Linq Interview
Questions & Answers

By Shailendra Chauhan
Microsoft MVP, Founder & CEO - Dot Net Tricks

DotNetTricks

LINQ Interview Questions & Answers

Release History

- Initial Release 1.0.0 - 1st Jul 2014
- Second Release 1.0.1 - 1st Jan 2015
- Third Release 1.0.2 - 12th Aug 2016
- Fourth Release 1.1 - 19th Nov 2018
- Fifth Release 1.2 - 4th Jan 2019

About Dot Net Tricks

Dot Net Tricks is founded by Shailendra Chauhan (Microsoft MVP), in Jan 2010. Dot Net Tricks came into existence in form of a blog post over various technologies including .NET, C#, SQL Server, ASP.NET, ASP.NET MVC, JavaScript, Angular, Node.js and Visual Studio etc.

The company which is currently registered by a name of Dot Net Tricks Innovation Pvt. Ltd. came into the shape in 2015. Dot Net Tricks website has an average footfall on the tune of 300k+ per month. The site has become a cornerstone when it comes to getting skilled-up on .NET technologies and we want to gain the same level of trust in other technologies. This is what we are striving for.

We have a very large number of trainees who have received training from our platforms and immediately got placement in some of the reputed firms testifying our claims of providing quality training. The website offers you a variety of free study material in form of articles.

Dot Net Tricks Training Solutions

Dot Net Tricks provide you training in traditional as well as new age technologies, via various formats.

Master Courses (Instructor-led)

For a beginner who needs regular guidance, we have a fully packed Master Courses. They are almost equal to semester courses taught in engineering colleges when it comes to length, breadth of content delivery, the only difference instead of 5-6 months, they take approx. 16-weekend classes (2 months).

The detail about Master courses can be found here: https://www.dotnettricks.com/instructor-led-courses

Hands-On Learning (Learn to code)

Hands-On Learning courses give you the confidence to code and equally helpful to work in real-life scenarios. This course is composed of hands-on exercise using IDE or cloud labs so that you can practice each and everything by yourself. You can learn to code at your own pace, time and place.

The detail about Hands-On Learning courses can be found here: https://www.scholarhat.com

Skill Bootcamps (Instructor-led)

Professionals who don't have two months' time and want to get skilled up in least possible time due to some new project that their company has to take in very near future, we have designed Skill Bootcamps Concept, where you will get trained on consecutive days in a fast-paced manner, where our full focus is going to be on hands-on delivery of technological exercises.

The detail about Skill Bootcamps can be found here: https://www.dotnettricks.com/skill-bootcamp

Self-paced Courses

Self-paced courses give you the liberty to study at your own pace, time and place. We understand everyone has their own comfort zone, some of you can afford to dedicate 2 hours a day, some of you not. Keeping this thing in mind, we created these self-paced courses. While creating these courses we have ensured that quality of courses doesn't get compromise at any parameter, and they also will be able to produce the same results as our other course formats, given the fact you will be able to put your own honest effort.

The detail about Self-paced courses can be found here: https://www.dotnettricks.com/self-paced-courses

Corporate Training (Online and Classroom)

Dot Net Tricks having a pool of mentors who help the corporate to enhance their employment skills as per changing technology landscape. Dot Net Tricks offers customized training programs for new hires and experienced employees through online and classroom mode. As a trusted and resourceful training partner, Dot Net Tricks helps the corporate to achieve success with its industry-leading instructional design and customer training initiatives.

The detail about Corporate Training can be found here: https://www.dotnettricks.com/corporate-training

Learning Platform

We have a very robust technology platform to answer the needs of all our trainees, no matter which program they enrolled in. We have a very self-intuitive Learning Management System (LMS), which help you in remain focused and keeping an eye over your progress.

We offer two Learning Platforms as given below:

1. Dot Net Tricks: https://www.dotnettricks.com
2. Scholar Hat: https://www.scholarhat.com

Apart from these, we also provide on-demand Skill bootcamps and personalized project consultation.

Dedication

My mother Mrs Vriksha Devi and my wife Reshu Chauhan deserve to have their name on the cover as much as I do for all their support made this possible. I would like to say thanks to all my family members Virendra Singh(father), Jaishree and Jyoti(sisters), Saksham and Pranay(sons), friends, to you and to readers or followers of my articles at https://www.dotnettricks.com/mentors/shailendra-chauhan to encourage me to write this book.

I would like to give a special thanks to Kanishk Puri and Avadhesh Sengar for actively participating in the feedback and contributions for this book via their valuable feedback and suggestions.

-Shailendra Chauhan

Introduction

Writing a book has never been an easy task. It takes a great effort, patience and consistency with strong determination to complete it. Also, one should have a depth knowledge over the subject is going to write.

So, what where my qualification to write this book? My qualification and inspiration come from my enthusiasm for and the experience with the technology and from my analytic and initiative nature. Being a trainer, analyst, consultant and blogger, I have thorough knowledge and understandings of .NET technologies. My inspiration and knowledge have also come from many years of my working experience and research over it.

So, the next question is who this book is for? This book is appropriate for the novice as well as for senior-level professionals who want to understand what LINQ does, how it does in .NET languages like C# and VB. This book is equally helpful to show you the best of using LINQ with the help of many practical ways to make your daily programming life easier and more productive.

This book does not only help you to learn LINQ but it also is helpful to learn Entity Framework. This book helps you to do hands-on LINQ as well as preparing yourself for an interview on LINQ.

I hope you will enjoy this book and find it useful. At the same time, I also encourage you to become a continuing reader of my blog, www.dotnettricks.com and be the part of the discussion. But most importantly practice a lot and enjoy the technology. That's what it's all about.

To get the latest information on LINQ, I encourage you to follow the MSDN at http://msdn.microsoft.com/en-us/library/bb397926.aspx. I also encourage you to subscribe to my blog at www.dotnettricks.com that contains .NET, C#, ASP.NET MVC, LINQ, Entity Framework, jQuery and many more tips, tricks and tutorials.

As the author, I am absolutely delighted about this. It's been a work of love and I want it to reach as many techy people as possible.

All the best for your interview and happy programming!

About the Author

Shailendra Chauhan - An Entrepreneur, Author, Architect, Corporate Trainer, and Microsoft MVP

He is the **Founder and CEO** of DotNetTricks which is a brand when it comes to e-Learning. DotNetTricks provides training and consultation over an array of technologies like **Cloud, .NET, Angular, React, Node and Mobile Apps development**. He has been awarded as **Microsoft MVP** three times in a row (2016-2018).

He has changed many lives from his writings and unique training programs. He has a number of most sought-after books to his name which have helped job aspirants in **cracking tough interviews** with ease.

Moreover, and to his credit, he has delivered **1000+ training sessions** to professionals worldwide in Microsoft .NET technologies and other technologies including JavaScript, AngularJS, Node.js, React and NoSQL Databases. In addition, he provides **Instructor-led online training**, **hands-on workshop** and **corporate training** programs.

Shailendra has a strong combination of **technical skills and solution development for complex application architecture with proven leadership and motivational skills** have elevated him to a world-renowned status, placing him at the top of the list of most sought-after trainers.

"I always keep up with new technologies and learning new skills to deliver the best to my students," says Shailendra Chauhan, he goes on to acknowledge that the betterment of his followers and enabling his students to realize their goals are his prime objective and a great source of motivation and satisfaction.

Shailendra Chauhan - **"Follow me and you too will have the key that opens the door to success"**

How to Contact Us

Although the author of this book has tried to make this book as accurate as it possible but if there is something strikes you as odd, or you find an error in the book please drop a line via e-mail.

The e-mail addresses are listed as follows:

- mentor@dotnettricks.com
- info@dotnettricks.com

We always happy to hear from our readers. Please provide your valuable feedback and comments!

You can follow us on YouTube, Facebook, Twitter, LinkedIn and Google Plus or subscribe to RSS feed.

Table of Contents

1
LINQ Introduction

Q1. What is LINQ and why to use it?

Ans. LINQ stands for "**Language Integrated Query**" and pronounced as "**LINK**". LINQ was introduced with .NET Framework 3.5 including Visual Studio 2008, C# 3.0 and VB.NET 2008 (VB 9.0). It enables you to query the data from the various data sources like SQL databases, XML documents, ADO.NET Datasets, Web services and any other objects such as Collections, Generics etc. by using a **SQL Query like syntax** with .NET framework languages like C# and VB.NET.

Why LINQ

LINQ has full **type checking** at compile-time and **IntelliSense** support in Visual Studio since it used the .NET framework languages like C# and VB.NET. This powerful feature helps you to avoid run-time errors.

LINQ also provides a **uniform programming model** (i.e. common query syntax) to query various data sources. Hence you don't need to learn different ways to query different data sources.

Q2. Which namespace is necessary to use LINQ?

Ans. **System.Linq** namespace is necessary for writing LINQ queries and to implement it.

Q3. What are different flavours of LINQ?

Ans. There are following three flavours of LINQ:

1. **LINQ to Objects**
   It enables you to query any in-memory object like as array, collection and generics types. It offers a new way to query objects with many powerful features like filtering, ordering and grouping with minimum code.

2. **LINQ to ADO.NET**
   LINQ to ADO.NET is used to query data from different databases like as SQL Server, Oracle, and others. Further, it can be divided into three flavours:-

   I. **LINQ to SQL (DLINQ)**
   It is specifically designed to work with only SQL Server database. It is an object-relational mapping (**ORM**) framework that allows **1-1 mapping** of SQL Server database to **.NET Classes**. These classes are automatically created by the wizard based on the database table and we can use these classes immediately.

II. **LINQ to Datasets**

It is an easy and faster way to query data **cached** in a **Dataset object**. This allows you to do further data manipulation operations (like searching, filtering, sorting) on Dataset using LINQ Syntax. It can be used to query and manipulate any database (like Oracle, MySQL, DB2 etc.) that can be queried with ADO.NET.

III. **LINQ to Entities**

In many ways, it looks like LINQ to SQL. It is an object-relational mapping (**ORM**) framework that allows **1-1 mapping**, **1-many mapping** and **many-many mapping** of a database to .NET Classes. Unlike LINQ to SQL, it can be used to query any database (like Oracle, MySQL, and DB2 etc.) including SQL Server. Now, it is called ADO.NET Entity Framework.

3. **LINQ to XML (XLINQ)**

This allows you to do different operations on XML data source like querying or reading, modifying, manipulating, and saving changes to XML documents. **System.Xml.Linq** namespace contains classes for LINQ to XML.

4. **Parallel LINQ (PLINQ)**

PLINQ was introduced with .NET Framework 4.0. It is a parallel implementation of LINQ to Objects. PLINQ use the power of parallel programming which targets the Task Parallel Library. PLINQ helps you to write a LINQ query which will be executed simultaneously or parallel on different processors.

Q4. What are advantages of LINQ?

Ans. There are following advantages of using LINQ:

1. It provides a uniform programming model (i.e. common query syntax) to query data sources (like SQL databases, XML documents, ADO.NET Datasets, Various Web services and any other objects such as Collections, Generics etc.)
2. It has full **type checking** at compile-time and **IntelliSense** support in Visual Studio. This powerful feature helps you to avoid run-time errors.
3. It supports various powerful features like filtering, ordering and grouping with minimum code.
4. Its Query can be reused.
5. It also allows debugging through the .NET debugger.

Q5. What are the disadvantages of LINQ?

Ans. There are following disadvantages of using LINQ:

1. LINQ is not good to write complex queries like SQL.
2. LINQ doesn't take the full advantage of SQL features like cached execution plan for the stored procedure.
3. Performance is degraded if you don't write the LINQ query correctly.
4. If you have done some changes in your query, you have to recompile it and redeploy its dll to the server.

Q6. What are different methods to write LINQ Query?

Ans. There are following three ways to write LINQ Query:

1. **Query Expression (Query Syntax)**

   Query expression syntax is like as SQL query syntax with just a few minor deviations. The result of a query expression is a query object, which is usually a collection of type IEnumerable<T> or IQueryable<T>. This syntax is easy to read and write and at compile time, the query expression is converted into Method Invocation.

   Query Expression Syntax:

```
from     [identifier]
in       [source collection]
let      [expression]
where    [boolean expression]
order by [expression(ascending/descending)]
select   [expression]
group    [expression] by [expression]
into     [expression]
```

   Query Expression Example:

```
// Datasource
List<int> numbers = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Query based syntax
IEnumerable<int> query =
         from num in numbers
         where num > 5 && num < 10
         select num; //result: 6,7,8,9
```

2. **Method Invocation (Method Syntax)**

   Method syntax is complex as compared to Query expression since it uses a lambda expression to write LINQ query. It is easily understood by .NET CLR. Hence at compile time, the Query expression is converted into Method Invocation. The result of a Method syntax is also a query object, which is usually a collection of type IEnumerable<T> or IQueryable<T>.

   Method Invocation Syntax:

```
[source collection]
.Where   [boolean expression]
.OrderBy [expression(ascending/descending)]
.Select  [expression]
.GroupBy [expression]
```

   Method Invocation Example:

```
// Datasource
List<int> numbers = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Method based syntax
```

```
IEnumerable<int> query = numbers.Where(num => num > 5 && num < 10);
//result: 6,7,8,9
```

3. **Mixed Syntax**

   You can use a mixture of both syntax by enclosing a query expression inside parentheses and make a call to method.

```
// Datasource
List<int> numbers = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Mixed syntax
int query = (from num in numbers
             where num > 5 && num < 10
             select num).Count();
//result: 4
```

Note: There are no semantic differences (in terms of performance, execution) between Query Syntax and Method Syntax.

Q7. How var type is different from anonymous type?

Ans. **var-** var data type was introduced with C# 3.0. It is used to declare implicitly typed local variable means it tells the compiler to figure out the type of the variable at compile time. A var variable must be initialized at the time of declaration.

For Example:

```
var i = 20; // implicitly typed
int i = 20; //explicitly typed

var str = "1"; //declaration and initialization
var num = 0;

string s = "string"; //explicitly typed
var s2 = s; // implicitly typed
```

Basically, var is an anonymous type, hence use it whenever you don't know the type of output or it is anonymous. Suppose you are joining two tables and retrieving data from both the tables then the result will be an anonymous type since data will come from both the tables.

