان کنید و بعد از آن عمل ادغام را بر روی مجموعه برگشتی انجام دهید.

ایجاد فایل‌های داده

۱. این خطوط را به فایلی که scores.csv نامیده خواهد شد کپی کرده و آن را در همان پوشه‌ی پروژه خود ذخیره کنید. این فایل بیانگر داده‌ی صفحه گسترده است. ستون ID دانش آموز است و ستون ۲ تا ۵ نمرات امتحانی هستند.

کد نمونه:

```
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
```

۲. این خطوط را به فایلی که names.csv نامیده خواهد شد کپی کنید و آن را در همان پوشه‌ی پروژه‌تان ذخیره کنید. این فایل بیانگر صفحه گسترده‌ای است که حاوی نام خانوادگی، نام و شماره شناسایی دانش آموزان است.

کد نمونه:

```
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
```

مثال

مثال زیر چگونگی استفاده از یک نوع به نام `Student` را برای ذخیره داده‌ی ادغام شده از دو مجموعه درون حافظه‌ای از رشته‌ها که داده‌ی صفحه گسترده را در قالب csv. شبیه‌سازی می‌کنند نشان می‌دهد. نخستین مجموعه از رشته‌ها بیانگر اسامی دانش آموزان و شماره شناسایی‌شان است و مجموعه‌ی دوم بیانگر شماره شناسایی دانش آموزان (در ستون نخست) و چهار نمره‌ی امتحانی آنهاست.

کد نمونه: **C#**

```
class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int ID { get; set; }
    public List<int> ExamScores { get; set; }
}

class PopulateCollections
{
    static void Main()
    {
        // These data files are defined in How to: Join Content from Dissimilar
Files (LINQ)
        string[] names = System.IO.File.ReadAllLines(@"../../../names.csv");
        string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");

        // Merge the data sources using a named type.
        // var could be used instead of an explicit type.
        // Note the dynamic creation of a list of ints for the
        // TestScores member. We skip 1 because the first string
        // in the array is the student ID, not an exam score.
        IEnumerable<Student> queryNamesScores =
            from name in names
            let x = name.Split(',')
            from score in scores
            let s = score.Split(',')
            where x[2] == s[0]
            select new Student()
            {
                FirstName = x[0],
                LastName = x[1],
                ID = Convert.ToInt32(x[2]),
                ExamScores = (from scoreAsText in s.Skip(1)
                              select Convert.ToInt32(scoreAsText)).
                              ToList()
            };

        // Optional. Store the newly created student objects in memory
        // for faster access in future queries. Could be useful with
        // very large data files.
        List<Student> students = queryNamesScores.ToList();
```

```
        // Display the results and perform one further calculation.
        foreach (var student in students)
        {
            Console.WriteLine("The average score of {0} {1} is {2}.",
                student.FirstName, student.LastName,
student.ExamScores.Average());
        }

        //Keep console window open in debug mode
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/* Output:
    The average score of Adams Terry is 85.25.
    The average score of Fakhouri Fadi is 92.25.
    The average score of Feng Hanying is 88.
    The average score of Garcia Cesar is 88.25.
    The average score of Garcia Debra is 67.
    The average score of Garcia Hugo is 85.75.
    The average score of Mortensen Sven is 84.5.
    The average score of O'Donnell Claire is 72.25.
    The average score of Omelchenko Svetlana is 82.5.
    The average score of Tucker Lance is 81.75.
    The average score of Tucker Michael is 92.
    The average score of Zabokritski Eugene is 83.
 */
```

منابع داده واقع در این مثال‌ها با مقداراولیه دهنده‌های شیئی مقداردهی می‌شوند. پرس‌وجو از از یک ضابطه‌ی **join** استفاده می‌کند تا اسامی را با نمرات تطبیق دهد. `ID` به عنوان کلید خارجی (foreign key) به کار برده می‌شود. اگرچه، در یک منبع `ID` یک رشته است و در منبع دیگر یک عدد صحیح است. از آن جایی که **join** نیازمند یک مقایسه برابری است، شما باید نخست `ID` را از رشته استخراج کنید و آن را به یک عدد صحیح تبدیل کنید. این کار در دو ضابطه‌ی **let** انجام می‌شود. شناسه‌ی واسط `x` در ضابطه‌ی **let** اولی آرایه‌ای از سه رشته‌ی ایجاد شده توسط تبدیل زیررشته‌ی `ID` به یک عدد صحیح را در خود ذخیره می‌کند. در ضابطه‌ی **select**، یک مقداردهنده‌ی اولیه‌ی شیئی به کار برده می‌شود تا هر یک از اشیاء `Student` جدید را با استفاده از داده‌ی آمده از دو منبع مقداردهی اولیه کند.

اگر لزومی ندارد تا نتایج یک پرس‌وجو را ذخیره کنید، انواع بی‌نام می‌توانند سرراست‌تر از انواع دارای نام باشند. انواع دارای نام زمانی لازم هستند که شما نتایج پرس‌وجو را به خارج از متدی که پرس‌وجو آنجا اجرا می‌شود ارسال می‌کنید.

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی ادغام محتویات از فایل‌های نامشابه

این مثال چگونگی ادغام داده‌های آمده از دو فایل که مقادیرشان با کاما ازهم جدا شده‌اند و مقدار یکسانی را به اشتراک می‌گذارند که این مقدار به عنوان کلید تطبیق به کار برده می‌شود را نشان می‌دهد. اگر لازم است تا داده‌ی آمده از دو صفحه گسترده و یا داده‌ی آمده از یک صفحه گسترده و یک فایل که دارای قالب دیگری است را در یک فایل جدید با هم ترکیب کنید این تکنیک سودمند خواهد بود. شما می‌توانید این مثال را تغییر دهید تا با هر نوع متن ساخت یافته‌ی دیگری هم کار کند.

توجه برای ایجاد فایل‌های داده همان مراحل بخش قبل را در پیش بگیرید.

Example

C# کد نمونه:

```
class JoinStrings
{
    static void Main()
    {
        // Join content from dissimilar files that contain
        // related information. names.csv contains the student name
        // plus an ID number. scores. csv contains the ID and a
        // set of four test scores. The following query joins
        // the scores to the student names by using ID as a
        // matching key.
```

```
        string[] names = System.IO.File.ReadAllLines(@"../../../names.csv");
        string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");


        // Name:    Last[0],       First[1],  ID[2],     Grade Level[3]
        //          Omelchenko,    Svetlana,  11,        2
        // Score:   StudentID[0],  Exam1[1]   Exam2[2],  Exam3[3],  Exam4[4]
        //          111,           97,        92,        81,        60

        // This query joins two dissimilar spreadsheets based on common ID value.
        // Multiple from clauses are used instead of a join clause
        // in order to store results of id.Split.
        IEnumerable<string> scoreQuery1 =
            from name in names
            let nameFields = name.Split(',')
            from id in scores
            let scoreFields = id.Split(',')
            where nameFields[2] == scoreFields[0]
            select nameFields[0] + "," + scoreFields[1] + "," + scoreFields[2]
                   + "," + scoreFields[3] + "," + scoreFields[4];

        // Pass a query variable to a method and
        // execute it in the method. The query itself
        // is unchanged.
        OutputQueryResults(scoreQuery1, "Merge two spreadsheets:");

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

    static void OutputQueryResults(IEnumerable<string> query, string message)
    {
        Console.WriteLine(System.Environment.NewLine + message);
        foreach (string item in query)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine("{0} total names in list", query.Count());
    }
}
/* Output:
Merge two spreadsheets:
Adams, 99, 82, 81, 79
Fakhouri, 99, 86, 90, 94
Feng, 93, 92, 80, 87
Garcia, 97, 89, 85, 82
Garcia, 35, 72, 91, 70
Garcia, 92, 90, 83, 78
Mortensen, 88, 94, 65, 91
O'Donnell, 75, 84, 91, 39
Omelchenko, 97, 92, 81, 60
Tucker, 68, 79, 88, 92
Tucker, 94, 92, 91, 91
Zabokritski, 96, 85, 91, 60
12 total names in list
 */
```

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی تقسیم یک فایل به چندین فایل با استفاده از گروه‌ها

این مثال روشی را برای ادغام محتویات دو فایل و سپس ایجاد مجموعه‌ای از فایل‌های جدید که داده را به شیوه‌ی جدیدی سازماندهی می‌کنند نشان می‌دهد.

برای ایجاد فایل‌های داده

۱. این اسامی را به فایل متنی که names1.txt نامیده خواهد شد کپی کرده و آن را در همان پوشه‌ی محلول خود ذخیره کنید:

کد نمونه:

```
Bankov, Peter
Holm, Michael
Garcia, Hugo
Potra, Cristina
Noriega, Fabricio
Aw, Kam Foo
Beebe, Ann
Toyoshima, Tim
Guy, Wey Yuan
Garcia, Debra
```

۲. این اسامی را به فایل متنی که names2.txt نامیده خواهد کپی کرده و آن را در پوشه‌ی محلول خود ذخیره کنید. توجه داشته باشید که هر دو فایل دارای برخی اسامی مشترکند:

کد نمونه:

```
Liu, Jinghao
Bankov, Peter
Holm, Michael
Garcia, Hugo
Beebe, Ann
Gilchrist, Beth
Myrcha, Jacek
Giakoumakis, Leo
McLin, Nkenge
El Yassir, Mehdi
```

