LINQ (Language-Integrated Query) – Part 1

Agenda

- Introduction
- Background
- Tasks without LINQ
- Tasks with LINQ
- Examples of LINQ Builder Methods

Tools

- Visual Studio 2013

Pre-requisite

- Understanding of C# Basics

Version	Last updated	Comments	Modified By
V1.0	December 2018		Bilal Shahzad

LINQ (Language-Integrated Query) is a programming construct which came with .NET Framework 3.5. You don't need to install anything separately to use this.

You may use

1- **Query Approach:** In this approach we write "query" in C#.
2- **LINQ Builder Method Approach:** In this approach, we use methods instead of "clauses".

Background

We all know about DB Query Writing language (SQL). What about having a **query language** in programming? We know we can create arrays (or lists) of objects in programing, what If we want to write some query on such list in C#?

Sample Data:

```
8 references
class StudentDTO
{
    4 references
    public int ID { get; set; }
    4 references
    public String Name { get; set; }
}
```

We've created a list of different objects of above class

```
List<StudentDTO> list = new List<StudentDTO>();

StudentDTO dto = new StudentDTO();
dto.ID = 1;
dto.Name = "Khurram";

list.Add(dto);

StudentDTO dto1 = new StudentDTO();
dto1.ID = 2;
dto1.Name = "Faisal";

list.Add(dto1);

StudentDTO dto2 = new StudentDTO();
dto2.ID = 3;
dto2.Name = "Waqas";

list.Add(dto2);
```

Task 1: Write C# code to find a student who has ID =3

Answer: Here we are iterating our list and applying our conditioning

```
//Find those from list where ID = 3
for (int i = 0; i < list.Count; i++)
{
    if (list[i].ID == 3)
    {
        Console.WriteLine("Name is:{0}", list[i].Name);
        break;
    }
}
```

Task 2: Write C# code to find a student who has ID =3 & Name = “Faisal”

```
//Find those from list where ID = 3 AND Name ="Faisal"
for (int i = 0; i < list.Count; i++)
{
    if (list[i].ID == 3 && list[i].Name == "Faisal")
    {
        Console.WriteLine("Name is:{0}", list[i].Name);
        break;
    }
}
```

Task 3: Write C# code to find students who has ID =3 OR Name = “Faisal”

```
//Find those from list where ID = 3 OR Name ="Faisal"
for (int i = 0; i < list.Count; i++)
{
    if (list[i].ID == 3 || list[i].Name == "Faisal")
    {
        Console.WriteLine("Name is:{0}", list[i].Name);
    }
}
```

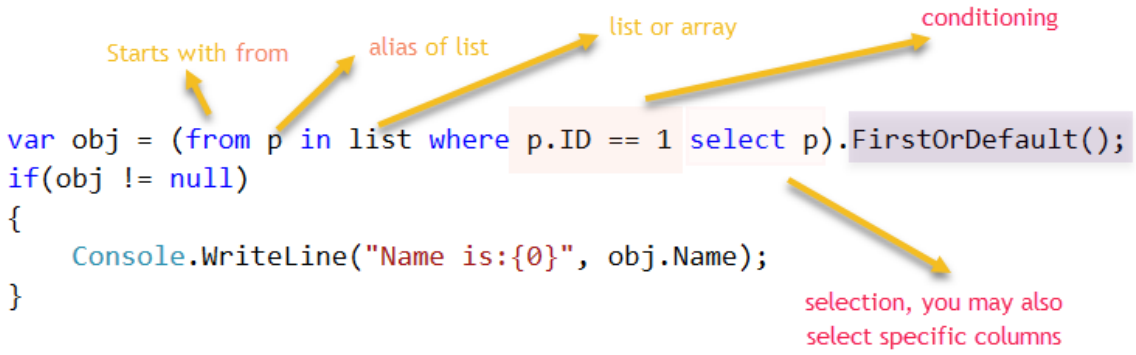
Question: Instead of iterating the list & finding the objects, can we write SQL like query in C# which will do all this iteration stuff behind the scene and give us result?

Something like following

Select * from list where ID = 3 OR Name = “Faisal”

Answer: Yes we’ve **LINQ**. It allows you to write your query in “**specific**” format and does the iteration stuff behind the scene. Syntax is little different than SQL.

Task 4: Using LINQ, Write C# code to find a student who has ID =3



In LINQ, we start our query with “from” and then give an alias to our “data source” (list in this case) then we use “in” keyword and then our source name.

After that we may have “where” section if there is any conditioning.

After that we may have “select” statement. Then if we want to do *, we may use alias. We may also pick specific columns.

This whole statement is just a “query”. You need to either “iterate” this query using “foreach” loop or you may call a method on it (e.g. FirstOrDefault, First, ToList()). There are different methods you may use according to your need. In above case, we know that we are going to get maximum one result so we’ve used FirstOrDefault (This will give one result or NULL if we query returns no result).

Task 5: Using LINQ, Write C# code to find a student who has ID =3 & Name = “Faisal”

```
var obj1 = (from p in list where (p.ID == 3 && p.Name == "Faisal")  select p).FirstOrDefault();
if (obj1 != null)
{
    Console.WriteLine("Name is:{0}", obj1.Name);
}
```

Task 6: Using LINQ, Write C# code to find students who has ID =3 OR Name = “Faisal”

Here result is List of StudentDTO

```
var result = (from p in list where (p.ID == 3 || p.Name == "Faisal") select p).ToList();
foreach (var item in result)
{
    Console.WriteLine("ID:{0}, Name is:{1}", item.ID, item.Name);
}
```

Some more examples

```
// Use of Orderby and multiline query
var result1 = (from p in list
               orderby p.ID descending
               select p).ToList();

//Select specific column
var result2 = (from p in list
               select p.ID).ToList();

//Select specific columns
var result3 = (from p in list
               select new { p.ID, p.Name }).ToList();
```

will give List of IDs

will give list of anonymous objects

Some examples of **LINQ Builder Method** Approach

```
//Find those from list where ID = 3
var r1 = list.Where(p => p.ID == 3).FirstOrDefault();

//Find those from list where ID = 3 AND Name ="Faisal"
var r2 = list.Where(p => p.ID == 3 && p.Name == "Faisal").FirstOrDefault();

//Find those from list where ID = 3 OR Name ="Faisal"
var r3 = list.Where(p => p.ID == 3 || p.Name == "Faisal").ToList();

//Order By
var r4 = list.OrderByDescending(p => p.ID).ToList();

//Select specific column
var r5 = list.Select(p => p.ID).ToList();

//Select specific columns
var r6 = list.Select(p => new { p.ID, p.Name }).ToList();

//Where & Select specific columns
var r7 = list.Where(p=> p.ID ==3).Select(p => new { p.ID, p.Name }).ToList();
```

Useful Links

- LINQ
  - Go through this link
    - http://stackoverflow.com/questions/16322/learning-about-linq
  - 101 Samples
    - https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b
    - http://linq101.nilzorblog.com/linq101-lambda.php
    - http://linqsamples.com/

Extra for Reading

- IEnumerator
    - Supports a simple iteration over a non-generic collection.
    - IEnumerator is the base interface for all non-generic enumerators.
    - The **foreach** statement of the C# language hides the complexity of the enumerators. Therefore, using **foreach** is recommended instead of directly manipulating the enumerator.
    - An enumerator remains valid as long as the collection remains unchanged.
    - For the generic version of this interface see IEnumerator<T>
    - https://msdn.microsoft.com/en-us/library/system.collections.ienumerator(v=vs.100).aspx

- IEnumerable
    - Exposes an enumerator, which supports a simple iteration over a non-generic collection.
    - IEnumerable is the base interface for all non-generic collections that can be enumerated. For the generic version of this interface see System.Collections.Generic.IEnumerable<T>
    - It is a best practice to implement IEnumerable and IEnumerator on your collection classes to enable the **foreach**
    - However implementing IEnumerable is not required but "GetEnumerator" method should be provided.
- IQueryable
    - Provides functionality to evaluate queries against a specific data source wherein the type of the data is not specified.
    - The IQueryable interface inherits the IEnumerable interface so that if it represents a query, the results of that query can be enumerated.

Enumeration causes the expression tree associated with an IQueryable object to be executed. The definition of "executing an expression tree" is specific to a query provider. For example, it may involve translating the expression tree to an appropriate query language for the underlying data source.