Expression assigned to var must be executed at compile time to know the type of output, so that var type may behave like the new type assigned to it.

```
DataContext context = new DataContext();
var q = (from e in context.Employee
         join d in context.Department on e.DeptID equals d.DeptID
         select new
         {
            e.EmpID,
            e.FirstName,
```

```
            d.DeptName,
            d.DeptLocation
        });
```

Anonymous Type- An anonymous type is a simple class generated by the compiler within IL to store a set of values. **var** data type and new keyword is used to create an anonymous type.

```
var emp = new { Name = "Deepak", Address = "Noida", Salary = 21000 };
```

At compile time, the compiler will create an anonymous type, as follows:

```
class __Anonymous1
 {
       private string name;
       private string address;
       int salary;
       public string Name
       {
          get { return name; }
          set { name=value }
       }
       public string Address
       {
          get { return address; }
          set { address = value; }
       }
       public int Salary
       {
          get { return salary; }
          set { salary = value; }
       }
   }
```

The anonymous type is very useful when you want to shape the result in your desired form like this:

```
DataContext context = new DataContext();
var result = from book in context.Books
             where book.Price > 200
             orderby book.IssueDate descending
             select new
             {
                Name = book.Name,
                IssueNumber = "#" + book.Issue
             };
```

In above example, I change the name of the “Issue” field of Book table to “IssueNumber” and add # before the value to get the desired output.

Q8. What is the anonymous method?

Ans. The concept of the anonymous method was introduced in C# 2.0. An anonymous method is an inline unnamed method in the code. It is created using the **delegate keyword** and doesn't have its name and **return type**.

In this way, an anonymous method has no name, optional parameters and return type; it has the only body. An anonymous method behaves like a regular method and allows you to write inline code in place of regular method.

```
class Program
{
    //delegate for representing an anonymous method
    delegate int del(int x, int y);

    static void Main(string[] args)
    {
        //anonymous method using delegate keyword
        del d1 = delegate(int x, int y) { return x * y; };

        int z1 = d1(2, 3);

        Console.WriteLine(z1);
    }
}
//output:

6
```

Key points about the anonymous method

1. A variable declared outside the anonymous method can be accessed inside the anonymous method.
2. A variable declared inside the anonymous method can't be accessed outside the anonymous method.
3. We use the anonymous method in event handling.
4. An anonymous method declared without parenthesis can be assigned to a delegate with any signature.
5. Unsafe code can't be accessed within an anonymous method.
6. An anonymous method can't access the ref or out parameters of an outer scope.

Q9. What is lambda expression?

Ans. The concept of lambda expression was introduced in C# 3.0. Typically, a lambda expression is a more concise syntax of the anonymous method. It is just a new way to write an anonymous method. At compile time all the lambda expressions are converted into anonymous methods according to lambda expression conversion rules. The left side of the lambda operator "=>" represents the arguments of the anonymous method and the right side represents the method body.

Lambda expression Syntax

```
(Input-parameters) => expression-or-statement-block
```

Types of Lambda expression

1. **Statement Lambda -**Statement lambda has a statement block on the right side of the lambda operator "=>".

```
x => { return x * x; };
```

2. **Expression Lambda -** Expression lambda has only an expression (no return statement or curly braces), on the right side of the lambda operator "=>".

```
x => x * x; // here x*x is expression
```

Anonymous Method and Lambda Expression example:

```
class Program
{
    //delegate for representing an anonymous method
    delegate int del(int x, int y);

    static void Main(string[] args)
    {
        //anonymous method using expression lambda
        del d1 = (x, y) => x * y;
        // or (int x, int y) => x * y;

        //anonymous method using statement lambda
        del d2 = (x, y) => { return x * y; };
        // or (int x, int y) => { return x * y; };

        //anonymous method using delegate keyword
        del d3 = delegate(int x, int y) { return x * y; };

        int z1 = d1(2, 3);
        int z2 = d2(3, 3);
        int z3 = d3(4, 3);

        Console.WriteLine(z1);
        Console.WriteLine(z2);
        Console.WriteLine(z3);
    }
}
//output:
6
9
12
```

Q10. What is the Extension method?

Ans. An extension method is a static method of a static class that can be invoked like as an instance method syntax. Extension methods are used to **add new behaviours to an existing type** without altering.

In the extension method "**this**" keyword is used with the first parameter and the first parameter will be of a **type** that is extended by the extension method. Other parameters of extensions types can be of any types (data type).

Example:

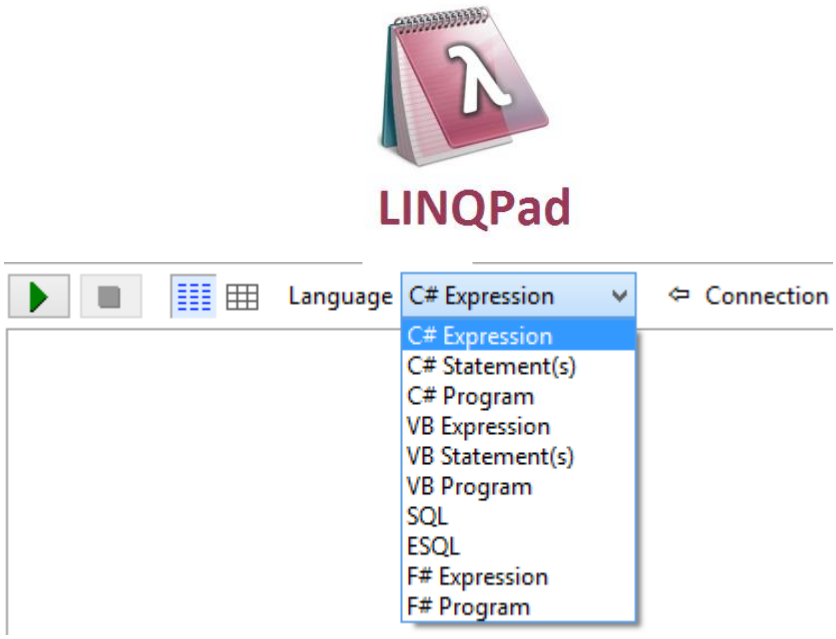
```
//defining extension method
public static class MyExtensions
{
    public static int WordCount(this String str)
    {
        return str.Split(new char[] { ' ', '.', ',' }).Length;
    }
}

class Program
{
    public static void Main()
    {
        string s = "Dot Net Tricks Extension Method Example";

        //calling extension method
        int i = s.WordCount();

        Console.WriteLine(i);
    }
}

//output:
//6
```

Key points about the extension method

1. An extension method is defined as a static method but it is called like as an instance method.
2. An extension method first parameter specifies the type of the extended object, and it is preceded by the *"this"* keyword.
3. An extension method having the same name and signature like as an instance method will never be called since it has low priority than an instance method.
4. An extension method couldn't override the existing instance methods.
5. An extension method cannot be used with fields, properties or events.
6. The compiler doesn't cause an error if two extension methods with the same name and signature are defined in two different namespaces and these namespaces are included in a same class file using directives. A compiler will cause an error if you will try to call one of the extension method.

Q11. What is LINQPad?

Ans. LINQPad is a most popular tool for testing LINQ queries. It is a standalone application developed by *Joseph Albahari* that allows you to execute LINQ queries or other C#, VB, F#, T-SQL statements.



It can be downloaded at from www.linqpad.net. It can be used without visual studio.

Q12. What LINQ providers are supported by LINQPad?

Ans. LINQPad supports Entity Framework, LINQ to SQL, WCF Data Services, and Microsoft DataMarket Service out-of-the-box. It can also be used with others third-party drivers to support most of the .NET ORMs.

Q13. What is LINQ Insight?

Ans. It is an alternative to LINQPad which is developed by Devart. It provides viewing of SQL, generated for LINQ to Entities, LINQ to NHibernate, LINQ to SQL, and LinqConnect, and profiling data access events for Entity Framework, NHibernate, LinqConnect, and LINQ to SQL. To test your LINQ queries it requires Visual Studio 2010, Visual Studio 2012, or Visual Studio 2013.

2
LINQ Operators

Q1. What are different types of operators in LINQ?

Ans. LINQ provides you, various operators, to write a LINQ query. Various types of operators are given below:

Operator Type	Operator Name
Aggregate	Aggregate, Average, Count, LongCount, Max, Min, Sum
Concatenation	Concat
Conversions	Cast, OfType, ToList, ToArray, ToLookup, ToDictionary, AsEnumerable
Element	Single, SingleOrDefault, First, FirstOrDefault, Last, LastOrDefault, ElementAt, ElementAtOrDefault
Equality	SequenceEqual
Generation	Repeat, Range, Empty
Grouping	GroupBy
Join	Join, GroupJoin
Ordering	OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
Partitioning	Skip, SkipWhile, Take, TakeWhile,
Projection	Select, SelectMany
Quantifier	Contains, All, Any
Restriction	Where
Set	Union, Intersect, Except, Distinct

Q2. What are Quantifiers?

Ans. LINQ extension methods which return a **boolean** value are called as Quantifiers. Some of them are as:

1. All()
2. Any()
3. Contains()
4. SequenceEqual()

Q3. What are aggregate operators?

Ans. Aggregate operators perform calculations on a set of values and return a single value. The aggregate operators are Average, Count, Max, Min, and Sum.

```
List<int> data = new List<int> { 10, 20, 30, 40, 50 };

//Sum of numbers
```

```
Console.WriteLine(data.Sum());   //result:150

//Average of numbers
Console.WriteLine(data.Average()); //result:30

//Max of numbers
Console.WriteLine(data.Max());  //result:50
```

Q4. What are conversion operators?

Ans. Conversion operators convert the type of input objects into a specific type or sequence. The aggregate operators are AsEnumerable, Cast, ToList, ToArray and ToDictionary.

```
List<int> data = new List<int> { 10, 20, 30, 40, 50 };

//Cast to Array
int[] strArray = data.Cast<int>().ToArray(); //result:10 20 30 40 50
for (int i = 0; i < strArray.Length; i++)
{
   Console.Write(strArray[i] + " ");
}
```

Q5. What are element operators?

Ans. Element operators return a single element or a specific element from a collection. The elements operators are Single, SingleOrDefault, First, FirstOrDefault, Last, LastOrDefault.

```
List<int> data = new List<int> { 10, 20, 30, 40, 50 };

//Try to get element at specified position
Console.WriteLine(data.ElementAt(1));   //result:20

//Try to get element at specified position if exist, else returns default value
Console.WriteLine(data.ElementAtOrDefault(10));  //result:0, since default value
is 0

Console.WriteLine(data.First());  //result:10
Console.WriteLine(data.Last());  //result:50

//try to get first element from matching elements collection
Console.WriteLine(data.First(d => d <= 20));  //result:10

//try to get first element from matching elements collection else returns default
value
Console.WriteLine(data.SingleOrDefault(d => d >= 100));  //result:0, since
default value is 0

//Try to get single element
// data.Single();  //Exception:Sequence contains more than one element

//Try to get single element if exist otherwise returns default value
```

```
//  data.SingleOrDefault();  //Exception:Sequence contains more than one element

//try to get single element 10 if exist
Console.WriteLine(data.Single(d => d == 10));  //result:10

//try to get single element 100 if exist otherwise returns default value
Console.WriteLine(data.SingleOrDefault(d => d == 100));  //result:0, since
default value is 0
```

Q6. What is a difference among Single, SingleOrDefault, First and FirstOrDefault?

Ans. **Single -** It returns a single specific element from a collection of elements if the element match found. An exception is thrown, if none or more than one match found for that element in the collection.

SingleOrDefault - It returns a single specific element from a collection of elements if the element match found. An exception is thrown if more than one match found for that element in the collection. A default value is returned, if no match is found for that element in the collection.

First - It returns the first specific element from a collection of elements if one or more than one match found for that element. An exception is thrown, if no match is found for that element in the collection.

FirstOrDefault - It returns the first specific element from a collection of elements if one or more than one match found for that element. A default value is returned, if no match is found for that element in the collection.

Q7. When to use Single, SingleOrDefault, First and FirstOrDefault?

Ans. You should take care of following points while choosing Single, SingleOrDefault, First and FirstOrDefault.

1. When you want an exception to be thrown if the result set contains many records, use **Single** or **SingleOrDefault**.
2. When you want a default value is returned if the result set contains no record, use **SingleOrDefault**.
3. When you always want one record no matter what the result set contains, use **First** or **FirstOrDefault**.
4. When you want a default value if the result set contains no record, use **FirstOrDefault**.

Q8. Which one is fast between SingleOrDefault and FirstOrDefault?

Ans. **FirstOrDefault** usually performs faster as compared **SingleOrDefault**, since these iterate the collection until they find the first match. While **SingleOrDefault** iterates the whole collection to find one single match.

Q9. What is LINQ provider and what are different types of LINQ providers?

Ans. A LINQ provider is a library which helps you to execute a given query against a data source. The different types of LINQ providers are LINQ to Objects, LINQ to SQL, LINQ to XML LINQ to Datasets and LINQ to Entities.

Q10. What are advantages of using LINQ on DataSet?

Ans. LINQ to DataSet is useful to run **strongly typed** queries on multiple datasets. SQL query is able to populate the dataset from the database but not able to retrieve a particular value from the dataset. LINQ is the best way to do further data manipulation operations (like searching, filtering, sorting) on Dataset in an efficient way.

Q11. What is the difference between Select and SelectMany in LINQ?

Ans. Select and SelectMany are projection operators. The select operator is used to selecting a value from a collection and the SelectMany operator is used to selecting values from a collection of collection i.e. **nested collection**.

Select operator produces one result value for every source value while **SelectMany** produces a single result that contains a concatenated value for every source value. Actually, **SelectMany** operator flatten **IEnumerable<IEnumerable<T>> to IEnumrable<T>** i.e**. list of list to list**.

```
class Employee
{
    public string Name { get; set; }
    public List<string> Skills { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
        List<Employee> employees = new List<Employee>();
        Employee emp1 = new Employee { Name = "Deepak", Skills = new List<string>
{ "C", "C++", "Java" } };
        Employee emp2 = new Employee { Name = "Karan", Skills = new List<string> {
"SQL Server", "C#", "ASP.NET" } };

        Employee emp3 = new Employee { Name = "Lalit", Skills = new List<string> {
"C#", "ASP.NET MVC", "Windows Azure", "SQL Server" } };

        employees.Add(emp1);
        employees.Add(emp2);
        employees.Add(emp3);

        // Query using Select()
        IEnumerable<List<String>> resultSelect = employees.Select(e=> e.Skills);

        Console.WriteLine("Using Select():\n");

        // Two foreach loops are required to iterate through the results
        // because the query returns a collection of arrays.
        foreach (List<String> skillList in resultSelect)
        {
            foreach (string skill in skillList)
            {
                Console.WriteLine(skill);
            }
            Console.WriteLine();
        }

        // Query using SelectMany()
```

```
        IEnumerable<string> resultSelectMany = employees.SelectMany(emp =>
emp.Skills);

        Console.WriteLine("Using SelectMany():");

        // Only one foreach loop is required to iterate through the results
        // since query returns a one-dimensional collection.
        foreach (string skill in resultSelectMany)
        {
            Console.WriteLine(skill);
        }

        Console.ReadKey();
    }
}

/*Output :

Using Select():

C
C++
Java

SQL Server
C#
ASP.NET

C#
ASP.NET MVC
Windows Azure
SQL Server

Using SelectMany():

C
C++
Java
SQL Server
C#
ASP.NET
C#
ASP.NET MVC
Windows Azure
SQL Server
*/
```

Q12. What is a difference among Any, All and Contains?

Ans. **Any -** It checks whether at least one element in a collection to satisfies a specified condition. It returns true if at least one element satisfies the condition else returns false if they do not.

All - It checks whether all the elements in a collection satisfy a specified condition. It returns true if all the elements satisfy the condition else returns false if they do not.

```
string[] names = { "Rohan", "Raj", "Rahul", "Rajesh", "Rita" };
bool IsFirstLetterR = names.All(name => name.StartsWith("R")); //return true

string[] skills = { "C", "C++", "C#", "ASP.NET", "LINQ" };
bool IsFirstLetterC = skills.Any(s => s.StartsWith("C")); //return true

string[] hobbies = { "Swimming", "Cricket", "Singing", "Watching Moview"};
bool IsHobby = hobbies.Contains("Swimming"); //return true
```

Contains - It checks for an element match in a collection. It returns true if match found else returns false if no match found.

Q13. What is the difference between into and let keyword in LINQ?

Ans. **Into -** This is used to store the results of a group, join or select clause into a temporary variable. It hides the previous range variable and creates a temporary range variable which you can be used further.

```
DataContext context = new DataContext();
var q=from emp in context.tblEmployee
      group emp by new { emp.Salary, emp.EmpId }
      into groupingEmp
      let avgsalary = (groupingEmp.Sum(gEmp => gEmp.Salary) /
groupingEmp.Count())
      where groupingEmp.Key.Salary == avgsalary
      select new { groupingEmp.Key.Salary, groupingEmp.Key.EmpId };
```

Let - This is used to store the result of a subexpression into a new variable. It doesn't hide the previous range variable and creates a new variable which can be used further with the previous variable.

Q14. Explain Union, Intersect and Except?

Ans. **Union -** It returns the union of two collections. In union elements will be unique.

```
int[] numbers1 = { 1, 2, 3, 4, 5};
int[] numbers2 = { 5, 6, 7, 8, 9, 10 };

IEnumerable<int> union = numbers1.Union(numbers2); //1,2,3,4,5,6,7,8,9,10
```

Intersect - It returns the intersection of two collections.

```
int[] numbers1 = { 1, 2, 3, 4, 5};
int[] numbers2 = { 5, 6, 7, 8, 9, 10 };
```

```
IEnumerable<int> intersection = numbers1. Intersect(numbers2); //5
```

Except - It returns the difference between two collections. It is just opposite to intersect.

```
int[] numbers1 = { 1, 2, 3, 4, 5};
int[] numbers2 = { 5, 6, 7, 8, 9, 10 };

IEnumerable<int> except = numbers1.Except(numbers2); //1,2,3,4
```

Q15. What is the extension of the file, when LINQ to SQL is used?

Ans. The extension of the file is **.dbml**.

Q16. What is data context class? Explain its role?

Ans. **Data context class** is a LINQ to SQL class that acts as a medium between a SQL server database and LINQ to SQL entities classes mapped to that database. It has the following responsibilities:

1. Contains the connection string information, methods to connect to the database and manipulating the data in the database.
2. Contains several methods that send updated data from LINQ to SQL entity classes to the database.
3. Methods can also be mapped to stored procedures and functions.
4. With data context, we can perform select, insert, update and delete operations over the data in the database.

Q17. What is deferred execution and immediate execution?

Or

What is the difference between deferred execution and immediate execution?

Ans. **Deferred Execution:** In case of differed execution, a query is not executed at the point of its declaration. It is executed when the Query variable is iterated by using a loop like as for, foreach.

```
DataContext context = new DataContext();
var query = from customer in context.Customers
            where customer.City == "Delhi"
            select customer; // Query does not execute here

 foreach (var Customer in query) // Query executes here
 {
     Console.WriteLine(Customer.Name);
 }
```

A LINQ query expression often causes deferred execution. Deferred execution provides the facility of **query reusability** since it always fetches the **updated data** from the data source which exists at the time of each execution.

Immediate Execution: In the case of immediate execution, a query is executed at the point of its declaration. The query which returns a **singleton** value (a single value or a set of values) like Average, Sum, Count, List etc. caused Immediate Execution.

You can force a query to execute immediately of by calling ToList, ToArray methods.

```
DataContext context = new DataContext();
var query = (from customer in context.Customers
             where customer.City == "Delhi"
             select customer).Count(); // Query execute here
```

Immediate execution doesn't provide the facility of query reusability since it always contains the **same data** which is fetched at the time of query declaration.

Q18. What are lazy/deferred loading and eager loading?

Or

What is the difference between lazy/deferred loading and eager loading?

Ans. **Lazy/Deferred Loading:** In case of lazy loading, related objects (child objects) are not loaded automatically with its parent object until they are requested. By default, LINQ supports lazy loading.

For Example:

```
DataContext context = new DataContext();
var query = context.Department.Take(3); // Lazy loading

foreach (var Dept in query)
{
   Console.WriteLine(Dept.Name);
   foreach (var Emp in Dept.Employee)
   {
       Console.WriteLine(Emp.EmpID);
    }
 }
```

Generated SQL Query by LINQ Pad will be:

```
SELECT TOP (3)
[c].[DeptID] AS [DeptID],
[c].[Name] AS [Name],
[c].[CreatedDate] AS [CreatedDate]
FROM [dbo].[Department] AS [d]
GO
-- Region Parameters
DECLARE @EntityKeyValue1 Int = 1
-- EndRegion
SELECT
[Extent1].[EmpID] AS [EmpID],
[Extent1].[Name] AS [Name],
[Extent1].[Salary] AS [Salary],
[Extent1].[DeptID] AS [DeptID],
[Extent1].[CreatedDate] AS [CreatedDate],
FROM [dbo].[Employee] AS [Extent1]
```

```
WHERE [Extent1].[DeptID] = @EntityKeyValue1
GO

-- Region Parameters
DECLARE @EntityKeyValue1 Int = 2
-- EndRegion
SELECT
[Extent1].[EmpID] AS [EmpID],
[Extent1].[Name] AS [Name],
[Extent1].[Salary] AS [Salary],
[Extent1].[DeptID] AS [DeptID],
[Extent1].[CreatedDate] AS [CreatedDate],
FROM [dbo].[Employee] AS [Extent1]
WHERE [Extent1].[DeptID] = @EntityKeyValue1
GO

-- Region Parameters
DECLARE @EntityKeyValue1 Int = 3
-- EndRegion
SELECT
[Extent1].[EmpID] AS [EmpID],
[Extent1].[Name] AS [Name],
[Extent1].[Salary] AS [Salary],
[Extent1].[DeptID] AS [DeptID],
[Extent1].[CreatedDate] AS [CreatedDate],
FROM [dbo].[Employee] AS [Extent1]
WHERE [Extent1].[DeptID] = @EntityKeyValue1
```

In above example, you have 4 SQL queries which mean calling the database 4 times, one for the Categories and three times for the Products associated to the Categories. In this way, the child entity is populated when it is requested.

Eager loading: In case of eager loading, related objects (child objects) are loaded automatically with its parent object. To use Eager loading you need to use **Include**() method.

