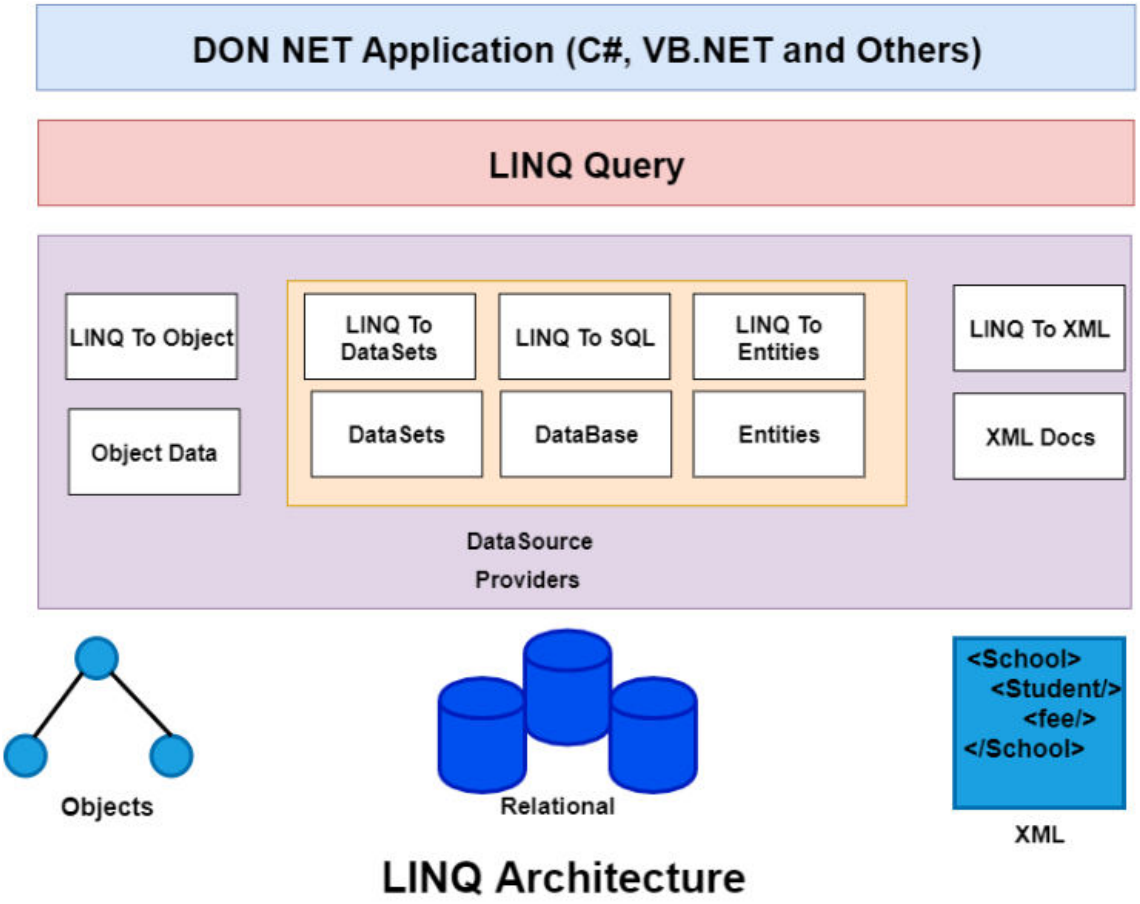
INTRODUCTION

What is LINQ ?

The full form of LINQ is 'Language Integrated Query,' and introduced in .NET Framework 3.5 to query the data from different sources of data such as collections, generics, XML documents, ADO.NET Datasets, SQL, Web Services, etc.

LINQ functionality can be achieved by importing the **System.Linq** namespace in our application.



LINQ Architecture

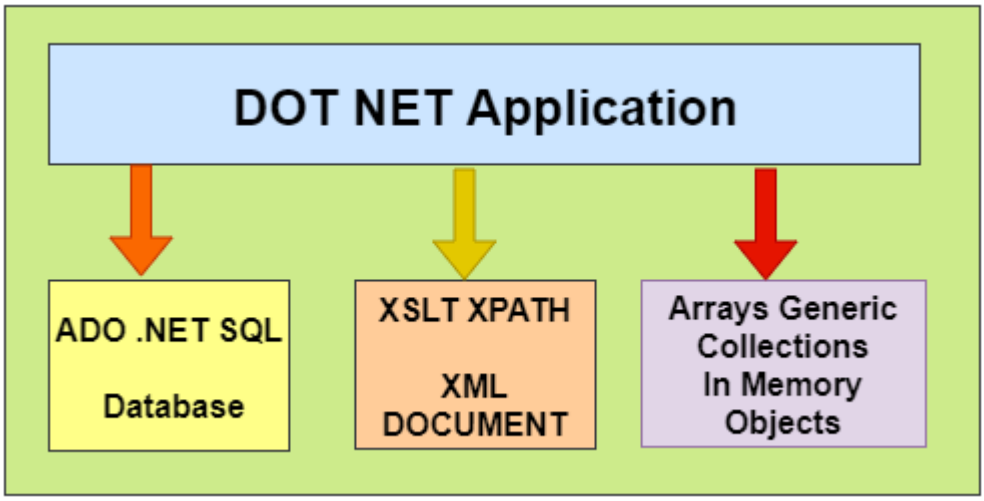
LINQ DataSource Provider :

The responsibility of the LINQ provider is to convert the LINQ Query into a format so that the data source can understand it.

Example: Here, we will take a scenario, let us say the application wants to fetch the data from SQL Database. In this case, LINQ Query will fit into the LINQ to SQL Provider. In this case, it will convert the LINQ Query into T-SQL so that the underlying database can understand.

Need of LINQ

It allows us to work with different data sources by using a standard, or in a unified coding style. Hence, we don't require to learn the different syntaxes to query for various data sources.



Advantages of LINQ

1. We do not need to learn new query language syntaxes for different sources of data because it provides the standard query syntax for the various data sources.
2. In LINQ, we have to write the Less code in comparison to the traditional approach. With the use of LINQ, we can minimize the code.
3. LINQ provides the compile-time error checking as well as intelligence support in Visual Studio. This powerful feature helps us to avoid run-time errors.
4. LINQ provides a lot of built-in methods that we can be used to perform the different operations such as filtering, ordering, grouping, etc. which makes our work easy.
5. The query of LINQ can be reused.

Disadvantages of LINQ

1. With the use of LINQ, it's very difficult to write a complex query like SQL.
2. It was written in the code, and we cannot make use of the Cache Execution plan, which is the SQL feature as we do in the stored procedure.
3. If the query is not written correctly, then the performance will be degraded.
4. If we make some changes to our queries, then we need to recompile the application and need to redeploy the dll to the server.

LINQ Syntax

Each Query is a combination of three things; they are:

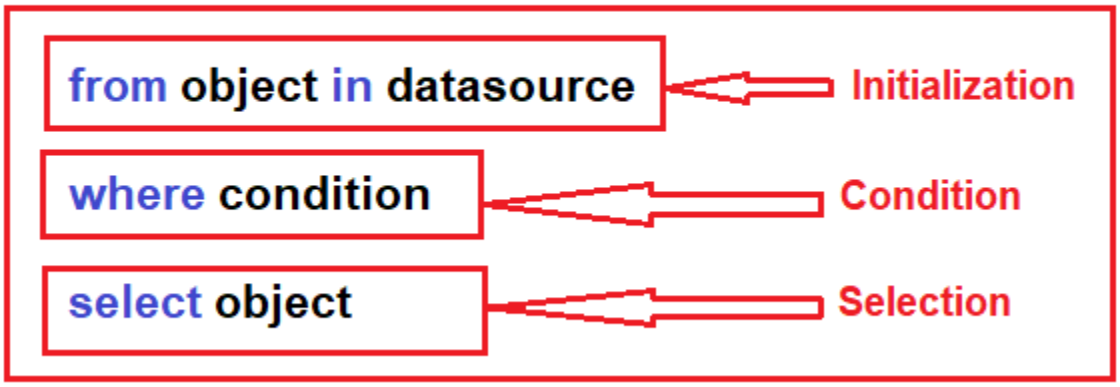
1. Initialization (to work with a particular data source)
2. Condition (where, filter, sorting condition)
3. Selection (single selection, group selection or joining)

LINQ has three ways to write the queries:

- Using Query Syntax
- Using Method Syntax
- Using Mixed Syntax

LINQ Query Syntax

Syntax of LINQ is as:




```
//LINQ Query using Query Syntax
var QuerySyntax = from obj in integerList
                  where obj > 5
                  select obj;

//Execution
foreach (var item in QuerySyntax)
{
    Console.Write(item + " ");
}
```

Initialization

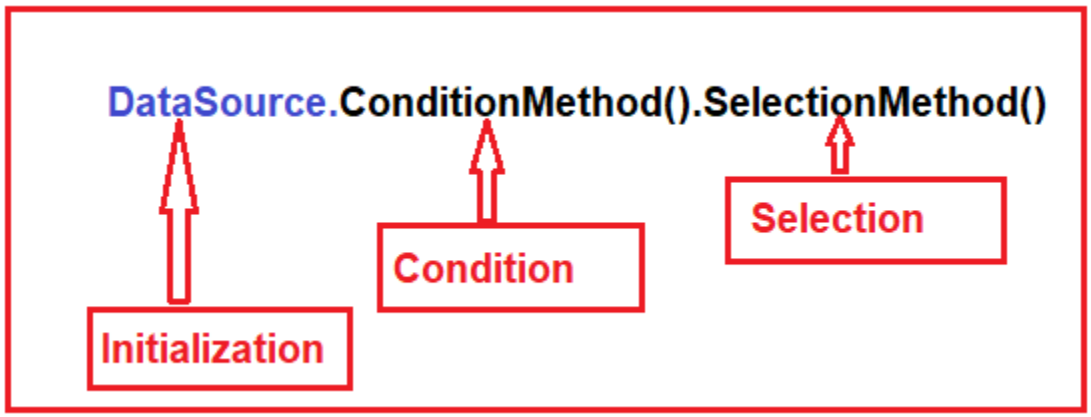
Condition

Selection

The order of the clauses in LINQ query will be like, as shown below:

From	[Identifier]
In	[Source Collection]
Let	[Expression]
Where	[Boolean Expression]
order by	[Expression]
Select	[Expression]
group by	[Expression]
into	

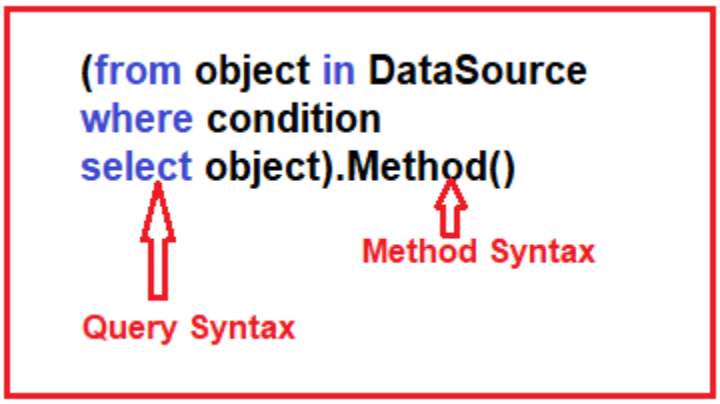
LINQ Method Syntax




```
//LINQ Query using Method Syntax
varMethodSyntax = integerList.Where(obj =>obj> 5).ToList();
```

LINQ Mixed Syntax

LINQ Mixed Syntax is a combination of both **Query** and **MethodSyntax**.



FUNCTIONS

Min () Function : .Min()

Max () Function : .Max()

Sum () Function : .Sum()

Count () Function : .Count()

is used to count the number of elements in the **list** or **collection**.

Aggregate () Function

Aggregate () function is used to perform the operation on each item of the list.

The **Aggregate ()** function performs the action on the first and second elements and then carry forward the result.

For the next operation, it will consider the previous result and the third element and then carry forwards, etc.

```
double Average = Num.Aggregate((a, b) => a * b);


Console.WriteLine("Product is {0}", Average); //Output 362880 (((((((1*2)*3)*4)*5)*6)*7)*8)*9)
```

LINQ Sorting Operators

Operator	Description	Query Exg
OrderBy	This operator will sort the values in ascending order.	student.OrderBy(x => x.Name);
OrderByDescending	This operator will sort the values in descending order.	student.OrderByDescending(x => x.Name);
ThenBy	ThenBy sorting operator is used to implement the sorting on multiple fields, and by default, ThenBy operator will sort the collection of items in **ascending order.** Generally, in LINQ ThenBy operator is used along with OrderBy operator to implement the sorting on multiple fields in the list/collection.	student.OrderBy(x => x.Name) .ThenBy(x => x.RoleId);
ThenByDescending	ThenByDescending operator is used to implement the sorting on multiple fields in the list/ collection, and by default, ThenByDescending operator will sort the list of items in **descending order**. In LINQ, we use the ThenByDescending operator along with OrderBy operator.	student.OrderBy(x => x.Name) .ThenByDescending(x => x.RoleId);
Reverse	This operator is used to reverse the order of elements in the collection.	Not applicable

LINQ PARTITION OPERATORS

Partition Operators are used to partition the list/collections items into two parts and return one part of the list items.

The table shows more detailed information related to partition operators in LINQ.

Operator	Description	Query-Exg
TAKE	This operator is used to return the specified number of elements in the sequence.	countries.Take(3);
TAKEWHILE	This operator is used to return the elements in the sequence which satisfy the specific condition. It does not check whole collection, it stops when condition fails first.	countries.TakeWhile(x => x.StartsWith("U"));
SKIP	This operator is used to skip the specified number of elements from start in a sequence and return the remaining elements.	countries.Skip(3);
SKIPWHILE	This operator is used to skip the elements in a sequence based on the condition, which is defined as true. It skips the element till the conditions satisfy, once the condition does not satisfy it will take all the element after that.	countries.SkipWhile(city=> city.EndsWith('a'));

CONVERSION OPERATORS

ToLookup() Method :

it is used to extract a set of key/value pairs from the source.

```
List<Employee> objEmployee = new List<Employee>()
            {
                new Employee(){ Name="Akshay Tyagi", Department="IT", Country="India"},
                new Employee(){ Name="Vaishali Tyagi", Department="Marketing", Country="Australia"},
                new Employee(){ Name="Arpita Rai", Department="HR", Country="China"},
                new Employee(){ Name="Shubham Ratogi", Department="IT", Country="USA"},
                new Employee(){ Name="Himanshu Tyagi", Department="IT", Country="Canada"}
            };
```

```
var Emp = objEmployee.ToLookup(x => x.Department);
```

we grouped the Employee by Department using the **ToLookup** method. Since ToLookup produces the key/value pair so that we used it in the **foreach** loop and the inner loop extracts the values based on the Key passed as input.

```
Grouping Employees by Department
--------------------------------
IT
        Akshay Tyagi    IT      India
        Shubham Ratogi  IT      USA
        Himanshu Tyagi  IT      Canada
Marketing
        Vaishali Tyagi  Marketing       Australia
HR
        Arpita Rai      HR      China
```

```
// here we will get collection of group of key/value pair.
// here key will be the thing (i.e Department) on which we are grouping.
// value will be the collection of elements from datasource which falls under that specified group
in lookup().
```

Cast() Method :

Cast operator is used to cast/convert all the elements present in a collection into a specified data type of new collection.

OfType() Method :

OfType() operator is used to return the element of the specific type, and another element will be ignored from the list/collection.

```
IEnumerable<string> result = obj.OfType<string>();
```

AsEnumrable() Method :

AsEnumerble() method is used to convert the specific type of given list to its IEnumerable() equivalent type.

```
var result = numarray.AsEnumerable();
```

ToList() Method :

ToList operator takes the element from the given source, and it returns a new List.

ToArray() Method :

ToArray() operator in LINQ is used to convert the input elements in the collection to an Array.

ToDictionary() Method :

ToDictionary() Method is used to convert the items of list/collection(IEnumerable<T>) to new dictionary object (Dictionary<TKey,TValue>) and it will optimize the list/collection items by required values only.

```
//Create a object objStudent of Student class and add the information of student in the List
        List<Student> objStudent = new List<Student>()
        {
           new Student() { Id=1,Name = "Vinay Tyagi", Gender = "Male",Location="Chennai" },
           new Student() { Id=2,Name = "Vaishali Tyagi", Gender = "Female", Location="Chennai" },
           new Student() { Id=3,Name = "Montu Tyagi", Gender = "Male",Location="Bangalore" },
           new Student() { Id=4,Name = "Akshay Tyagi", Gender = "Male", Location ="Vizag"},
           new Student() { Id=5,Name = "Arpita Rai", Gender = "Male", Location="Nagpur"}
        };
   /*here with the help of ToDictionary() method we are converting the colection
   of information in the form of dictionary and will fetch only the required information*/
           var student = objStudent.ToDictionary(x => x.Id, x => x.Name);
   //foreach loop is used to print the information of the student
           foreach (var stud in student)
           {
              Console.WriteLine(stud.Key + "\t" + stud.Value);
           }
```

```
1       Vinay Tyagi
2       Vaishali Tyagi
3       Montu Tyagi
4       Akshay Tyagi
5       Arpita Rai
```

LINQ Element Operators

Operators	Description
First	It returns the first element in sequence or from the collection based on the condition.
FirstOrDefault	It is the same as **First**, but it returns the default value in case when no element is found in the collection.
Last	It returns the last element in sequence or the last element based on the matching criteria.
ElementAt	It returns an element from the list based on the specific index position.
ElementAtOrDefault	It is the same as **ElementAt**, but it returns the default value in case if no element is present at the specified index of the collection.
Single	It returns the single specific element from the collection. In case, if the Single() method found more than one element in collection or no element in the collection, then it will throw the "**InvalidOperationException**" error.
SingleOrDefault	It is the same as **Single**, but it returns the default value in case if no element is found in the collection.
DefaultfEmpty	It returns the default value in case if the list or collection contains empty or null values.

Single() / SingleOrDefault()	**First () / FirstOrDefault()**
Single() - There is exactly 1 result, an exception is thrown if no result is returned or more than one result. **SingleOrDefault()** – Same as Single(), but it can handle the null value.	**First()** - There is at least one result, an exception is thrown if no result is returned. **FirstOrDefault()** - Same as First(), but not thrown any exception or return null when there is no result.
Single() asserts that one and only one element exists in the sequence.	First() simply gives you the first one.
When to use Use Single / SingleOrDefault() when you sure there is only one record present in database or you can say if you querying on database with help of primary key of table.	When to use Developer may use First () / FirstOrDefault() anywhere, when they required single value from collection or database.
Single() or SingleOrDefault() will generate a regular TSQL like "SELECT ...".	The First() or FirstOrDefault() method will generate the TSQL statment like "SELECT TOP 1..."
In the case of Fist / FirstOrDefault, only one row is retrieved from the database so it performs slightly better than single / SingleOrDefault. such a small difference is hardly noticeable but when table contain large number of column and row, at this time performance is noticeable.	

GROUPING OPERATOR

GroupBy() Method :

GroupBy operator is used to grouping the list/collection items based on the specified value of the key, and it returns a collection of **IGrouping<key, Values>**.

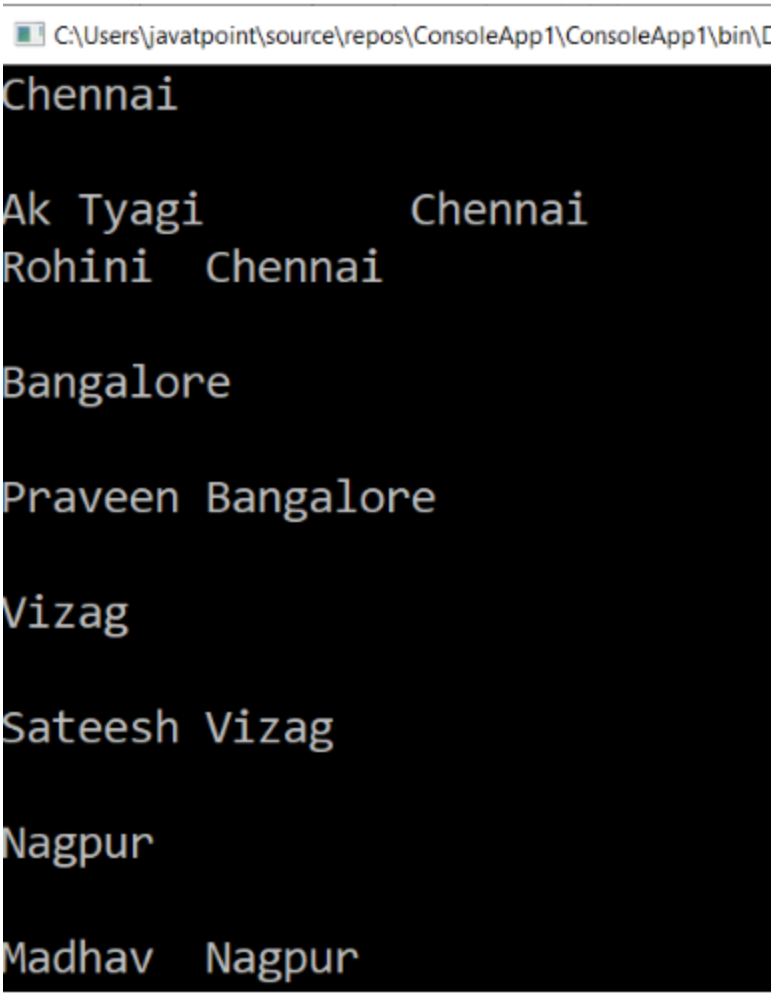
```
//Create an object 'objStudent' of the list of the student
        List<Student> objStudent = new List<Student>
  ()
  {
  new Student() { Name = "Ak Tyagi", Gender = "Male",Location="Chennai" },
  new Student() { Name = "Rohini", Gender = "Female", Location="Chennai" },
  new Student() { Name = "Praveen", Gender = "Male",Location="Bangalore" },
  new Student() { Name = "Sateesh", Gender = "Male", Location ="Vizag"},
  new Student() { Name = "Madhav", Gender = "Male", Location="Nagpur"}
  };
 // here with the help of GrouBy we will fetch the student on the base of location
  var student1 = objStudent.GroupBy(x => x.Location);
  foreach (var sitem in student1)
  {
 // WriteLine() function here count the number of student
  Console.WriteLine(sitem.Key, sitem.Count());
  Console.WriteLine();
  foreach (var stud in sitem)
  {
  //Console.WriteLine(stud.Name + "\t" + stud.Location) show the information of the student on the base of the location
  Console.WriteLine(stud.Name + "\t" + stud.Location);
  }

  Console.WriteLine();


  }
```

We can also do :

```
var student = from std in objStudent
group std by std.Location;
```


C:\Users\javatpoint\source\repos\ConsoleApp1\ConsoleApp1\bin\D
Chennai
Ak Tyagi Chennai
Rohini Chennai
Bangalore
Praveen Bangalore
Vizag
Sateesh Vizag
Nagpur
Madhav Nagpur

JOIN OPERATORS

LINQ Inner Join :

```
var result = from d in objDept
join e in objEmp
on d.DepId equals e.DeptId
select new
{
EmployeeName = e.Name,
DepartmentName = d.DepName
};
```

we are trying to get the elements from "**objEmp**", "**objDept**" collections based on the matching "**DeptId**" column values.

LINQ Left Join :

```
var courseInfo = from course in odDataContext.COURSEs
join student in odDataContext.STUDENTs
on course.course_id equals  student.course_id into studentInfo
from students in studentInfo.DefaultIfEmpty()
select new
{
   STUDENTNAME = (students.student_name == null)? NULL":students.student_name,
   STUDENTCITY = (students.student_city == null)? "NULL":students.student_city,
   COURSENAME = course.course_name,
   COUSREDESCRIPTION = course.course_desc
};
```

LINQ Cross Join :

```
var studentInfo = from student in odDataContext.STUDENTs
from course in odDataContext.COURSEs
select new { student.student_name, student.student_city, course.course_name, course.course_desc };
```

In the above code we are doing a cross-join on both the STUDENT table and the COURSE table. We get all rows from both tables and the total rows are STUDENT table rows * COURSE table rows.

LINQ Set Operations

Operators	Description
UNION	Union operator combines multiple collections into single collection and return the resultant collection with unique element. `var result = count1.Union(count2);`
INTERSECT	It returns the element in a sequence, which is common in both the input sequence. `var result = count1.Intersect(count2);`

DISTINCT	It removes the duplicate elements from the collection and returns the collection with unique values.
	`IEnumerable<int> result = numbers.Distinct();`
	`IEnumerable<string> result = countries.Distinct(StringComparer.OrdinalIgnoreCase);`
	
	we applied a **Distinct** method with **StringComparer.OrdinalIgnoreCase** case property, otherwise, it will perform the operation with case sensitivity on"**countries**" collection, and it will treat "**India**" and "india" as different.

EXCEPT	It returns the sequence element from the first input sequence, which is not present in the second input sequence. `var result = arr1.Except(arr2);`

SequenceEqual Method

SequenceEqual method is used to compare the sequence of two collections that are equal or not.

It determines two sequences whether they are equal or not by comparing the elements in a pair-wise manner, and two sequences contain the equality number of the element or not.

The LINQ **SequenceEqual** method will return the Boolean value true in case two sequence elements are equal, and all the elements match in both the sequences; otherwise, it will throw false.

```
 string[] array1 = { "welcome", "to", "tutlane", "com" };
 string[] array2 = { "welcome", "TO", "Noida", "com" };
 string[] array3 = { "welcome", "to", "noida" };
 string[] array4 = { "WELCOME", "TO", "NOIDA" };
nce.Equal() method is used to check if both te sequences are
al or not
 var res1 = array1.SequenceEqual(array2);
 var res2 = array1.SequenceEqual(array2, StringComparer
     .OrdinalIgnoreCase);
 var res3 = array1.SequenceEqual(array3);
 var res4 = array3.SequenceEqual(array4, StringComparer
     .OrdinalIgnoreCase);
 Console.WriteLine("Result1: {0}", res1);
 Console.WriteLine("Result2: {0}", res2);
 Console.WriteLine("Result3: {0}", res3);
 Console.WriteLine("Result4: {0}", res4);
```

```
Result1: False
Result2: False
Result3: False
Result4: True
```

LINQ Concat Method

Concat method or operator is used to concatenate or append the two collection elements into a single collection, and it does not remove the duplicates from two sequences.

```
var result = arr1.Concat(arr2);
```

Generation Operations

Method	Description
DefaultfEmpty	If the collection contains the empty elements, then it will return the default value.
Empty	It returns the empty collection of sequences.

Range	It returns the collection that contains a sequence of numbers.
	
	Range method or operator is used to generate the sequence of integer or numbers based on the specified values of the start index and end index.
	
	`//Enumerable.Range() method iterate upto 10 numbers from 100 to 109`
	`IEnumerable<int> obj1 = Enumerable.Range(100, 10);`
	`//foreach loop used to print the numbers from 100 to 109`

Repeat	It returns a collection that contains the one repeated value-based on a specified length.
	
	Repeat method or operator is used to generate the collection with the same number as the repeated times based on the specified index value.
	
	`//IEnumerable.Repeat() method iterate upto 10 numbers`
	`IEnumerable<int> obj1 = Enumerable.Repeat(100, 10);`
	`//foreach loop is used to print the number 100 upto 10 times.`

LINQ with Sql

SQL Select Query

```
EmployeeDBDataContext db1 = new EmployeeDBDataContext();

var result = from ed in db.EmployeeDetails



        select new

        {

           EmployeeName = ed.EmpName,

           Location = ed.Location,

           Gender = ed.Gender

        };
```

EmployeeName	Location	Gender
Vaishali	Noida	Female
Akshay	Noida	Male

LINQ to SQL Inner Join

```
var result = from ed in db.Employeedetail

        join d in db.Department on ed.DeptId equals d.DeptId

        where d.DeptName.Equals("software")

        select new

        {

           Name = ed.EmpName,

           Location = ed.Location,

           Gender = ed.Gender,

           Department = d.DeptName

        };
```

Name	Location	Gender	Department
Suresh Dasari	Chennai	Male	Software
Praveen Alavala	Guntur	Male	Software

Note **:** In entity framework we use Db<Set> class_names which is some kind of collection which represent table in database.

So, we can query this class_name using linq syntax.

A DbSet represents the collection of all entities in the context, or that can be queried from the database, of a given type