#1.Explain what is the difference between Skip() and SkipWhile() extension method?

Ans-

The Skip() and SkipWhile() methods are both used in LINQ.
Skip(n) skips the first n elements in the sequence.
SkipWhile(predicate) skips elements in the sequence while the specified predicate is true, and once the predicate becomes false, it stops skipping and returns the remaining elements.

#2 Explain what is the difference between Statement Lambda and Expression Lambda?

Statement Lambdas are enclosed in braces and can contain multiple statements. They use the => operator to separate the parameter list from the body of the lambda.

Expression Lambdas are shorter and more concise. They consist of a single expression and use the => operator to separate the parameter list from the body of the lambda.

#3 Explain how you can retrieve a single row with LINQ?

Can use the **First(), FirstOrDefault(), Single(), or SingleOrDefault()** methods.

#4 What difference is between FirstOrDefault() and First() selector methods in the LINQ?

The difference between FirstOrDefault() and First() in LINQ is that First() will throw an exception if no matching element is found, while FirstOrDefault() will return the default value for the element type if no match is found.

#5 What are SelectMany() and Select() in LINQ?

SelectMany() is used to project each element of a sequence to an IEnumerable and flatten the resulting sequences into one sequence.

Select() is used to transform elements in a collection by applying a function to each element and returning a new collection of the transformed elements.

#6 What are the Quantifier Operators?

Quantifier operators in LINQ are **All** ,**ANY** and **Contains**
All checks if all elements in a sequence satisfy a condition.
Any checks if any element in a sequence satisfies a condition.
Contains checks if the element is present in or not.

#7 What are the advantages of Deferred Execution?

1. Improved performance.
2.Reduced memory consumption.
3.It allows LINQ queries to be optimized and executed only when the results are actually needed.

#8 How will you count the elements of a collection or a list

Using the Count() method.

#9 How would you use LINQ to join two or more lists or tables together?

JOIN() METHOD.

#10 What steps would you take to use LINQ to count the number of items in a collection that match a certain condition?

Count Method .

#11 How can you use LINQ to perform a Union or Intersect operation on two lists?

```
List<int> list1 = new List<int> { 1, 2, 3, 4, 5 };
List<int> list2 = new List<int> { 3, 4, 5, 6, 7 };

// Union
var unionResult = list1.Union(list2);
Console.WriteLine("Union of list1 and list2:");
foreach (var item in unionResult)
{
```

```
    Console.Write(item + " ");
}
```

#12 Explain the difference between First, FirstOrDefault, Single and SingleOrDefault in LINQ and when you would use each.

First and FirstOrDefault return the first element of a sequence, but First throws an exception if no element is found, while FirstOrDefault returns the default value for the element type.

Single and SingleOrDefault are similar, but they ensure that there is exactly one matching element in the sequence. Single throws an exception if there is not exactly one element, while SingleOrDefault returns the default value.

#13 How would you use LINQ to check if a collection contains a specific item?

Can use the **Contains() method**

#14 How do you use LINQ to retrieve distinct items from a collection and when would you want to do this?

Distinct() method

-When we want to retrieve distinct items from a collection.

#15 Explain the difference between Any and All in LINQ and when you would use each

The **Any** operator in LINQ checks if any element in a sequence satisfies a given condition.
The **All** operator checks if all elements in a sequence satisfy a given condition.

16 what will be the output:

```
                IList<Student> studentList = new List<Student>() {
    new Student() { StudentID = 1, StudentName = "John", Age = 18,
StandardID = 1 } ,
```

```
    new Student() { StudentID = 2, StudentName = "Steve",  Age = 21,
StandardID = 1 } ,
    new Student() { StudentID = 3, StudentName = "Bill",  Age = 18,
StandardID = 2 } ,
    new Student() { StudentID = 4, StudentName = "Ram" , Age = 20,
StandardID = 2 } ,
    new Student() { StudentID = 5, StudentName = "Ron" , Age = 21 }
      };

      IList<Standard> standardList = new List<Standard>() {
            new Standard(){ StandardID = 1, StandardName="Standard 1"},
            new Standard(){ StandardID = 2, StandardName="Standard 2"},
            new Standard(){ StandardID = 3, StandardName="Standard 3"}
      };


      a.) var studentNames = studentList.Where(s => s.Age > 18)
                               .Select(s => s)
                               .Where(st => st.StandardID > 0)
                               .Select(s => s.StudentName);
```

Output :
Steve
Ram

```
      b.) var teenStudentsName = from s in studentList
                       where s.age > 12 && s.age < 20
                       select new { StudentName = s.StudentName };

            teenStudentsName.ToList().ForEach(s =>
Console.WriteLine(s.StudentName));
```

Output:
John
Bill

```
      c.) var studentsGroupByStandard = from s in studentList
                               group s by s.StandardID into sg
                               orderby sg.Key
                                     select new { sg.Key, sg };


                  foreach (var group in studentsGroupByStandard)
```

```
                {
                      Console.WriteLine("StandardID {0}:", group.Key);

                      group.sg.ToList().ForEach(st =>
Console.WriteLine(st.StudentName ));
                }
```

Output :

StandardID 0:
Ron
StandardID 1:
John
Steve
StandardID 2:
Bill
Ram

```
      d.) IList<int> intList = new List<int>() { 7, 10, 21, 30, 45, 50, 87
};
                  IList<string> strList = new List<string>() { null, "Two",
"Three", "Four", "Five" };

                  Console.WriteLine("1st Element which is greater than 250
in intList: {0}",

intList.First( i > 250));

                  Console.WriteLine("1st Even Element in intList: {0}",

strList.FirstOrDefault(s => s.Contains("T")));
```

Output :
Runtime exception - Sequence contains no matching element

```
      e.) IList<string> strList1 = new List<string>() { "Two", "Three",
"Four", "Five" };
                  IList<string> strList2 = new List<string>() { null,
"Two", "Three", "Four", "Five" };
```

```
                Console.WriteLine(strList1.LastOrDefault(s =>
s.Contains("T")));
                Console.WriteLine(strList2.LastOrDefault(s =>
s.Contains("T")));
```

Output :

a) -Three

b)Run-time exception : Object reference not set to an instance of an object.

```
     f.) IList<string> strList1 = new List<string>() { "Two", "Three",
"Four", "Five" };
                IList<string> strList2 = new List<string>() { null,
"Two", "Three", "Four", "Five" };
                IList<string> strList3 = new List<string>();

                Console.WriteLine(strList1.Single(s => s.Contains("T")));
                Console.WriteLine(strList2.SingleOrDefault(s =>
s.Contains("T")));
                Console.WriteLine(strList3.SingleOrDefault(s =>
s.Contains("T")));
```

Run-time exception : More than one matching element