LINQ

1. **LINQ**

Language-Integrated Query (LINQ) is a powerful set of technologies based on the integration of query capabilities directly into the C# language. LINQ Queries are the first-class language construct in C# .NET, just like classes, methods, events. The LINQ provides a consistent query experience to query objects (LINQ to Objects), relational databases (LINQ to SQL), and XML (LINQ to XML).

LINQ (Language Integrated Query) is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats. It is integrated in C# or VB, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.

1. **=>**
2. **Where**

The Where operator (Linq extension method) filters the collection based on a given criteria expression and returns a new collection. The criteria can be specified as lambda expression or Func delegate type.

1. **OfType**

The OfType operator filters the collection based on the ability to cast an element in a collection to a specified type.

1. **ThenBy**

The ThenBy and ThenByDescending extension methods are used for sorting on multiple fields.

The OrderBy() method sorts the collection in ascending order based on specified field. Use ThenBy() method after OrderBy to sort the collection on another field in ascending order. Linq will first sort the collection based on primary field which is specified by OrderBy method and then sort the resulted collection in ascending order again based on secondary field specified by ThenBy method.

The same way, use ThenByDescending method to apply secondary sorting in descending order.

1. **GroupBy, ToLookUp**

The GroupBy operator returns a group of elements from the given collection based on some key value. Each group is represented by IGrouping<TKey, TElement> object. Also, the GroupBy method has eight overload methods, so you can use appropriate extension method based on your requirement in method syntax.

ToLookup is the same as GroupBy; the only difference is GroupBy execution is deferred, whereas ToLookup execution is immediate. Also, ToLookup is only applicable in Method syntax. **ToLookup is not supported in the query syntax.**

1. **Join**

The Join operator operates on two collections, inner collection & outer collection. It returns a new collection that contains elements from both the collections which satisfies specified expression. It is the same as inner join of SQL.

1. **GroupJoin**

We have seen the Join operator in the previous section. The GroupJoin operator performs the same task as Join operator except that GroupJoin returns a result in group based on specified group key. The GroupJoin operator joins two sequences based on key and groups the result by matching key and then returns the collection of grouped result and key.

As you can see in the first overload method takes five input parameters (except the first 'this' parameter): 1) outer 2) inner 3) outerKeySelector 4) innerKeySelector 5) resultSelector. Please notice that resultSelector is of Func delegate type that has second input parameter as IEnumerable type for inner sequence.

1. **Select**

The Select operator always returns an IEnumerable collection which contains elements based on a transformation function. It is similar to the Select clause of SQL that produces a flat result set.

1. **All, Any**

The All operator evalutes each elements in the given collection on a specified condition and returns True if all the elements satisfy a condition.

Any checks whether any element satisfy given condition or not? In the following example, Any operation is used to check whether any student is teen ager or not.

1. **Contains**

The Contains operator checks whether a specified element exists in the collection or not and returns a boolean.

The Contains() extension method has following two overloads. The first overload method requires a value to check in the collection and the second overload method requires additional parameter of IEqualityComparer type for custom equalality comparison.

1. **Aggregate**

The aggregation operators perform mathematical operations like Average, Aggregate, Count, Max, Min and Sum, on the numeric property of the elements in the collection.

1. **Avarage**

Average extension method calculates the average of the numeric items in the collection. Average method returns nullable or non-nullable decimal, double or float value.

The following example demonstrate Agerage method that returns average value of all the integers in the collection.

1. **Count**

The Count operator returns the number of elements in the collection or number of elements that have satisfied the given condition.

1. **Max**

The Max() method returns the largest numeric element from a collection.

The following example demonstrates Max() on primitive collection.

1. **Sum**

The Sum() method calculates the sum of numeric items in the collection.

The following example demonstrates Sum() on primitive collection.

1. **ElementAt, ElementAtOrdefault**

Element operators return a particular element from a sequence (collection).

The following table lists all the Element operators in LINQ.

The ElementAt() method returns an element from the specified index from a given collection. If the specified index is out of the range of a collection then it will throw an Index out of range exception. Please note that index is a zero based index.

The ElementAtOrDefault() method also returns an element from the specified index from a collaction and if the specified index is out of range of a collection then it will return a default value of the data type instead of throwing an error.

