LINQ

Intro

Language-Integrated Query (LINQ) adalah suatu inovasi yang diperkenalkan pada Visual Studio 2008 dan .NET Framework 3.5 yang menghubungkan antara object dan data. LINQ memungkinkan programmer untuk melakukan query *inline* sebagaimana mereka menggunakan standar sintaks SQL.

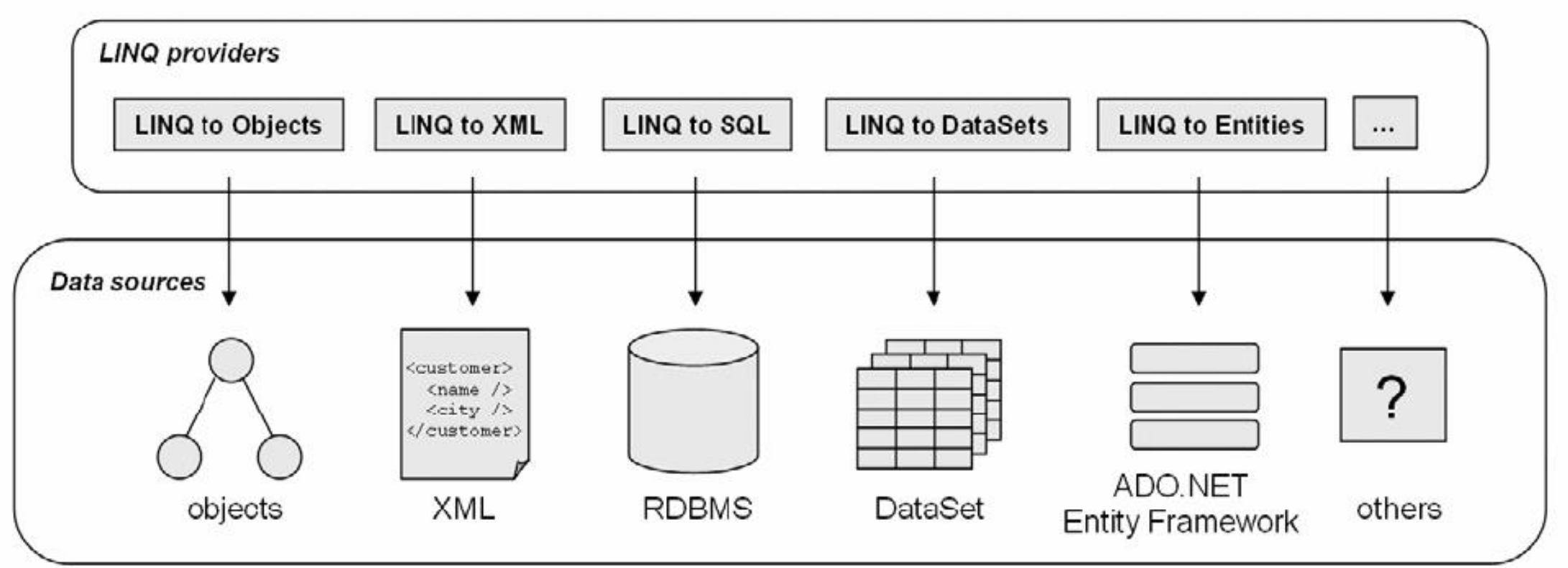
LINQ juga memperkenalkan konsep-konsep yang sebenarnyaa berasal dari bahasa pemrograman fungsional seperti Haskell, LISP. Beberapa konsep-konsep baru adalah:

- Lambda, yang mengijinkan fungsi *anonymous* (method di .NET) untuk di panggil lebih dari satu urutan.
- Pengolahan rekursif lebih dari satu urutan.
- *Lazy evaluation*

Kelebihan

- LINQ dapat digunakan untuk datasource yang berbeda-beda.
- LINQ dapat diaplikasikan ke objek-objek yang mendukung IEnumerable atau interface generik IEnumerable<T>.
- LINQ juga mendukung Entity Framework ADO .Net dan provider-provider LINQ lainnya.

LINQ Building Blocks



LINQ ke Objek

Istilah “LINQ ke Object” mengacu pada penggunaan query LINQ dengan koleksi IEnumerable atau IEnumerable secara langsung. Programmer dapat menggunakan koleksi enumerable seperti List<T>, Array, or Dictionary<Tkey, Tvalue>. Koleksi yang digunakan dapat berupa user-defined ataupun yang berasal dari .NET Framework API.

