**LINQ**

**1) Explain what is LINQ? Why is it required?**

Language Integrated Query or LINQ is the collection of standard query operators which provides query facilities into.NET framework language like C#, VB.NET.

LINQ is required as it bridges the gap between the world of data and world of objects.

**4) List out the three main components of LINQ? Explain what is the extension of the file, when LINQ to SQL is used?**

Three main components of LINQ are

* Standard Query Operators
* Language Extensions
* LINQ Providers

**Define what is Where clause and Let clause?**

* **Where clause**: It allows adding some conditional filters to the query.
* **Let clause**: It allows defining a variable and assigning it a value calculated from the data values.

**Explain why SELECT clause comes after FROM clause in LINQ?**

* With other programming language and C#, LINQ is used, it requires all the variables to be declared first. “FROM” clause of LINQ query defines the range or conditions to select records. So, FROM clause must appear before SELECT in L
* **8) Explain what is lambda expressions in LINQ?**
* Lambda expression is referred as a unique function use to form delegates or expression tree types, where right side is the output and left side is the input to the method. For writing LINQ queries particularly, Lambda expression is used.
* **9) Explain how LINQ with databases can be used?**
* LINQ supports XML, SQL, Dataset and Objects. Through LINQ to objects or LINQ to Datasets one can use LINQ with other databases.

**Explain what is the difference between Skip() and SkipWhile() extension method?**

* **Skip()** **:** It will take an integer argument and from the given IEnumerable it skips the top n numbers
* **SkipWhile ():** It will continue to skip the elements as far as the input condition is true. It will return all remaining elements if the condition is false

[Take()](https://msdn.microsoft.com/en-us/library/bb503062%28v=vs.110%29.aspx)**method**

The Take() method extracts the first n elements (where n is a parameter to the method) from the beginning of the target sequence and returns a new sequence containing only the elements taken.

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

List**<int>** ResultQS = **(from** num in numbers

**select** num**)**.Take**(**4**)**.ToList**()**;

**foreach(**var num in ResultQS**)**

**{**

Console.Write**(**$"{num} "**)**;

**}**

**Output: 1 2 3 4**

What is the difference between First() and Take(1)?

The difference between First() and Take() is that First() returns the element itself, while Take() returns a sequence of elements that contains exactly one element. (If you pass 1 as the parameter).

**In LINQ how will you find the index of the element using where () with Lambda Expressions?**

* In order to find the index of the element using where () with the lambda expression
* Where ( ( i, ix ) => i == ix);

Mention what is the role of DataContext classes in LINQ?

**DataContext** class acts as a bridge between SQL Server database and the LINQ to SQL. For accessing the database and also for changing the data in the database, it contains connections string and the functions. Essentially a DataContext class performs the following three tasks:

* Create connection to database.
* It submits and retrieves object to database.
* Converts objects to SQL queries and vice versa.
* **15) Explain what are LINQ query expressions?**
* Query expression is nothing but an LINQ query. It is a combination of query clauses that identifies the data sources for a query. It contains information for sorting, filtering, grouping or joining to apply to the source data. It determines what information should be retrieved from the data source.CV.

**16) Explain what are compiled queries?**

In compiled LINQ queries, the plan is cached in a static class and static class is a global cache. Rather than preparing the query plan from scratch, LINQ prepares plan using stating class object.

**17) Explain how standard query operators useful in LINQ?**

Standard Query Operators useful in LINQ are

* Get a total count of elements in the collection
* Order the results of a collection
* Grouping
* Computing average
* Joining two collections based on matching keys
* Filter the results

**18) Explain what is the purpose of LINQ providers in LINQ?**

LINQ providers are set of classes that take an LINQ query which generates method that executes an equivalent query against a particular data source.

**21) Explain what is “LINQ to Objects”?**

When LINQ queries any IEnumerable(Of T) collection or IEnumerable directly without the use of an intermediate LINQ provider or API such as LINQ to SQL or LINQ to XML is referred as “LINQ to Objects.”

What are the types of LINQ?

Answer

* LINQ to Objects
* LINQ to XML
* LINQ to Dataset
* LINQ to SQL
* LINQ to Entities

What are Anonymous Types?

Anonymous types are types that are generated by compiler at run time. When we create a anonymous type we do not specify a name. We just write properties names and their values. Compiler at runtime create these properties and assign values to them.

var k = new { FirstProperty = "value1", SecondProperty = "value2" };

Console.WriteLine(k.FirstProperty);

Anonymous class is useful in LINQ queries to save our intermediate results.

There are some restrictions on Anonymous types as well:

* Anonymous types can not implement interfaces.
* Anonymous types can not specify any methods.
* We can not define static members.
* All defined properties must be initialized.
* We can only define public fields.

C# - Anonymous Method

As the name suggests, an anonymous method is a method without a name. Anonymous methods in C# can be defined using the delegate keyword and can be assigned to a variable of delegate type.

public delegate void Print(int value);

static void Main(string[] args)

{

Print print = delegate(int val) {

Console.WriteLine("Inside Anonymous method. Value: {0}", val);

};

print(100);

}

Anonymous methods can access variables defined in an outer function

Anonymous methods can also be passed to a method that accepts the delegate as a parameter.

public delegate void Print(int value);

class Program

{

public static void PrintHelperMethod(Print printDel,int val)

{

val += 10;

printDel(val);

}

static void Main(string[] args)

{

PrintHelperMethod(delegate(int val) { Console.WriteLine("Anonymous method: {0}", val); }, 100);

}

}

Example: Anonymous Method as Event Handler

saveButton.Click += delegate(Object o, EventArgs e)

{

System.Windows.Forms.MessageBox.Show("Save Successfully!");

};

Anonymous Method Limitations

* It cannot contain jump statement like goto, break or continue.
* It cannot access ref or out parameter of an outer method.
* It cannot have or access unsafe code.
* It cannot be used on the left side of the is operator.

https://www.tutorialsteacher.com/Content/images/bulb-glow.png Points to Remember :

1. Anonymous method can be defined using the delegate keyword
2. Anonymous method must be assigned to a delegate.
3. Anonymous method can access outer variables or functions.
4. Anonymous method can be passed as a parameter.
5. Anonymous method can be used as event handlers.

**ENTITY FRAMEWORK ORM (EF, L2SQL, ADO.NET):**

(++)

* Rapid development
* Data access code now under source control
* You're isolated from changes in DB. If that happens you only need to update your model/mappings in one place.
* Easy to add test cases

(--)

* Performance may be worse
* No or little control over SQL the ORM produces (could be inefficient or worse buggy). Might need to intervene and replace it with custom stored procedures. That will render your code messy (some LINQ in code, some SQL in code and/or in the DB out of source control).
* As any abstraction can produce "high-level" developers having no idea how it works under the hood

Could you explian what is the exact deference between deferred execution and Lazy evaluation in C#?

Answer

In practice, they mean essentially the same thing. However, it's preferable to use the term deferred.

* Lazy means "don't do the work until you absolutely have to."
* Deferred means "don't compute the result until the caller actually uses it."

When the caller decides to use the result of an evaluation (i.e. start iterating through an IEnumerable<T>), that is precisely the point at which the "work" needs to be done (such as issuing a query to the database).

The term *deferred* is more specific/descriptive as to what's actually going on. When I say that I am lazy, it means that I avoid doing unnecessary work; it's ambiguous as to what that really implies. However, when I say that execution/evaluation is deferred, it essentially means that I am not giving you the real result at all, but rather a ticket you can use to claim the result. I defer actually going out and getting that result until you claim it.

Define what is Let clause?

Answer

In a query expression, it is sometimes useful to store the result of a sub-expression in order to use it in subsequent clauses. You can do this with the let keyword, which creates a new range variable and initializes it with the result of the expression you supply.