1. **First, FirstOrDefault**

The First and FirstOrDefault method returns an element from the zeroth index in the collection i.e. the first element. Also, it returns an element that satisfies the specified condition.

First and FirstOrDefault has two overload methods. The first overload method doesn't take any input parameter and returns the first element in the collection. The second overload method takes the lambda expression as predicate delegate to specify a condition and returns the first element that satisfies the specified condition.

1. **Last, LastOrDefault**

The Last() and LastOrDefault() extension methods returns the last element from the collection.

The Last() method returns the last element from a collection, or the last element that satisfies the specified condition using lambda expression or Func delegate. If a given collection is empty or does not include any element that satisfied the condition then it will throw InvalidOperation exception.

The LastOrDefault() method does the same thing as the Last() method. The only difference is that it returns default value of the data type of a collection if a collection is empty or doesn't find any element that satisfies the condition.

1. **Single, SingleOrDefault**

Single and SingleOrDefault have two overload methods. The first overload method doesn't take any input parameter and returns a single element in the collection. The second overload method takes the lambda expression as a predicate delegate that specifies the condition and returns a single element that satisfies the specified condition.

Single() returns the only element from a collection, or the only element that satisfies the specified condition. If a given collection includes no elements or more than one elements then Single() throws InvalidOperationException.

The SingleOrDefault() method does the same thing as Single() method. The only difference is that it returns default value of the data type of a collection if a collection is empty, includes more than one element or finds no element or more than one element for the specified condition.

1. **SequenceEquel**

There is only one equality operator: SequenceEqual. The SequenceEqual method checks whether the number of elements, value of each element and order of elements in two collections are equal or not.

If the collection contains elements of primitive data types then it compares the values and number of elements, whereas collection with complex type elements, checks the references of the objects. So, if the objects have the same reference then they considered as equal otherwise they are considered not equal.

1. **Concat**

The Concat() method appends two sequences of the same type and returns a new sequence (collection).

1. **DefaultEmpty**

The DefaultIfEmpty() method returns a new collection with the default value if the given collection on which DefaultIfEmpty() is invoked is empty.

Another overload method of DefaultIfEmpty() takes a value parameter that should be replaced with default value.

1. **Empty, Range, Repeat**

LINQ includes generation operators DefaultIfEmpty, Empty, Range & Repeat. The Empty, Range & Repeat methods are not extension methods for IEnumerable or IQueryable but they are simply static methods defined in a static class Enumerable.

The Empty() method is not an extension method of IEnumerable or IQueryable like other LINQ methods. It is a static method included in Enumerable static class. So, you can call it the same way as other static methods like Enumerable.Empty<TResult>(). The Empty() method returns an empty collection of a specified type as shown below.

The Range() method returns a collection of IEnumerable<T> type with specified number of elements and sequential values starting from the first element.

The Repeat() method generates a collection of IEnumerable<T> type with specified number of elements and each element contains same specified value.

1. **Distinct**

The Distinct extension method returns a new collection of unique elements from the given collection.

1. **Except**

The Except() method requires two collections. It returns a new collection with elements from the first collection which do not exist in the second collection (parameter collection).

1. **Intersect**

The Intersect extension method requires two collections. It returns a new collection that includes common elements that exists in both the collection.

1. **Union**

The Union extension method requires two collections and returns a new collection that includes distinct elements from both the collections.

1. **Skip, SkipWhile**

Partitioning operators split the sequence (collection) into two parts and return one of the parts.

The Skip() method skips the specified number of element starting from first element and returns rest of the elements.

As the name suggests, the SkipWhile() extension method in LINQ skip elements in the collection till the specified condition is true. It returns a new collection that includes all the remaining elements once the specified condition becomes false for any element.

The SkipWhile() method has two overload methods. One method accepts the predicate of Func<TSource, bool> type and other overload method accepts the predicate Func<TSource, int, bool> type that pass the index of an element.

1. **Take, TakeWhile**

Partitioning operators split the sequence (collection) into two parts and returns one of the parts.

The Take() extension method returns the specified number of elements starting from the first element.

The TakeWhile() extension method returns elements from the given collection until the specified condition is true. If the first element itself doesn't satisfy the condition then returns an empty collection.

The TakeWhile method has two overload methods. One method accepts the predicate of Func<TSource, bool> type and the other overload method accepts the predicate Func<TSource, int, bool> type that passes the index of element.