Exploring and Analyzing Language Integrated Query (LINQ) in .Net

Lakshmi Sudha Marothu
Computing Studies, Arizona State University

Abstract

Language Integrated Query (LINQ) is a general purpose query facility added to .Net Framework 3.5[1]. LINQ has the ability to query any source of data, be it relational database, XML files or a collection of objects. In this project I made an attempt to explore and analyze LINQ using three applications that I built using LINQ and without LINQ. The first application is a Home Budget built on PostgreSQL database which keeps track of income, expenditure and savings of a household. LINQ for SQL component is used to query Home Budget application. The second application is an Electronics store built on XML that has a list of all electronic gadgets present in a store. LINQ for XML API is used to query Electronics Store application. The third application developed is a Hashtable which uses a collection of objects to store key value pairs. LINQ for objects component is used to query Hashtable. Finally, I concluded by comparing the advantages and disadvantages observed while using LINQ and analyzing if LINQ is a value addition to .Net Framework 3.5 or not.

I. INTRODUCTION

One of the challenges in programming is to reduce the complexity of accessing information that is not natively defined, using object oriented technology [1]. In particular, database access and XML manipulation are often cumbersome. Parsing and manipulation of XML are tedious as well. As we cannot programmatically interact with the database at the native language level, syntax errors can go undetected until runtime. Rather than adding classes and methods to address these deficiencies, Microsoft decided to add general-purpose query facilities to the .NET Framework 3.5 that apply to all sources of information, not just relational or XML data. This facility is called .NET Language-Integrated Query (LINQ).LINQ enables us to query and manipulate data by using a consistent model that is independent of data sources. Microsoft uses LINQ to query three major data sources-relational databases, XML and objects. LINQ to database, LINQ to XML and LINQ to Objects are different components of LINQ. Each of these implementations is defined through a set of extension. Access to these features is controlled by the imported namespaces.

In this project, I explored the effectiveness of using LINQ in three major data sources -database, XML and objects with examples.

II. LITERATURE REVIEW

A. Ling Architecture

Language Integrated Query (LINQ) is a new feature in .Net Framework 3.5 that extends query capabilities, in C# and Visual Basic languages. .Net Framework 3.5 comes with LINQ provider assemblies that enable the use of Language Integrated Queries with different data sources. LINQ provides uniform ways to perform action on any object which implements the IEnumerable<T> OR IQueryable<T> interface with different type of data.

Figure 1: LINQ Architecture [16]
The above figure [Figure 1] shows the architecture of LINQ, which can query different data sources using different programming languages.

B. Simple LINQ Query Format

LINQ keywords provide the basis for writing a query. In some respects, these keywords work and act like the keywords in SQL. Here is a simple format of LINQ Query with an example.

Var ThisQuery = from StringValue in QueryString
    where StringValue.Condition
    select StringValue;

Code Example:

String[] linqDataSource = { "One", "Two", "Three", "Four", "Five" };

var linqQuery = from query in linqDataSource
    where query.Length >= 4
    select query + "\r\n";

foreach (var ThisValue in linqQuery)
    Console.WriteLine(ThisValue);

‘linqDataSource’ in the above code contains an array of data values. ‘linqQuery’ is a LINQ query of type ‘var’. ‘var’ is implicit type local variable. ‘var’ keyword instructs the compiler to infer the type of the variable from the expression on the right side of the initialization statement. ‘query’ is a range variable. A range variable is like an iteration variable in a foreach statement but it never actually stores data from the source. ‘linqQuery’ is used to select values from ‘linqDataSource’ where string length is greater than or equal to four.

The output of this query would be “Three”, “Four” and “Five”.

C. LINQ Namespaces

There is no single LINQ namespace. .Net Framework 3.5 has a number of LINQ namespaces, each of which creates a different kind of data connection. System.Linq namespace contains all basic classes and interfaces for LINQ. System.Linq.Expressions namespace contains classes, interfaces, and enumerations used to create expressions. For example, a binary expression can be defined to subtract one number from another. System.Data.Linq namespace contains the classes, structures, interfaces, and enumerations used for relational database interactions. This is the basic namespace used for LINQ to SQL scenarios. The classes in System.Data.Linq namespace perform operations like SELECT, UPDATE, DELETE and many more. System.Data.Linq.Mapping namespace contains classes and enumerations to map data between an imperative language such as C# or Visual Basic .NET and a declarative language such as SQL. System.Data.SqlClient namespace contains the classes used to create a basic connection with SQL Server. System.Xml.Linq namespace contains classes and enumerations used to interact with XML data of all type.