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

Sorting and Grouping

```
var groups    =    from word in words orderby word ascending
                   group word by word.Length into lengthGroups
                   orderby lengthGroups.Key descending
                   select new { Length = lengthGroups.Key, Words = lengthGroups };
```

LINQ Aggregate Functions

- Max
- Min
- Sum
- Average
- Aggregate
- Dll.

```
decimal totalAmount = orders.Sum(order => order.Amount);
```

LINQ Aggregate Functions

```
//Aggregate
   string[] MySkills = {
           "C#.net",
           "Asp.net",
           "MVC",
           "Linq",
           "EntityFramework",
           "Swagger",
           "Web-Api",
           "OrcharCMS",
           "Jquery",
           "Sqlserver",
           "Docusign"
   };
   var commaSeperatedString = MySkills.Aggregate((s1, s2) => s1 + ", " + s2);
```

Output :

```
"C#.net, Asp.net, MVC, Linq, EntityFramework,
Swagger, Web-Api, OrcharCMS, Jquery,
Sqlserver, Docusign"
```

LINQ Ke XML

“LINQ ke XML” menyediakan query dan kemapuan transformasi dari XQuery dan XPath yang terintegrasi ke dalam .NET. Adapaun fasilitas yang disediakan adalah kemampuan untuk meng-edit dokumen XML dan tree element in-memory, juga kemampuang streaming, yang berarti programmer dapat menggunakan LINQ ke XML untuk melakukan banyak tugas-tugas proses XML dengan lebih mudah.

LINQ ke XML

```
// Our book collection
    Book[] books = new Book[] {
    new Book("Ajax in Action", "Manning", 2005),
    new Book("Windows Forms in Action", "Manning", 2006),
    new Book("ASP.NET 2.0 Web Parts in Action", "Manning", 2006)
    };
    // Build the XML fragment based on the collection
    XElement xml = new XElement("books",
    from book in books
    where book.Year == 2006
    select new XElement("book",
        new XAttribute("title", book.Title),
        new XElement("publisher", book.Publisher)
    )
    );
    // Dump the XML to the console
    Console.WriteLine(xml);
    }
```

LINQ Ke SQL

“LINQ ke SQL” menyediakan *languange-integrated data access* dengan mengunakan mekanisme ekstens LINQ. LINQ ke SQL dibangun dengan pada ADO.NET untuk memetakan tabel dan baris ke class dan object.

Entity

Tahap pertama untuk membangung LINQ ke SQL adalah dengan medeklarasikan kelas yang akan merepresentasikan data, yaitu entities.

```
[Table(Name="Contacts")]
 class Contact
 {
    [Column(IsPrimaryKey=true)]
     public int ContactID;
    [Column(Name="ContactName")]
     public string Name;
    [Column]
     public string City;
 }
```

- Attribut Table disediakan LINQ ke SQL, mempunya properti nama untuk menentukan nama dari tabel database.
- Attribut Column mempuyai berbagai properti yg dapat digunakan untuk kostumisasi pemetaan yang tepat antara field atau properti dan kolom database.

DataContext

Hal berikutnya yang harus dipersiapkan sebelum dapat menggunakan query *language-integrated* adalah object ***System.Data.Linq.DataContext***. Tujuan dari *DataContext* adalah mentranslasikan permintaan dari objek kedalam query SQL yang dibuat terhadap database dan kemudian merakit object hasil.

Konstruktor dari *DataContext* membutuhkan *connection string* sebagai parameter.

```
string dbPath = Path.GetFullPath(@"..\..\..\..\Data\northwnd.mdf");
DataContext db = new DataContext(dbPath);
```

DataContext

DataContext menyediakan akses ke tabel dalam database.

```
Table<Contact> contacts = db.GetTable<Contact>();
```

Full Code

```
using System;
using System.Linq;
using System.Data.Linq;
static class HelloLinqToSql
{
    [Table(Name="Contacts")]
    class Contact
    {
        [Column(IsPrimaryKey=true)]
        public int ContactID;
        [Column(Name="ContactName")]
        public string Name;
        [Column]
        public string City;
    }
```

```
 static void Main()
    {
        // Get access to the database
        string dbPath = Path.GetFullPath(@"..\..\..\..\Data\northwind.mdf");
        DataContext db = new DataContext(dbPath);
        // Query for contacts from Paris
        var contacts    =    from contact in db.GetTable<Contact>()
                             where contact.City == "Paris"
                             select contact;
        // Display the list of matching contacts
        foreach (var contact in contacts)
           Console.WriteLine("Bonjour "+contact.Name);
    }
}
```