مثال

کد نمونه: **C#**

```
class SplitWithGroups
{
    static void Main()
    {
        string[] fileA = System.IO.File.ReadAllLines(@"../../../names1.txt");
        string[] fileB = System.IO.File.ReadAllLines(@"../../../names2.txt");

        // Concatenate and remove duplicate names based on
        // default string comparer
        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(',')
                         group name by n[0][0] into g
                         orderby g.Key
                         select g;

        // Create a new file for each group that was created
        // Note that nested foreach loops are required to access
        // individual items with each group.
        foreach (var g in groupQuery)
        {
            // Create the new file name.
            string fileName = @"../../../testFile_" + g.Key + ".txt";

            // Output to display.
            Console.WriteLine(g.Key);

            // Write file.
            using (System.IO.StreamWriter sw = new
System.IO.StreamWriter(fileName))
            {
```

```
                foreach (var item in g)
                {
                    sw.WriteLine(item);
                    // Output to console for example purposes.
                    Console.WriteLine("   {0}", item);
                }
            }
        }
        // Keep console window open in debug mode.
        Console.WriteLine("Files have been written. Press any key to exit");
        Console.ReadKey();
    }
}
/* 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
 */
```

برنامه برای هر یک از گروه‌ها فایل مجزایی را در همان پوشه‌ی فایل‌های داده (پوشه‌ی محلول) می‌نویسد.

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای #C) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های #C یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.

- این کد را به پروژه خود کپی کنید.
- کلید F5 را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

چگونگی محاسبه‌ی مقادیر ستون‌ها در یک فایل متنی CSV

این مثال چگونگی انجام محاسبات به هم پیوسته‌ای چون Sum، Average، Min و Max را بر روی ستون‌های یک فایل csv. نشان می‌دهد. مفاهیمی که در اینجا نشان داده شده‌اند قابل اعمال به هر نوع دیگری از متون ساخت یافته‌اند.

ایجاد فایل داده‌ی منبع

- این خطوط را به فایلی که scores.csv نامیده خواهد شد کپی کرده و آن را در همان پوشه‌ی پروژه خود ذخیره کنید. این فایل بیانگر داده‌ی صفحه گسترده است. ستون ID دانش آموز است و ستون ۲ تا ۵ نمرات امتحانی هستند.

کد نمونه:

```
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
```

مثال

کد نمونه: 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
        // one is added 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:");

        // examNum specifies the column to run the
        // calculations on. This could also be
        // passed in dynamically at runtime.

        // columnQuery is a IEnumerable<int>
        // This query performs two steps:
        // 1) split the string into a string[]
        // 2) convert the specified element to
        //    int and select it.
        var columnQuery =
            from line in strs
            let x = line.Split(',')
            select Convert.ToInt32(x[examNum]);

        // Execute and cache the results for performance.
        // Only needed with very large files.
        var results = columnQuery.ToList();

        // Perform aggregate calculations
        // 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 the columnQuery. Explicit typing is used
        // to make clear that the columnQuery will produce
        // nested sequences. You can also just use 'var'.
```

```
        // The columnQuery performs these steps:
        // 1) convert the string to a string[]
        // 2) skip over the "Student ID" column and take the rest
        // 3) convert each string to an int and select that
        //    entire sequence as one row in the results.
        IEnumerable<IEnumerable<int>> query =
            from line in strs
            let x = line.Split(',')
            let y = x.Skip(1)
            select (from str in y
                    select Convert.ToInt32(str));

        // Execute and cache the results for performance.
        // ToArray could also be used here.
        var results = query.ToList();

        // Find out how many columns we have.
        int columnCount = results[0].Count();

        // Perform aggregate calculations on each column.
        // One loop for each score column in scores.
        // We can use a for loop because we have already
        // executed the columnQuery in the call to ToList.
        for (int column = 0; column < columnCount; column++)
        {
            var res2 = from row in results
                       select row.ElementAt(column);
            double average = res2.Average();
            int max = res2.Max();
            int min = res2.Min();

            // 1 is added to column because Exam numbers
            // begin with 1
            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
 */
```

اگر فایل شما یک فایل با قالب جدولی است، تنها آرگومان واقع در متد `Split` را به `\t` تغییر دهید.

کامپایل کردن کد:

- یک پروژه ویژوال بیسیک را که.NET Framework نسخه‌ی 3.5 را هدف گرفته باشد ایجاد کنید. پروژه به طور پیش فرض دارای یک ارجاع به System.Core.dll و یک راهنمای **using** (برای C#) و یا **Imports** (برای ویژوال بیسیک) برای فضای نام System.Linq است. در پروژه‌های C# یک راهنمای **using** را برای فضای نام System.IO اضافه کنید.
- این کد را به پروژه خود کپی کنید.
- کلید **F5** را بزنید تا برنامه کامپایل و اجرا شود.
- برای بستن پنجره‌ی کنسول کلیدی را فشار دهید.

اللهم عجل لولیک الفرج

مهدی محبیان

بهار ۹۰