For Example:

```
DataContext context = new DataContext();
var query = context.Department.Include("Employee").Take(3); // Eager loading

foreach (var Dept in query)
 {
    Console.WriteLine(Dept.Name);
    foreach (var Emp in Dept.Employee)
    {
       Console.WriteLine(Emp.EmpID);
       }
  }
```

Generated SQL Query will be:

```
SELECT [Project1].[DeptID] AS [DeptID],
       [Project1].[Name] AS [Name],
       [Project1].[CreatedDate] AS [CreatedDate],
       [Project1].[D1] AS [D1],
       [Project1].[EmpID] AS [EmpID],
       [Project1].[Name] AS [Name1],
       [Project1].[Salary] AS [Salary],
       [Project1].[DeptID] AS [DeptID1],
       [Project1].[CreatedDate] AS [CreatedDate1]
       FROM (SELECT
       [Limit1].[DeptID] AS [DeptID],
       [Limit1].[Name] AS [Name],
       [Limit1].[CreatedDate] AS [CreatedDate],
       [Extent2].[EmpID] AS [EmpID],
       [Extent2].[Name] AS [Name1],
       [Extent2].[Salary] AS [Salary],
       [Extent2].[DeptID] AS [DeptID1],
       [Extent2].[CreatedDate] AS [CreatedDate1]
       CASE WHEN ([Extent2].[EmpID] IS NULL) THEN CAST(NULL AS int)
ELSE 1 END AS [D1]
FROM      (SELECT  TOP  (3)  [d].[DeptID]  AS  [DeptID],  [d].[Name]  AS  [Name],
[c].[CreatedDate] AS [CreatedDate]
        FROM [dbo].[Department] AS [d] )
AS [Limit1]
        LEFT OUTER JOIN [dbo].[Employee] AS [Extent2]
ON [Limit1].[DeptID] = [Extent2].[EmpID]) AS [Project1]
        ORDER BY [Project1].[DeptID] ASC, [Project1].[D1] ASC
```

In above example, you have only one SQL queries which mean calling the database only one time, for the Categories and the Products associated to the Categories. In this way, the child entity is populated with the parent entity.

Q19. Can you disable lazy/deferred loading?

Ans. Yes, you can turn off the lazy loading feature by setting a **LazyLoadingEnabled** property of the **ContextOptions** on context to false. Now you can fetch the related objects with the parent object in one query itself.

```
context.ContextOptions.LazyLoadingEnabled = false;
```

Q20. What is explicit loading?

Ans. By default, related objects (child objects) are not loaded automatically with its parent object until they are requested. To do so you have to use the **load method** on the related entity's navigation property.

For example:

```
// Load the products related to a given category
context.Entry(cat).Reference(p => p.Product).Load();
```

If lazy loading is disabled then it is still possible to lazily load related entities by explicit loading.

3

Joins and Queries

Q1. What are different types of joins in LINQ?

Ans. LINQ has a JOIN query operator that provides SQL JOIN like behaviour and syntax. There are four types of Joins in LINQ.

1. **INNER JOIN** - Inner join returns only those records or rows that match or exists in both the tables.

```
DataContext context = new DataContext();
var q = (from pd in context.Products
         join od in context.Orders on pd.ProductID equals od.ProductID
         orderby od.OrderID
         select new
         {
           od.OrderID,
           pd.ProductID,
           pd.Name,
           pd.UnitPrice,
           od.Quantity,
           od.Price,
         }).ToList();
```

2. **GROUP JOIN**- When a join clause uses an INTO expression, then it is called a group join. A group join produces a sequence of object arrays based on properties equivalence of left collection and right collection. If the right collection has no matching elements with left collection then an empty array will be produced.

```
DataContext context = new DataContext();
var q = (from pd in context.tblProducts
         join od in context.tblOrders on pd.ProductID equals od.ProductID
         into t
         orderby pd.ProductID
         select new
         {
           pd.ProductID,
           pd.Name,
           pd.UnitPrice,
           Order = t
        }).ToList();
```

Basically, GROUP JOIN is like as **INNER-EQUIJOIN** except that the result sequence is organized into groups.

3. **LEFT OUTER JOIN** - LEFT JOIN returns all records or rows from the left table and from right table returns only matched records. If there are no columns matching in the right table, it returns NULL values.

   In LINQ to achieve LEFT JOIN behavior, it is mandatory to use "INTO" keyword and "DefaultIfEmpty()" method. We can apply LEFT JOIN in LINQ like as:

```
DataContext context = new DataContext();
var q = (from pd in context.tblProducts
        join od in context.tblOrders on pd.ProductID equals od.ProductID into t
        from rt in t.DefaultIfEmpty()
        orderby pd.ProductID
        select new
        {
          OrderID = rt.OrderID,
          pd.ProductID,
          pd.Name,
          pd.UnitPrice,
          rt.Quantity,
          rt.Price,
        }).ToList();
```

4. **CROSS JOIN**- Cross join is a Cartesian join means the Cartesian product of both the tables. This join does not need any condition to join two tables. This join returns records or rows that are a multiplication of record number from both the tables' means each row on the left table will relate to each row of the right table.

   In LINQ to achieve CROSS JOIN behaviour, there is no need to use Join clause and where clause. We will write the query as shown below.

```
DataContext context = new DataContext();
var q = from c in context.Customers
        from o in context.Orders
        select new
         {
             c.CustomerID,
             c.ContactName,
             o.OrderID,
             o.OrderDate
         };
```

Q2. How to write LINQ query for Inner Join with *and* condition?

Ans. Sometimes, you need to apply inner join with *and* condition. To write a query for inner join with *and* condition you need to make two anonymous types (one for a left table and one for a right table) by using new keyword and compare both the anonymous types as shown below:

```
DataContext context = new DataContext();
var q=from cust in context.tblCustomer
      join ord in context.tblOrder
```

```
// Both anonymous types should have the exact same number of properties having
same name and datatype
       on new {a=(int?)cust.CustID, cust.ContactNo} equals new {a=ord.CustomerID,
ord.ContactNo}
      select new
      {
            cust.Name,
            cust.Address,
            ord.OrderID,
            ord.Quantity
      };

// Generated SQL
SELECT [t0].[Name], [t0].[Address], [t1].[OrderID], [t1].[Quantity]
FROM [tblCustomer] AS [t0]
INNER JOIN [tblOrder] AS [t1] ON (([t0].[CustID]) = [t1].[CustomerID]) AND
([t0].[ContactNo] = [t1].[ContactNo])
```

Note -

1. Always remember, both the anonymous types should have the exact same number of properties with same name and Datatype otherwise you will get the compile-time error "**Type inference failed in the call to Join**".
2. Both the comparing fields should define either NULL or NOT NULL values.
3. If one of them is defined NULL and other is defined NOT NULL then we need to do typecasting of the NOT NULL field to NULL data type like as above fig.

Q3. How to write LINQ query for Inner Join with *OR* condition?

Ans. Sometimes, you need to apply inner join with *or* condition. To write a query for inner join with *or* condition you to need to use the || operator in where condition as shown below:

```
DataContext context = new DataContext();
var q=from cust in context.tblCustomer
      from ord in context.tblOrder
      where (cust.CustID==ord.CustomerID || cust.ContactNo==ord.ContactNo)
           select new
           {
              cust.Name,
              cust.Address,
              ord.OrderID,
              ord.Quantity
           };

// Generated SQL
SELECT [t0].[Name], [t0].[Address], [t1].[OrderID], [t1].[Quantity]
FROM [tblCustomer] AS [t0], [tblOrder] AS [t1]
```

```
WHERE (([t0].[CustID]) = [t1].[CustomerID]) OR ([t0].[ContactNo] =
[t1].[ContactNo])
```

Q4. Write LINQ query to find 2nd highest salary?