D. Using LINQ to access database

There is a huge divide between modern programming languages and databases in the way information is represented and manipulated. The data model is different between the two. Modern programming languages define information in the form of objects where as relational databases uses tables. This is the reason why programming languages access information in database through queries that are specified as text strings. These text string queries do not benefit from compile-time verification and design time intellisense [2].

LINQ to SQL is used to query relational data without leaving the syntax or compile-time environment of local programming language. LINQ to SQL takes care of translating LINQ expressions to equivalent SQL and passing it on to the database for execution. It also returns the results back to the calling application by tracking changes made to the objects. LINQ to SQL integrates SQL schema information into Common Language Runtime in .Net that can be accessed through code. It defines [Table] and [Column] attributes which are parameterized. [Table] attribute can be applied to a class and [Column] attribute can be applied to any field or property. Objects linked to relational data can be defined just like normal objects. LINQ to SQL supports stored procedures and user-defined functions in the database. A design-time tool is provided to automate translating pre-existing relational database schemas into object definitions.

The following example shows an SQL table and its equivalent C# code.

SQL Table:
Create table People (  
    Name nvarchar(32) primary key not null,  
    Age int not null,  
);

C# equivalent code:

[Table(Name="People")]
public class Person  
{  
    [Column(DbType="nvarchar(32) not null",  
    Id=true)]  
    public string Name;  
    [Column]  
    public int Age;  
}
On the top we have an SQL table ‘People’ that has two columns ‘Name’ and ‘Age’. Below is the C# code that defines mapping between SQL table and C# classes. [Table] attribute has a ‘Name’ property that is used to specify the name of database table ‘People’. [Column] attribute defines ‘Name’ and ‘Age’ columns.

LINQ to SQL supports Microsoft SQL Server starting from SQL Server 2000 and Microsoft SQL Server Compact starting from version 3.5. Microsoft does not directly support other data sources like PostgreSQL, MySQL, Oracle or excel for LINQ. There are various other providers available for LINQ to query data from sources beyond what Microsoft provides. DBLinq is the LINQ provider for MySql, Oracle and Postgresql. JSLINQ is a LINQ provider for JavaScript. LINQ to Excel is a LINQ provider for Excel. The website developed by Charlie Calvert called “Links to LINQ” [15] provides list all the available third party providers available for LINQ.

LINQ to SQL has better performance when compared to ADO.Net entity framework [3]. LINQ to SQL applications are easy to work with and reduce a lot of programming time. LINQ to SQL enables us to see the query and execute the result when debugging. This helps the code to be less error prone. It supports IntelliSense.

E. Using LINQ to access XML

Working with XML is not very easy. LINQ to XML is a new in-memory XML programming API that works against XML data. It provides query and transformation power of XQuery and XPath integrated into .NET Framework. XQuery is used to query XML data, XPath is used to navigate through elements and attributes in an XML document. XQuery is build on XPath.

Following is XQuery expression used to get titles of the books with price greater than 50 from books.xml file.

For $x in doc("books.xml")/bookstore/book where $x/price>50 return $x/title

LINQ to XML is a new method of creating and manipulating XML data. The properties and methods of LINQ help to navigate and manipulate XML elements and attributes. LINQ to XML component aims to simplify working with XML. It can be used to create, update and delete XML elements in an XML tree. LINQ to XML provides different query operators like projections, aggregates, partitioning, grouping and conversion.

Following is LINQ to XML expression to get titles of books with price greater than 50 from books.xml file.

var x = from r in books where r.Price > 50 select r;

LINQ is integrated with .Net languages where as XQuery requires a separate parser. LINQ is independent of data. It supports same syntax for objects, database access and XML. Hence LINQ is far more advantages than XQuery to query XML data.

LINQ to XML framework is based on set of classes. All these classes are prefixed with X. The following figure [Figure 2] shows LINQ to XML class Hierarchy.

![LINQ to XML class Hierarchy](image)

Each of LINQ to XML class is used for specific task or function. The above figure shows abstract classes and its derived classes. XObject abstracts both XNode and XAttribute classes. XObject provides annotation and event functionality. XNode represents nodes of an XML tree. XData, XText, XContainer, XComment and XPrecessingInstruction and XDocumentType are derived from XNode class. XContainer is an abstract class: XElement and XDocument are derived from it. XElement is the fundamental class in LINQ to XML. It is generally used to make XML trees. XElements are associated with name/value pairs of XAttributes. XML declaration is used to declare XML version and encoding of a document. It also specifies if an XML document is stand-alone or not. XName represents names of elements and XNamespace represents namespaces.

Following example shows how an XML file can be queried using LINQ to XML.

XML File:
<Tutorials>
  <Tutorial>
    <Author>The Reddest</Author>
    <Title>
      Creating an XP Style WPF Button with Silverlight
    </Title>
  </Tutorial>
</Tutorials>
LINQ to XML code:

```csharp
XDocument xmlDoc = XDocument.Load("TestFile.xml");

var tutorials = from tutorial in
    xmlDoc.Descendants("Tutorial")
    select new
    {
        Author = tutorial.Element("Author").Value,
        Title = tutorial.Element("Title").Value,
        Date = tutorial.Element("Date").Value,
    };

foreach (var tutorial in tutorials)
{
    Console.WriteLine("Author: " + tutorial.Author);
    Console.WriteLine("Title: " + tutorial.Title);
    Console.WriteLine("Author: " + tutorial.Date);
}
```

The output for the above code will be 8 and 10.

Traditionally we had to go through a lot of looping code to find required value, whereas using LINQ we can directly query collections and filter the required value. LINQ provides powerful filtering, ordering, and grouping capabilities as well [5]. LINQ to object queries are more concise and readable. They can be ported to other databases with minimum modifications.

### G. LINQ Implementation in Other Languages

LINQ is not just supported by C# and Visual Basic .Net alone. Microsoft’s new language F# and C++ also implements LINQ [6]. Microsoft is developing PLINQ for a parallel execution of LINQ queries. PLINQ is capable of implementing a query in a distributed manner on a multi core or a multi processor system [7]. Apart from Microsoft Languages, Language Integrated Query concept is also implemented in other languages. Saffron is an open source extension of Java that incorporates SQL-like relational expressions into Java. Relations can be database tables, in-memory collections or any other data sources. PHPLINQ is a set of PHP classes which extends LINQ features to PHP. Borland also supports LINQ on Delphi platform [7].

JLinq implements LINQ functionality in Javascript. Instead of using for loops and if statements, JLinq uses query style syntax in JavaScript. JLinql is also extensible. We can create new functions and plug them into library. Below is a sample JLINQ query [8].

```csharp
jlinq.from(data.users)
    .starts("first", "a")
    .or("b")
    .sort("age")
    .select();
```

JaQu stands for Java Query, it implements LINQ for Java. JaQu can be easily integrated with Java applications unlike SQL. JaQu can replace JDBC in Java. It supports auto complete feature in IDE. Following is a sample JaQu query [9]

```java
SQL Statement

Select * from Products P

Equivalent JaQu Query

Product p = new Product();

List<Product> soldOutProducts =
    db.from(p).where(p.uintsInStock).is(0).select;
```
III. METHODOLOGY

H. Project Description and Development

In this project a Home Budget, an Electronics Store and a Hashtable application are developed. Each of these applications uses different data sources.

I. HOME BUDGET

Home Budget application is used to track income, expenditures and savings of a household. Having a budget is a way to manage finances efficiently. Home Budget application helps users to stay in budget and reduces the problem of overspending. It is developed on PostgreSQL database and queried using LINQ on SQL.

Below is the use case diagram [Figure 3] for Home Budget application. New users to Home Budget can create their accounts. Once an account is created, user can login and start using the application. They can add both income and expense transactions. Users can also track summary of their finances.

![Figure 3: UseCase Diagram of Home Budget Application](image)

The transactions of Home Budget are saved in PostgreSQL database. Three tables are created in the database–UserInfo, Income and Expense. UserInfo table is used to store user information. Income and expense table’s stores income and expenditure information respectively. The following figure [Figure 4] shows entity relationship diagram for HomeBudget database.

![Figure 4: Entity Relationship diagram of Home Budget Database](image)

User Interface of Home Budget application is designed in a simple and user friendly manner. The following figure [Figure 5] shows welcome screen of Home Budget application. New users can create their account by clicking on the “Create Account” button. Users are required to provide a unique user name for creating account. Once an account is created users can login to the application.

![Figure 5: Welcome Screen of Home Budget](image)

User is directed to Transaction screen from the login Screen. On the Transaction screen there are two tabs – one for adding transactions and the other for viewing summary of transactions. User can add the amount of income and expenditure incurred for any particular day on the ‘Add Transactions’ tab. Show summary tab is used to generate a report of income, expenditure and savings during a period of time.

![Figure 6: Transactions screen of Home Budget](image)
Technical Details

As Microsoft does not support PostgreSQL for LINQ I used DBLinq, a third party provider in Home Budget project. DBLinq supports LINQ on PostgreSQL database. DBLinq compiles all tables defined in HomeBudget database to .Net Common Language Runtime types that can be accessed through code. Financialmanagement.cs is an auto generated DBLinq class which defines UserInfo, Income and Expense classes with [Table] attributes. All columns are defined in the form of properties. Objects linked to UserInfo, Income and Expense tables can be defined and used just like normal objects with attributes.

Following is the LINQ code for looping through a collection of values in a table.

```csharp
FinancialManagement db = new FinancialManagement(connectionString);
foreach (UserInfo user in db.UserInfo)
{
    if (uname_textbox.Text == user.Uname)
    {
        temp++;
        MessageBox.Show("UserName already in use!", "Alert");
        break;
    }
}
```

An object ‘db’ is created for the Financial Management database class. Tables in the database can be accessed using the dot operator ‘db.UserInfo’. ‘user’ is an object of UserInfo class. user.Uname gives values in ‘UserName’ column in UserInfo table.

Following is the LINQ code for inserting values in a table.

```csharp
db.Income.InsertOnSubmit(new Income
    {
        Uname = FinancialManagement.Main.currentuser,
        Date = dateTimePicker.Value,
        Source = category_combo.Text,
        Amount = Amount_textbox.Text
    });
```

InsertOnSubmit() method is defined to insert values in a table. ‘db.Income’ represents ‘Income’ table in financial management database. Uname, date, source and Amount stands for columns in Income database.

Following is the LINQ code for selecting required information from database.

```csharp
var exp = from expense in db.Expense
    where expense.Uname == uname &&
    (expense.Date >= dateTimePicker1.Value ||
    expense.Date <= dateTimePicker2.Value)
    select expense;
foreach (var v in exp)
{
    totalexpense = Convert.ToInt16(v.Amount) + totalexpense;
}
```

var is implicit type local variable. var keyword instructs the compiler to infer the type of the variable from the expression on the right side of the initialization statement. In the above code ‘var’ provides functionality to evaluate queries against a specific data source where the type of data is unknown. ‘expense’ is an object of expense class. ‘exp’ gets a collection of values from expense table satisfying a condition. ‘v.Amount’ refers ‘Amount’ value in expense table.

J. ELECTRONICS STORE

Electronics Store application is used to track list of all electronic gadgets present in a store. The data is saved in an XML file. LINQ on XML component is used to query data from XML document in this application.

Below [Figure 8] is the usecase diagram for Electronics Store application. Users can view existing items in the store, add new items and remove items that are out of stock.
User Interface of Electronics Store application is designed in a user friendly manner. The following figure [Figure 9] shows the main screen of Electronics Store application. Clicking ‘Load Items’ displays all the existing electronic items in store. Each Item has ItemName, Brand and Cost associated with it. ‘Add Item’ is used to add new item to the list and ‘Remove Item’ is used to remove item from the list.

