LINQ - OVERVIEW

Developers across the world have always encountered problems in querying data because of the lack of a defined path and need to master a multiple of technologies like SQL, Web Services, XQuery, etc.

Introduced in Visual Studio 2008 and designed by Anders Hejlsberg, LINQ LanguageIntegratedQuery allows writing queries even without the knowledge of query languages like SQL, XML etc. LINQ queries can be written for diverse data types.

Example of a LINQ query

C#

```csharp
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        string[] words = {"hello", "wonderful", "LINQ", "beautiful", "world"};
        //Get only short words
        var shortWords = from word in words
                         where word.Length <= 5
                         select word;
        //Print each word out
        foreach (var word in shortWords)
        {
            Console.WriteLine(word);
        }
        Console.ReadLine();
    }
}
```

VB

```vbnet
Module Module1
    Sub Main()
        Dim words As String() = {"hello", "wonderful", "LINQ", "beautiful", "world"}
        ' Get only short words
        Dim shortWords = From word In words
                         Where word.Length <= 5
                         Select word
        ' Print each word out.
        For Each word In shortWords
            Console.WriteLine(word)
        Next
        Console.ReadLine()
    End Sub
End Module
```

When the above code of C# or VB is compiled and executed, it produces the following result:

```
hello
LINQ
world
```
**Syntax of LINQ**

There are two syntaxes of LINQ. These are the following ones.

- **Lambda Method Syntax**

  Example

  ```csharp
  var longWords = words.Where( w => w.length > 10);
  Dim longWords = words.Where(Function(w) w.length > 10)
  ```

- **Query Comprehension Syntax**

  Example

  ```csharp
  var longwords = from w in words where w.length > 10;
  Dim longwords = from w in words where w.length > 10
  ```

**Types of LINQ**

The types of LINQ are mentioned below in brief.

- LINQ to Objects
- LINQ to XML
- LINQ to DataSet
- LINQ to SQL
- LINQ to Entities

Apart from the above, there is also a LINQ type named PLINQ which is Microsoft’s parallel LINQ.

**LINQ Architecture in .NET**

LINQ has a 3-layered architecture in which the uppermost layer consists of the language extensions and the bottom layer consists of data sources that are typically objects implementing `IEnumerable<T>` or `IQueryable<T>` generic interfaces. The architecture is shown below in Figure.
Query Expressions

Query expression is nothing but a LINQ query, expressed in a form similar to that of SQL with query operators like Select, Where and OrderBy. Query expressions usually start with the keyword “From”.

To access standard LINQ query operators, the namespace System.Query should be imported by default. These expressions are written within a declarative query syntax which was C# 3.0.

Below is an example to show a complete query operation which consists of data source creation, query expression definition and query execution.

C#

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Operators
{
    class LINQQueryExpressions
    {
        static void Main()
        {
            // Specify the data source.
            int[] scores = new int[] { 97, 92, 81, 60 };

            // Define the query expression.
            IEnumerable<int> scoreQuery = from score in scores
                                           where score > 80
                                           select score;

            // Execute the query.
            foreach (int i in scoreQuery)
            {
                Console.Write(i + " ");
            }
            Console.ReadLine();
        }
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
97 92 81
```

Extension Methods

Introduced with .NET 3.5, Extension methods are declared in static classes only and allow inclusion of custom methods to objects to perform some precise query operations to extend a class without being an actual member of that class. These can be overloaded also.

In a nutshell, extension methods are used to translate query expressions into traditional method calls object-oriented.

Difference between LINQ and Stored Procedure

There is an array of differences existing between LINQ and Stored procedures. These differences are mentioned below.

- Stored procedures are much faster than a LINQ query as they follow an expected execution plan.
- It is easy to avoid run-time errors while executing a LINQ query than in comparison to a stored procedure as the former has Visual Studio’s Intellisense support as well as full-type
• LINQ allows debugging by making use of .NET debugger which is not in case of stored procedures.

• LINQ offers support for multiple databases in contrast to stored procedures, where it is essential to re-write the code for diverse types of databases.

• Deployment of LINQ based solution is easy and simple in comparison to deployment of a set of stored procedures.

Need For LINQ

Prior to LINQ, it was essential to learn C#, SQL, and various APIs that bind together the both to form a complete application. Since, these data sources and programming languages face an impedance mismatch; a need of short coding is felt.

Below is an example of how many diverse techniques were used by the developers while querying a data before the advent of LINQ.

```csharp
SqlConnection sqlConnection = new SqlConnection(connectString);
sqlConnection.Open();
sqlCommand.Connection = sqlConnection;
sqlCommand.CommandText = "Select * from Customer";
return sqlCommand.ExecuteReader(CommandBehavior.CloseConnection);
```

Interestingly, out of the featured code lines, query gets defined only by the last two. Using LINQ, the same data query can be written in a readable color-coded form like the following one mentioned below that too in a very less time.

```csharp
Northwind db = new Northwind("C:\Data\Northwnd.mdf");
var query = from c in db.Customers
            select c;
```

Advantages of LINQ

LINQ offers a host of advantages and among them the foremost is its powerful expressiveness which enables developers to express declaratively. Some of the other advantages of LINQ are given below.

• LINQ offers syntax highlighting that proves helpful to find out mistakes during design time.

• LINQ offers IntelliSense which means writing more accurate queries easily.

• Writing codes is quite faster in LINQ and thus development time also gets reduced significantly.

• LINQ makes easy debugging due to its integration in the C# language.

• Viewing relationship between two tables is easy with LINQ due to its hierarchical feature and this enables composing queries joining multiple tables in less time.

• LINQ allows usage of a single LINQ syntax while querying many diverse data sources and this is mainly because of its unitive foundation.

• LINQ is extensible that means it is possible to use knowledge of LINQ to querying new data source types.

• LINQ offers the facility of joining several data sources in a single query as well as breaking complex problems into a set of short queries easy to debug.

• LINQ offers easy transformation for conversion of one data type to another like transforming SQL data to XML data.