OR

Write LINQ query to find nth highest salary?

Ans. You can find the nth highest salary by writing the following query.

```
DataContext context = new DataContext();
var q= context.tblEmployee.GroupBy(ord => ord.Salary)
      .OrderByDescending(f => f.Key)
      .Skip(1) //second, third ..nth highest salary where n=1,2...n-1
      .First()
      .Select(ord=>ord.Salary)
      .Distinct();
```

Q5. How to use GroupBy in LINQ?

Ans. It is used for grouping of elements based on a specified value. Each group has a key. It returns a collection of IGrouping<Key, Values>.

```
DataContext context = new DataContext();
var query=from ord in context.tblOrder
          group ord by ord.CustomerID into grouping
          select
          new
          {
              grouping.Key,
              grouping
          };
```

Q6. How to use Having in LINQ?

Ans. There is no having operator in LINQ, but you can get the same result by using a Where clause after a Group By clause.

```
DataContext context = new DataContext();
var q=from ord in context.tblOrder
      group ord by ord.CustomerID into grouping
      where grouping.Count()>=2
      select
      new
      {
         grouping.Key,
         grouping
      };
```

Q7. What is Expression?

Ans. In .NET, Expression is an abstract class which contains static methods and inherited by various types (like ParameterExpression, MethodCallExpression, BinaryExpression etc.) to create expression tree nodes of specific types. Also, all these expression-specific expression tree types are defined in System.Linq.Expressions namespace.

Q8. What is Expression Trees?

Ans. Expression Trees was first introduced in C# 3.0 (Visual Studio 2008), where they were mainly used by LINQ providers. Expression trees represent code in a tree-like format, where each node is an expression (**for example**, a method call or a binary operation such as x < y).

You can also convert expression trees into compiled code and run it. This transformation enables dynamic modification of executable code as well as the execution of LINQ queries in various databases and the creation of dynamic queries.

Q9. How can you create Expression Trees?

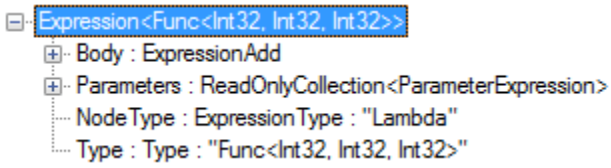
Ans. You can create expression trees by using the following two ways:

1. **Using Expression Lambda -** The easiest way to generate an expression tree is to create an instance of the Expression<T> type, where T is a delegate type, and assign a lambda expression to this instance.

```
// Create an expression using expression lambda
Expression<Func<int, int, int>> expression = (num1, num2) => num1 + num2;

// Compile the expression
Func<int, int, int> compiledExpression = expression.Compile();

// Execute the expression.
int result = compiledExpression(3, 4); //return 7
```

In this example, the C# compiler generates the expression tree from the provided lambda expression. This looks much more complicated, but this is what actually happens when you supply a lambda expression to an expression tree.

The structure of the above expression trees will be like as:



2. **Using Expression Tree API -** Expression class is used to create expression trees by using the API. In .NET Framework 4, the expression trees API also supports assignments and control flow expressions such as loops, conditional blocks, and try-catch blocks. By using the API, you can create expression trees that are

more complex than those that can be created from lambda expressions. Using API, the above code can be re-written as:

```
//Create the expression parameters
ParameterExpression num1 = Expression.Parameter(typeof(int), "num1");
ParameterExpression num2 = Expression.Parameter(typeof(int), "num2");

//Create the expression parameters
ParameterExpression[] parameters = new ParameterExpression[] { num1, num2 };

//Create the expression body
BinaryExpression body = Expression.Add(num1, num2);

//Create the expression
Expression<Func<int, int, int>> expression = Expression.Lambda<Func<int,
int, int>>(body, parameters);

// Compile the expression
Func<int, int, int> compiledExpression = expression.Compile();

// Execute the expression
int result = compiledExpression(3, 4); //return 7
```

Q10. How Expressions Trees are different from Lambda Expression?

Ans. A common misunderstanding is that expression trees are identical to lambda expressions. But this is not true since you can also create and modify expression trees by using API methods i.e. without using lambda expression syntax at all.

Also, every lambda expression cannot be implicitly converted into an expression tree. Only expression lambda is implicitly converted an expression tree and statement lambda i.e. multi-line lambda cannot be implicitly converted into an expression tree.

Q11. How LINQ query is compiled to Expression Trees?

Ans. In LINQ, a query expression is compiled to expression trees or to delegates, depending on the type that is applied to query result. The IEnumerable<T> type LINQ queries are compiled to delegates and IQueryable or IQueryable<T> type LINQ queries are compiled to expression trees.

In short, LINQ queries that execute in the process are compiled into delegates while the queries that execute out of the process are compiled into expression trees.

In LINQ, domain-specific queries (like LINQ to SQL, LINQ to Entity) result into IQueryable<T> type. The C# and Visual Basic compilers compile these queries into code that builds an expression tree at runtime. Then query provider traverses the expression tree and translate it into a query language (like T-SQL) which is appropriate for that data source.

Consider the following LINQ to SQL query expression:

```
var query = from c in db.Customers
            where c.City == "Delhi"
```

```
            select new { c.City, c.CompanyName };
```

Here query variable, returned by this LINQ query is of type IQueryable and the declaration of IQueryable is as:

```
public interface IQueryable : IEnumerable
{
   Type ElementType { get; }
   Expression Expression { get; }
   IQueryProvider Provider { get; }
}
```

As you can see, IQueryable contains a property of type Expression which holds the expression tree and represents data structure equivalent to the executable code found in a query expression. The IQueryProvider type property holds the LINQ provider (like LINQ to SQL, LINQ to Entity etc.) and based on LINQ provider query is translated into an appropriate query (like T-SQL query). The Type property represents the type of the element(s) that are returned when the expression tree is executed.

In this way, the above code is never actually executed inside your program. It is first translated into the following SQL statement and then executed on SQL Server side.

```
SELECT [t0].[City], [t0].[CompanyName]
FROM [dbo].[Customers] AS [t0]
WHERE [t0].[City] = @p0
```

Q12. When to use var or IEnumerable to store the query result in LINQ?

Ans. You can use var or IEnumerable<T> to store the result of a LINQ query. You should take care of following points while choosing between var and IEnumerable.

var	IEnumerable
Use var type when you want to make a "custom" type on the fly.	Use IEnumerable when you already know the type of query result.
var is also good for remote collection since var is an IQueryable type that executes the query in SQL server with all filters.	IEnumerable is good for in-memory collection.

Q13. What is the difference between IEnumerable and IQueryable?

Ans. There are following differences between ADO.NET and Entity Framework:

IEnumerable	IQueryable
Exists in System.Collections Namespace	Exists in System.Linq Namespace
Best to query data from in-memory collections like List, Array etc.	Best to query data from out-memory (like remote database, service) collections.

