Language Integrated Query

Interview Questions & Answers

- www.dotnet-tricks.com
- www.dotnetinterviewtricks.com

by Shailendra Chauhan

LINQ Interview Questions and Answers

Dedication

My mother Mrs. Vriksha Devi and my wife Reshu Chauhan deserve to have theirs name on the cover as much as I do for all theirs support made this possible. I would like to say thanks to all my family members Virendra Singh(father), Jaishree and Jyoti(sisters), Saksham and Kalap(sons), friends, to you and to readers or followers of my blog www.dotnet-tricks.com 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 theirs 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 sr. software engineer, consultant, blogger and author, I have through knowledge and understandings of .NET technologies. My inspiration and knowledge has 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 covers useful Interview Questions and Answers on LINQ. This book is appropriate for novice as well as for senior level professionals who wants to strengthen their skills before appearing for an interview on LINQ. This book is equally helpful to sharpen their programming skills and understanding LINQ in a short time.

This book is not only the LINQ interview book but it is more than that. This book helps you to get the depth knowledge of LINQ with a simple and elegant way. This book is updated to latest version of LINQ.

I hope you will enjoy this book and find it useful. At the same time I also encourage you to become a continue reader of the blog www.dotnet-tricks.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.dotnet-tricks.com that contains .NET, C#, ASP.NET MVC, LINQ, Entity Framework, jQuery and many more tips, tricks and tutorials.

All the best for your interview and happy programming!

About the author

Shailendra Chauhan

Works as Sr. Software Engineer at reputed MNC and has more than 5 years of hand over Microsoft .NET technologies. He is a .NET Consultant, Author, Blogger, founder and chief editor of www.dotnet-tricks.com and www.dotnetinterviewtricks.com.

A number of articles of him has become articles-of-the-day, selected in daily-community-spotlight, and listed in Recommended Resources for MVC section in The Official Microsoft ASP.NET Site. His blog www.dotnet-tricks.com is a well-known knowledge and support resource in the field of .NET technologies worldwide and is listed as a non-Microsoft resource in The Microsoft Official

He likes to share his working experience, research and knowledge through his well-known blogs. He is an author of books ASP.NET MVC Interview Questions and Answers and LINQ Interview Questions and Answers. He is a technical reviewer of book ASP.NET MVC 4 Mobile App Development.

He loves to work with web applications, Mobile apps and Mobile websites using Microsoft technology including C#, ASP.NET, ASP.NET MVC, SQL Server, WCF, WEB API, LINQ, Entity Framework, jQuery, jQuery UI, jQuery Mobile, Knockout.js, Windows Azure, Backbone.js, PhoneGap and many more web technologies.

He strives to be the best he can be. He always keeps up with new technologies and learning new skills that allow him to provide better solutions to problems.

How to contact the author

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 author e-mail addresses are listed as follows:

- pro.shailendra@dotnet-tricks.com
- pro.dotnettricks@gmail.com

I am always happy to hear from my readers. Please provide with your valuable feedback and comments!

You can also follow www.dotnet-tricks.com at facebook, twitter, linkedin and google plus or subscribe to RSS feed.

Table of Contents

LINQ Interview Questions and Answers 1

Dedication 2

Introduction 3

About the author 4

How to contact the author 5

LINQ 8

Q1. What is LINQ and why to use it? 8

Q2. Which namespace is necessary to use LINQ? 8

Q3. What are different flavors of LINQ? 8

Q4. What are advantages of LINQ? 9

Q5. What are disadvantages of LINQ? 9

Q6. What are different methods to write LINQ Query? 9

Q7. How var type is different from anonymous type? 10

Q8. What is anonymous method? 12

Q9. What is lambda expression? 13

Q10. What is Extension method? 14

Q11. What is LINQPad? 15

Q12. What LINQ providers are supported by LINQPad? 15

Q13. What is LINQ Insight? 15

Q14. What are Quantifiers? 15

Q15. What are different types of operators in LINQ? 16

Q16. What are aggregate operators? 16

Q17. What are conversion operators? 16

Q18. What are element operators? 16

Q19. What is LINQ provider and what are different types of LINQ providers? 16

Q20. What are advantages of using LINQ on DataSet? 16

Q21. What is difference between Select and SelectMany in LINQ? 17

Q22. What is difference among Single, SingleOrDefault, First and FirstOrDefault? 19
Q23. When to use Single, SingleOrDefault, First and FirstOrDefault? 19
Q24. Which one is fast between SingleOrDefault and FirstOrDefault? 19
Q25. What is difference among Any, All and Contains? 19
Q26. What is difference between into and let keyword in LINQ? 20
Q27. Explain Union, Intersect and Except? 20
Q28. What is the extension of the file, when LINQ to SQL is used? 20
Q29. What is data context class? Explain its role? 20
Q30. What are deferred execution and immediate execution? 21
Q31. What are lazy/deferred loading and eager loading? 21
Q32. Can you disable lazy/deferred loading? 24
Q33. What is explicit loading? 24
Q34. What are different types of joins in LINQ? 24
Q35. How to write LINQ query for Inner Join with *and* condition? 26
Q36. How to write LINQ query for Inner Join with *OR* condition? 27
Q37. Write LINQ query to find 2nd highest salary? 27
Q38. How to use GroupBy in LINQ? 27
Q39. How to use Having in LINQ? 28
Q40. What is difference between IEnumerable and IQueryable? 28
Q41. When var or IEnumerable is used to store query result in LINQ? 29
Q42. What is difference between IEnumerable and IList? 29
Q43. What is SQL metal in LINQ? 29
Q44. What is difference between LINQ and Stored Procedures? 30
Q45. What are disadvantages of LINQ over Stored Procedures? 30
Q46. What is difference between XElement and XDocument? 30
Q47. What is difference between XElement.Load() and XDocument.Load()? 30
Q48. What is difference between ADO.NET and LINQ to SQL? 31
Q49. How can you handle concurrency in LINQ to SQL? 31
Q50. How can you handle concurrency at field level in LINQ to SQL? 32

1 LINQ

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 the 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 flavors of LINQ?

Ans. There are following three flavors 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**
   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 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 query 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.

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 .NET debugger.

Q5. What are 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 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 two 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, query expression is converted into Method Invocation.

The Syntax for Query Expression is as:

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

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

   Method syntax is complex as compared to Query expression since it uses lambda expression to write LINQ query. It is easily understood by .NET CLR. Hence at compile time, 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>.

   The Syntax for Method Invocation is as:

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

Note: You can use a mixture of both syntax by enclosing a query expression inside parentheses and make a call to method. There are no sematic 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 value to get desired output.

Q8. What is anonymous method?

Ans. The concept of 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 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 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 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 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, lambda expression is a more concise syntax of anonymous method. It is just a new way to write 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 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 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 = d1(3, 3);
        int z3 = d1(4, 3);

        Console.WriteLine(z1);
        Console.WriteLine(z2);
```

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

Q10. What is 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 behaviors to an existing type** without altering.

In extension method "**this**" keyword is used with the first parameter and the first parameter will be of **type** that is extended by 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
```

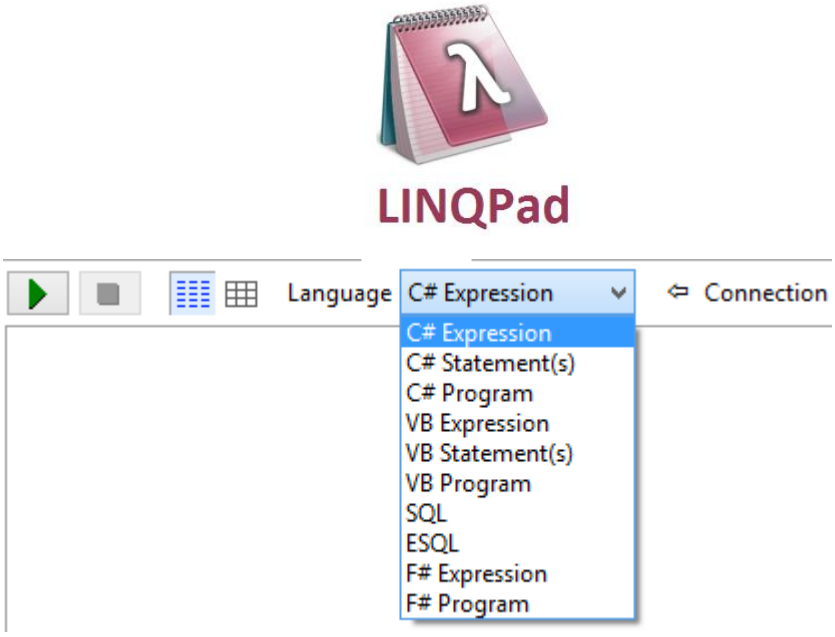
Key points about extension method

1. An extension method is defined as 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 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 same name and signature are defined in two different namespaces and these namespaces are included in same class file using directives. Compiler will cause an error if you will try to call one of them 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 of 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.

Q14. 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()

Q15. 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

Q16. 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.

Q17. 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.

Q18. 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.

Q19. 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.

Q20. 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.

Q21. What is difference between Select and SelectMany in LINQ?

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

Select operator produce one result value for every source value while **SelectMany** produce 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
*/
```

Q22. What is difference among Single, SingleOrDefault, First and FirstOrDefault?

Ans. **Single -** It returns a single specific element from a collection of elements if 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 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 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 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.

Q23. 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**.

Q24. Which one is fast between SingleOrDefault and FirstOrDefault?

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

Q25. What is difference among Any, All and Contains?

Ans. **Any -** It checks whether at least one element in a collection satisfy a specified condition. It returns true if at least one element satisfy 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.

Q26. What is 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 create 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 sub expression into a new variable. It doesn't hide the previous range variable and create a new variable which can be used further with previous variable.

Q27. 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
```

Q28. What is the extension of the file, when LINQ to SQL is used?

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

Q29. 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 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.

Q30. What are deferred execution and immediate execution?

Or

What is 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 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 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.

Q31. What are lazy/deferred loading and eager loading?

Or

What is 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 means calling the database 4 times, one for the Categories and three times for the Products associated to the Categories. In this way, 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 means calling the database only one time, for the Categories and the Products associated to the Categories. In this way, child entity is populated with parent entity.

Q32. Can you disable lazy/deferred loading?

Ans. Yes, you can turn off the lazy loading feature by setting **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;
```

Q33. 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.

Q34. What are different types of joins in LINQ?

Ans. LINQ has a JOIN query operator that provides SQL JOIN like behavior 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 use 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 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 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 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 multiplication of record number from both the tables' means each row on left table will relate to each row of right table.

In LINQ to achieve CROSS JOIN behavior, 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
         };
```

Q35. How to write LINQ query for Inner Join with *and* condition?

Ans. Sometimes, you need to apply inner join with *and* condition. To write query for inner join with *and* condition you need to make two anonymous types (one for left table and one for 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 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 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 NOT NULL field to NULL data type like as above fig.

Q36. How to write LINQ query for Inner Join with *OR* condition?

Ans. Sometimes, you need to apply inner join with *or* condition. To write query for inner join with *or* condition you need to use || 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])
```

Q37. Write LINQ query to find 2nd highest salary?

OR

Write LINQ query to find nth highest salary?

Ans. You can find 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();
```

Q38. 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
            };
```

Q39. 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
      };
```

Q40. What is 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 database, IEnumerable execute select query on server side, load data in-memory on 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 database, IQueryable execute select query on 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 lazy loading. Hence it is suitable for paging like scenarios.
Doesn't supports custom query.	Supports custom query using CreateQuery() and Execute() methods.

Q41. When var or IEnumerable is used to store 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 IEnumrable.

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 query in SQL server with all filters.	IEnumerable is good for in-memory collection.

Q42. What is 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 backward 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.

Q43. 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 /? at 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 perform 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.

Q44. What is 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 stored procedure. This powerful feature helps you to avoid run-time errors.
3. LINQ allows debugging through .NET debugger as compared to stored procedure.
4. LINQ also supports various .NET framework features like multi –threading 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. Stored procedure is a 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 complied into the DLLs. Hence you need to deploy only DLLs.

Q45. What are 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 the 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.

Q46. What is difference between XElement and XDocument?

Ans. XElement and XDocument are the classes defined with in **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"))
                             );
```

Q47. 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.

Q48. What is 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 .NET Framework since .NET Framework 1.0	It is a part of .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 compile-time and 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.
Easier syntax and coding.	Syntax and coding is somewhat complex.

Q49. 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 remains the LINQ object values as it is and does not push the new values from the database in to 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.

We need to use the **"RefereshMode"** to specify above options.

To handle concurrency conflicts, wrap the code with in a "TRY" block and catch the "ChangeConflictException". Within catch block iterate through the "ChangeConflicts" collection to resolve the conflict as shown below:

```
try
{
  //To Do:
}
catch (ChangeConflictException ex)
{
   foreach (ObjectChangeConflict changeconf in context.ChangeConflicts)
   {
      changeconf.Resolve(RefreshMode.OverwriteCurrentValues);
    }
 }
```

Q50. How can you handle concurrency at field level in LINQ to SQL?

Ans. Handling concurrency conflicts at field level is the best way provided by the LINQ. For this you need to define "**UpdateCheck**" attribute at field level. UpdateCheck attribute 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.

For Example:

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