Consider:

var names = new string[] { "Dog", "Cat", "Giraffe", "Monkey", "Tortoise" };

var result =

from animalName in names

let nameLength = animalName.Length

where nameLength > 3

orderby nameLength

select animalName;

Explain what are LINQ compiled queries?

**Mid**

**[LINQ](https://www.fullstack.cafe/interview-questions/linq" \o "LINQ Interview Questions)**[38](https://www.fullstack.cafe/interview-questions/linq" \o "LINQ Interview Questions)

Answer

There may be scenario where we need to execute a particular query many times and repeatedly. LINQ allows us to make this task very easy by enabling us to create a query and make it compiled always. Benefits of Compiled Queries:

* Query does need to compiled each time so execution of the query is fast.
* Query is compiled once and can be used any number of times.
* Query does need to be recompiled even if the parameter of the query is being changed.

Consider:

static class MyCompliedQueries {

public static Func <DataClasses1DataContext, IQueryable <Person>> CompliedQueryForPerson =

CompiledQuery.Compile((DataClasses1DataContext context) = >from c in context.Persons select c);

}

When to use First() and when to use FirstOrDefault() with LINQ?

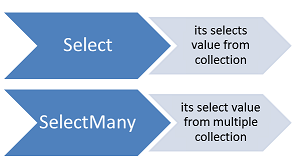
**Mid**

**[LINQ](https://www.fullstack.cafe/interview-questions/linq" \o "LINQ Interview Questions)**[38](https://www.fullstack.cafe/interview-questions/linq" \o "LINQ Interview Questions)

Answer

* Use First() when you know or expect the sequence to have at least one element. In other words, when it is an exceptional occurrence that the sequence is empty.
* Use FirstOrDefault() when you know that you will need to check whether there was an element or not. In other words, when it is legal for the sequence to be empty. You should not rely on exception handling for the check. (It is bad practice and might hurt performance).

First() will throw an exception if there's no row to be returned, while FirstOrDefault() will return the default value (NULL for all reference types) instead.



 A select operator is used to select value from a collection and SelectMany operator is used to selecting values from a collection of collection i.e. nested collection.

Here is a LINQ example you can try

List<string> animals = new List<string>() { "cat", "dog", "donkey" };

List<int> number = new List<int>() { 10, 20 };

var mix = number.SelectMany(num => animals, (n, a) => new { n, a });

the mix will have following elements in flat structure like

{(10,cat), (10,dog), (10,donkey), (20,cat), (20,dog), (20,donkey)}

**What are Anonymous data types?**  
  
Anonymous types are types that are generated by the compiler upon runtime. We don’t need to specify a name when we create an anonymous type. Only properties names are created and values assigned to them at runtime.  
  
In LINQ, Anonymous types are used to also save intermediate results. However, Anonymous types cannot implement interfaces, specify methods, or define static methods.  
  
All defined properties must be initialised and only public fields can be defined.  
  
**5. What is an Anonymous function?**  
  
An Anonymous function is a function without a name. In an Anonymous function, we only define parameters and write the code in curly braces.

**6. Explain Lambda Expression.**  
  
Lambda expression is a shortcut to writing delegates. Lambda expression is used to write inline functions that can be passed as arguments to a function or returned as arguments from a function.  
  
The syntax of a Lambda expression is:  
We specify the input parameters on the left, the lambda operator in the middle and expression or statement block on the right side.  
  
**7. What is meant by Action in LINQ?**  
  
Action refers to the general delegates belonging to the base class library of .NET. We can store only methods with input parameters and void return types in Action. Upto 16 parameters can be specified.

**12. What is PLINQ?**  
  
PLINQ is Parallel Language Integrated Query. In PLINQ, a query can be executed using many processors. With PLINQ, Software can be made scalable across environments during execution. Apart from supporting all operators in LINQ, PLINQ also runs many LINQ queries simultaneously as well.  
  