While querying data from the database, IEnumerable executes a select query on the server side, load data in-memory on the client side and then filter data.
For Example:
`DataContext context = new DataContext();`
`IEnumerable<Employee> list = context.Employees.Where(p => p.Name.StartsWith("S"));`
`list = list.Take<Employee>(10);`
Generated SQL having no Top Keyword:
`SELECT [t0].[EmpID], [t0].[EmpName], [t0].[Salary] FROM [Employee] AS [t0] WHERE [t0].[EmpName] LIKE @p0`
	While querying data from the database, IQueryable executes a select query on the server side with all filters.
	For Example:
	`DataContext context = new DataContext();`
	`IQueryable<Employee> list = context.Employees.Where(p => p.Name.StartsWith("S"));`
	`list = list.Take<Employee>(10);`
	Generated SQL having Top Keyword:
	`SELECT TOP 10 [t0].[EmpID], [t0].[EmpName],[t0].[Salary] FROM [Employee] AS [t0] WHERE [t0].[EmpName] LIKE @p0`

Suitable for LINQ to Object and LINQ to XML queries.	Suitable for LINQ to SQL queries.
Doesn't support lazy loading. Hence not suitable for paging like scenarios.	Support for lazy loading. Hence it is suitable for paging like scenarios.
Doesn't supports custom query.	Supports custom query using CreateQuery() and Execute() methods.

Q14. What is the difference between IEnumerable and IList?

Ans. There are following differences between ADO.NET and Entity Framework:

IEnumerable	IList
Move forward only over a collection, it can't move backwards and between the items.	Used to access an element in a specific position/index in a list.
Doesn't support add or remove items from the list.	Useful when you want to Add or remove items from the list.
Find out the no of elements in the collection after iterating the collection.	Find out the no of elements in the collection without iterating the collection.
Supports further filtering.	Doesn't support further filtering.

Q15. What is SQL metal in LINQ?

Ans. SQL metal is a command line tool to generate code and mapping for LINQ to SQL. It automatically generates the entity classes for the given database. It is included in Windows SDK that is installed with Visual Studio.

Syntax for SQL metal at command prompt

```
sqlmetal [options] [<input file>]
```

To view the options list, type sqlmetal /? on the command prompt. You can define various options such as Connection Options, Extraction options, and Output options. Input file might be SQL Server .mdf file, .sdf file, or a .dbml intermediate file.

It performs the following actions:

1. Generate source code and mapping attributes or a mapping file from a database.
2. Generate an intermediate database markup language (.dbml) file for customization from a database.
3. Generate code and mapping attributes or a mapping file from a .dbml file.

Q16. What is the difference between LINQ and Stored Procedures?

Ans. There are the following differences between LINQ and Stored Procedures.

1. Stored procedures are faster as compared to LINQ query since they have a predictable execution plan and can take the full advantage of SQL features. Hence, when a stored procedure is being executed next time, the database used the cached execution plan to execute that stored procedure.
2. LINQ has full **type checking** at compile-time and **IntelliSense** support in Visual Studio as compared to the stored procedure. This powerful feature helps you to avoid run-time errors.
3. LINQ allows debugging through .NET debugger as compared to the stored procedure.
4. LINQ also supports various .NET framework features like multithreading as compared to stored procedures.
5. LINQ provides the uniform programming model (means common query syntax) to query the multiple databases while you need to re-write the stored procedure for different databases.
6. A stored procedure is the best way for writing complex queries as compared to LINQ.
7. Deploying LINQ based application is much easy and simple as compared to stored procedures based. Since in case of stored procedures, you need to provide a SQL script for deployment but in case of LINQ, everything gets compiled into the DLLs. Hence you need to deploy only DLLs.

Q17. What are the disadvantages of LINQ over Stored Procedures?

Ans. There are the following disadvantages of LINQ over Stored Procedures.

1. LINQ query is compiled each and every time while stored procedures re-used the cached execution plan to execute. Hence, LINQ query takes more time in execution as compared to stored procedures.
2. LINQ is not good for writing complex queries as compared to stored procedures.
3. LINQ is not a good way for bulk insert and update operations.
4. Performance is degraded if you don't write the LINQ query correctly.
5. If you have done some changes in your query, you have to recompile it and redeploy its dll to the server.

Q18. What is the difference between XElement and XDocument?

Ans. XElement and XDocument are the classes defined within **System.Xml.Linq** namespace. XElement class represents an XML fragment. While XDocument class represents an entire XML document with all associated properties.

For Example:

```
XDocument doc = new XDocument(new XElement("Book",
```

```
                                    new XElement("Name", ".NET Interview FAQ"),
                                    new XElement("Author", "Shailendra Chauhan"))
                                   );
```

Q19. What is difference between XElement.Load() and XDocument.Load()?

Ans. XElement and XDocument are very similar in function, but not interchangeable when you are loading a document from a data source. Use XElement.Load () when you want to load everything under the top-level element, Use XDocument.Load () when you want to load any markup before the top-level element.

Q20. What is the difference between ADO.NET and LINQ to SQL?

Ans. There are following differences between ADO.NET and Entity Framework:

ADO.NET	LINQ to SQL
It is a part of the .NET Framework since .NET Framework 1.0	It is a part of the .NET Framework since .NET Framework 3.5
SqlConnection/OleDbConnection is used for database connectivity.	We can use context for database connectivity.
Difficult to debug and cause syntax errors at run-time.	Easy to debug and cause syntax errors at compile-time.
It has full type checking at run-time and no IntelliSense support in Visual Studio since it used the T-SQL to query the database.	It has full type checking at compile-time and IntelliSense support in Visual Studio since it used the .NET Framework languages like C# and VB.
It used T-SQL to query the data to query the database and some other syntax for querying the other data source.	It used LINQ to query the data which provides the uniform programming model (means common query syntax) to query the various data sources.

Q21. How can you handle concurrency in LINQ to SQL?

Ans. When multiple users are updating the same record at the same time, a conflict will occur. To handle this concurrency conflicts, LINQ provides you following three ways.

1. **KeepCurrentValues** - This option will remain the LINQ object values as it is and does not push the new values from the database into the LINQ object.
2. **OverwriteCurrentValues** - This option will replace the LINQ object values with the database values.
3. **KeepChanges** - In this case, changed properties of an object/entity remains as it is but the properties which are not changed are fetched from the database and replaced.

All these options are available in ***refresh mode*** enum which exists in *System.Data.Linq* namespace.

To handle concurrency conflicts, wrap the code with in a try block and catch the *ChangeConflictException* by using catch block and iterate through the *ChangeConflicts* collection to resolve the conflict as shown below:

```
DataContext db = new DataContext();
try
{
 // TO DO:
  db.SubmitChanges(ConflictMode.ContinueOnConflict);
}
```

```
catch (ChangeConflictException ex)
{
   foreach (ObjectChangeConflict changeconf in db.ChangeConflicts)
   {
      changeconf.Resolve(RefreshMode.OverwriteCurrentValues);
    }
 }
```

Q22. How can you handle concurrency at field level in LINQ to SQL?

Ans. In LINQ to SQL, you can also handle concurrency at field level and this is the best way provided by the LINQ. To achieve this you need to define ***UpdateCheck*** attribute at field level and it has the following options:

1. **Never: -** This option will never check for concurrency conflicts.
2. **Always: -** This option will always check for concurrency conflicts.
3. **WhenChanged: -** This option will check for concurrency conflicts when the field's value has been changed.

```
[Column(DbType = "nvarchar(50)", UpdateCheck = UpdateCheck.Never)]
public string CustomerID
{
  set{CustomerID = value;}
  get{return _CustomerID;}
}
```

References

This book has been written by referring to the following sites:

1. https://docs.microsoft.com/en-us/dotnet/ - Microsoft Docs -.NET
2. https://stackoverflow.com/questions/tagged/linq - Stack Overflow - LINQ
3. https://www.dotnettricks.com/learn/linq - Dot Net Tricks – LINQ