Technical Details

In Electronics Store application, an instance of ‘store’ object is converted to an XML file by using serialization. ‘store’ class contains a collection of items which are user defined. In this application LINQ to XML queries are used to navigate and manipulate XML elements.

Following is the LINQ code for loading XML elements

XDocument xmlDoc = XDocument.Load("XMLDocument.xml");
var items = from item in xmlDoc.Descendants("itemslist")
select new
{
    ItemName = item.Element("Name").Value,
    Brand = item.Element("Brand").Value,
    Cost = item.Element("Cost").Value,
};
outputDataGridView.DataSource = items.ToList();

‘XDocument’ is a LINQ to XML class which represents an XML document. XDocument.Load() is used to load the XMLDocument. ‘item’ is a range variable of type XElement. A range variable is like an iteration variable in a foreach statement but it never actually stores data from the source. XElement is a LINQ to XML class which represents XML elements.

Following is the LINQ code to add new XML elements

XDocument xmlDoc = XDocument.Load("XMLDocument.xml");
xmlDoc.Element("Store").Add(new
    XElement("itemslist", new XElement("Name",
        ItemNameTextBox.Text),
        new XElement("Brand", BrandTextBox.Text),
        new XElement("Cost", CostTextBox.Text)));
xmlDoc.Save("XMLDocument.xml");

‘xmlDoc.Element.Add()’ adds new XML elements to the existing XML Document. ‘xmlDoc.Save()’ method is used to save changes to XMLDocument.

Following is the LINQ code to search for an XML element and delete it

XDocument xmlDoc = XDocument.Load("XMLDocument.xml");
XElement currentitem = xmlDoc.Descendants("itemslist").Where(c =>
    c.Element("Name").Value.Equals(ItemName_textbox.Text).
    ToString()).FirstOrDefault();
if (currentitem != null)
{
    currentitem.Remove();
    xmlDoc.Save("XMLDocument.xml");
}

‘xmlDoc.Descendants()’ returns a collection of descendant elements for the xml document. ‘c=>’ is a lambda operator which reads as “goes to”. The left side of the lambda operator specifies the input parameters and the right side holds the expression or statement. Lambda operators are used in LINQ queries as arguments to standard query operators such as where. FirstOrDefault() method returns the first value in the series, or the value zero or null if there is no first element. ‘currentitem.Remove()’ is used to remove elements from XML Document.

K. HASHTABLE APPLICATION
Hashtable application is used to store a collection of key value pairs. LINQ for Objects is used to query these objects directly without going through a lot of looping code.

Below [Figure 10] is the use case diagram for Hashtable application. Users can view the available key-value pairs, add new entries, and remove existing values. They can also search for corresponding ‘Values’ by providing ‘Keys’.

![Use Case Diagram](image)

Figure 10: User Case diagram of Hashtable Application.

User Interface for Hashtable application is designed in a simple manner. The following figure shows the main screen of Hashtable application. ‘Show Record’ displays all the existing key-value pairs. ‘Add Record’ is used to add new key-values and ‘Delete Record’ is used to remove existing key-values. ‘Search’ is used to get ‘value’ for the corresponding ‘key’ provided.

![User Interface](image)

Figure 11: User Interface for Hashtable Application

IV. ANALYSIS AND CONCLUSIONS

To conclude, the following are the advantages and disadvantages observed while using LINQ on XML, databases and objects in this project. Developers need not know languages like C# and secondary language like SQL or XQuery to query data from datasources. As LINQ integrates query capabilities to C# language there is no need to know separate techniques to query relational database languages, XML data and object collections. Using LINQ similar syntax can be used to query different data sources. LNQ queries use less lines of code compared to non LINQ queries. Hence it can be concluded that LINQ reduces a lot of programming time. LINQ provides facility to do compile-time checking for syntax errors which was not available earlier. LINQ also provides debugging and intellisense support. LINQ to object queries are more concise and readable. I found complex queries are difficult to build using LINQ which is a disadvantage. To conclude, adding Language Integrated Query (LINQ) to .Net Framework is definitely a value addition to the language.

REFERENCES


[11] N. Satheesh Kumar ; reviewer, Granville Barnett; "LINQ quickly : a practical guide to programming language integrated query with C#"  