**13. What is the difference between the Take and Skip clause?**  
  
The Take clause is used to return only a specified number of elements. The Skip clause on the other hand, skips a specified number of elements and returns the rest only.

How to multiply by 10 , each element of an integer Array by using LINQ ?

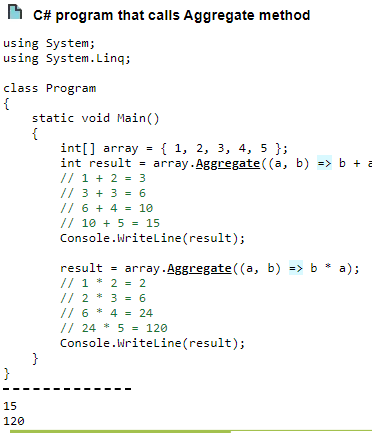
int[] number = { 1, 2, 3, 4, 5 };var result = number.Select(n => n \* 10);foreach (var item in result){Console.WriteLine(item);}

class Program{static void Main(string[] args){int[] mynumber = { 1,2,3,4,5,6,7,8,9,10};mynumber = mynumber.Select(b => b \* 10).ToArray();for (int i = 0; i < 10; i++){Console.Write(mynumber[i]);Console.WriteLine();}Console.ReadKey();}}

Can you provide a Example of Cross Product Join in LINQ ?

//Cross Product Join var players = new[] { "Tom", "Jay", "Mary" }.AsQueryable();IEnumerable query =from name1 in playersfrom name2 in playersselect name1 + " vs " + name2;query.Dump();

Aggregate is a declarative way of applying an accumulator function to a collection of elements. This means you can multiply or add all elements together. More complex accumulator functions can be used.



What is the difference between Count() and LongCount extension methods in LINQ?

1-LongCount() has a greater range than Count(). long.MinValue = -9223372036854775808 long.MaxValue = 9223372036854775807 DotNet Framework type is System.Int64 2-While count() DotNet Framework type is System.Int32 long.MinValue = -2,147,483,648 long.MaxValue = 2,147,483,647 3-So, if you want to count something which is quite big then use LongCount() extension method otherwise use Count()

How we can truncate a table Using LINQ?

EmpDataContext empobj= new EmpDataContext (); empobj.ExecuteCommand("TRUNCATE TABLE TABLENAME");

How to Fetch Different Values in LINQ?

Example:var Query= from i in db.tbl select new{i.id,i.name....};

LINQ – Lambda Expression vs Query Expression

As you’re probably aware of already, LINQ comes in two flavours – using Lambda expressions and using SQL-like query expressions:

|  |  |
| --- | --- |
|  | Func<int, bool> isEven = i => i % 2 == 0;  int[] ints = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };    // using Query expression  var evensQuery = from i in ints where isEven(i) select i;  // using Lambda expression  var evensLambda = ints.Where(isEven); |

Both yields the same result because query expressions are translated into their lambda expressions before they’re compiled. So performance-wise, there’s no difference whatsoever between the two.

Which one you should use is mostly personal preference, many people prefer lambda expressions because they’re shorter and more concise, but personally I prefer the query syntax having worked extensively with SQL. With that said, it’s important to bear in mind that there are situations where one will be better suited than the other.

// using lambda expression

    var lambda = people.Join(pets,              // outer sequence

                             person => person,  // inner sequence key

                             pet => pet.Owner,  // outer sequence key

                             (person, pet) =>

                                 new { OwnerName = person.Name, Pet = pet.Name });

    // using query expression

    var query = from person in people

                join pet in pets on person equals pet.Owner

                select new { OwnerName = person.Name, Pet = pet.Name };

}

There are a number of methods that are only available with the Lambda expression, Single(), Take(), Skip(), First() just to name a few. Although you can mix and match the two by calling the Lambda-only methods at the end of the query: