ASP.NET - LINQ

Most applications are data-centric, however most of the data repositories are relational databases. Over the years, designers and developers have designed applications based on object models.

The objects are responsible for connecting to the data access components - called the Data Access Layer *DAL*. Here we have three points to consider:

- All the data needed in an application are not stored in the same source. The source could be a relation database, some business object, XML file, or a web service.
- Accessing in-memory object is simpler and less expensive than accessing data from a database or XML file.
- The data accessed are not used directly, but needs to be sorted, ordered, grouped, altered etc.

Hence if there is one tool that makes all kind of data access easy that allows joining data from such disparate data sources and perform standard data processing operations, in few lines of codes, it would be of great help.

LINQ or Language-Integrated Query is such a tool. LINQ is set of extensions to the .Net Framework 3.5 and its managed languages that set the query as an object. It defines a common syntax and a programming model to query different types of data using a common language.

The relational operators like Select, Project, Join, Group, Partition, Set operations etc., are implemented in LINQ and the C# and VB compilers in the .Net framework 3.5, which support the LINQ syntax makes it possible to work with a configured data store without resorting to ADO.NET.

For example, querying the Customers table in the Northwind database, using LINQ query in C#, the code would be:

```
var data = from c in dataContext.Customers
where c.Country == "Spain"
select c;
```

Where:

- The 'from' keyword logically loops through the contents of the collection.
- The expression with the 'where' keyword is evaluated for each object in the collection.
- The 'select' statement selects the evaluated object to add to the list being returned.
- The 'var' keyword is for variable declaration. Since the exact type of the returned object is not known, it indicates that the information will be inferred dynamically.

LINQ query can be applied to any data-bearing class that inherits from IEnumerable<T>, here T is any data type, for example, List<Book>.

Let us look at an example to understand the concept. The example uses the following class: Books.cs

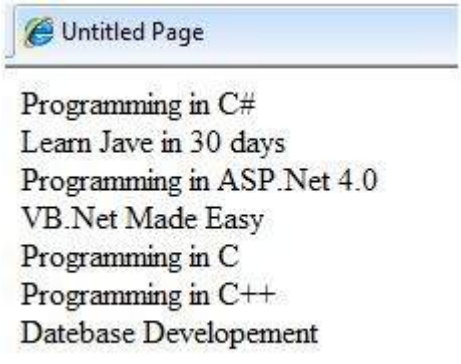
```
public class Books
{
   public string ID {get; set;}
   public string Title { get; set; }
   public decimal Price { get; set; }
   public DateTime DateOfRelease { get; set; }

   public static List<Books> GetBooks()
   {
      List<Books> list = new List<Books>();
```

```
      list.Add(new Books { ID = "001",
         Title = "Programming in C#",
         Price = 634.76m,
         DateOfRelease = Convert.ToDateTime("2010-02-05") });

      list.Add(new Books { ID = "002",
         Title = "Learn Jave in 30 days",
         Price = 250.76m,
         DateOfRelease = Convert.ToDateTime("2011-08-15") });

      list.Add(new Books { ID = "003",
         Title = "Programming in ASP.Net 4.0",
         Price = 700.00m,
         DateOfRelease = Convert.ToDateTime("2011-02-05") });

      list.Add(new Books { ID = "004",
         Title = "VB.Net Made Easy",
         Price = 500.99m,
         DateOfRelease = Convert.ToDateTime("2011-12-31") });

      list.Add(new Books { ID = "005",
         Title = "Programming in C",
         Price = 314.76m,
         DateOfRelease = Convert.ToDateTime("2010-02-05") });

      list.Add(new Books { ID = "006",
         Title = "Programming in C++",
         Price = 456.76m,
         DateOfRelease = Convert.ToDateTime("2010-02-05") });

      list.Add(new Books { ID = "007",
         Title = "Datebase Developement",
         Price = 1000.76m,
         DateOfRelease = Convert.ToDateTime("2010-02-05") });

      return list;
   }
}
```

The web page using this class has a simple label control, which displays the titles of the books. The Page_Load event creates a list of books and returns the titles by using LINQ query:

```
public partial class simplequery : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      List<Books> books = Books.GetBooks();
      var booktitles = from b in books select b.Title;

      foreach (var title in booktitles)
         lblbooks.Text += String.Format("{0} <br />", title);
   }
}
```

When the page is executed, the label displays the results of the query:



The above LINQ expression:

```
var booktitles =
from b in books
select b.Title;
```

Is equivalent to the following SQL query:

```
SELECT Title from Books
```

LINQ Operators

Apart from the operators used so far, there are several other operators, which implement all query clauses. Let us look at some of the operators and clauses.

The Join clause

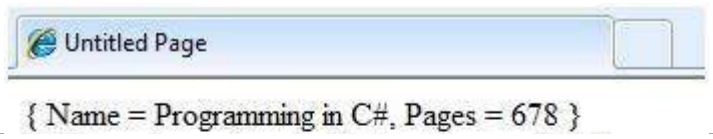
The 'join clause' in SQL is used for joining two data tables and displays a data set containing columns from both the tables. LINQ is also capable of that. To check this, add another class named Saledetails.cs in the previous project:

```
public class Salesdetails
{
   public int sales { get; set; }
   public int pages { get; set; }
   public string ID {get; set;}

   public static IEnumerable<Salesdetails> getsalesdetails()
   {
      Salesdetails[] sd =
      {
         new Salesdetails { ID = "001", pages=678, sales = 110000},
         new Salesdetails { ID = "002", pages=789, sales = 60000},
         new Salesdetails { ID = "003", pages=456, sales = 40000},
         new Salesdetails { ID = "004", pages=900, sales = 80000},
         new Salesdetails { ID = "005", pages=456, sales = 90000},
         new Salesdetails { ID = "006", pages=870, sales = 50000},
         new Salesdetails { ID = "007", pages=675, sales = 40000},
      };

      return sd.OfType<Salesdetails>();
   }
}
```

Add the codes in the Page_Load event handler to query on both the tables using the join clause:

```
protected void Page_Load(object sender, EventArgs e)
{
   IEnumerable<Books> books = Books.GetBooks();
   IEnumerable<Salesdetails> sales = Salesdetails.getsalesdetails();

   var booktitles = from b in books join s in sales on b.ID equals s.ID
      select new { Name = b.Title, Pages = s.pages };

   foreach (var title in booktitles)
      lblbooks.Text += String.Format("{0} <br />", title);
}
```

The resulting page is as shown:



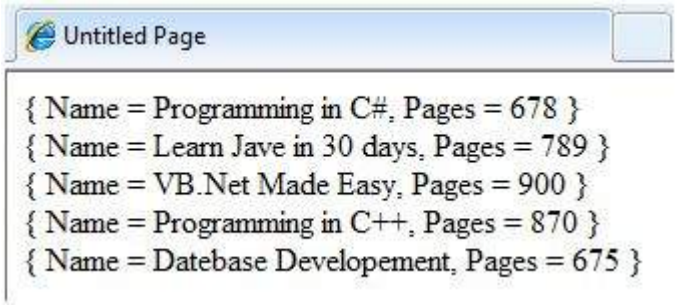
{ Name = Learn Jave in 30 days, Pages = 789 }
{ Name = Programming in ASP.Net 4.0, Pages = 456 }
{ Name = VB.Net Made Easy, Pages = 900 }
{ Name = Programming in C, Pages = 456 }
{ Name = Programming in C++, Pages = 870 }
{ Name = Datebase Developement, Pages = 675 }

The Where clause

The 'where clause' allows adding some conditional filters to the query. For example, if you want to see the books, where the number of pages are more than 500, change the Page_Load event handler to:

```
var booktitles = from b in books join s in sales on b.ID equals s.ID
   where s.pages > 500 select new { Name = b.Title, Pages = s.pages };
```

The query returns only those rows, where the number of pages is more than 500:

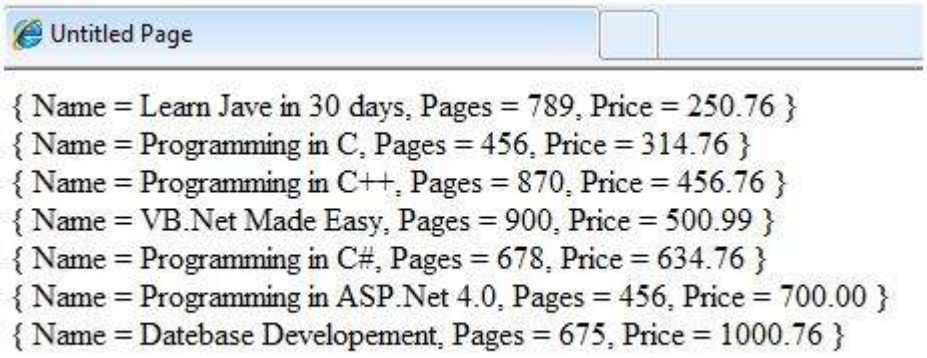


Orderby and Orderbydescending Clauses

These clauses allow sorting the query results. To query the titles, number of pages and price of the book, sorted by the price, write the following code in the Page_Load event handler:

```
var booktitles = from b in books join s in sales on b.ID equals s.ID
   orderby b.Price select new { Name = b.Title,  Pages = s.pages, Price = b.Price};
```

The returned tuples are:



The Let clause

The let clause allows defining a variable and assigning it a value calculated from the data values. For example, to calculate the total sale from the above two sales, you need to calculate:

```
TotalSale = Price of the Book * Sales
```

To achieve this, add the following code snippets in the Page_Load event handler:

The let clause allows defining a variable and assigning it a value calculated from the data values. For example, to calculate the total sale from the above two sales, you need to calculate:

```
var booktitles = from b in book join s in sales on b.ID equals s.ID
   let totalprofit = (b.Price * s.sales)
   select new { Name = b.Title, TotalSale = totalprofit};
```

The resulting query page is as shown: