**===Linq Session Points to remember**

IOrderedEnumerable implements IEnumerable Interface, so we can use IOrderedEnumerable or IEnumerable <> incase of order by collection retrieve

ToLookUp is just like dictionary, a lookup is a collection of key/value pairs. a dictionary cannot contain keys with identical values, where as a lookup can.

OfType - return only the elements of type x(i.e. in which type u r casting like int,string etc), where as Cast - will try to cast all the elements into type x. if some of them are not from this type you will get InvalidCastException

**------------------------------------------------------------------------------------------------------------------------------**

**===Into and Let in LINQ (Let vs. Into)**

The Into keyword allows creating a temporary variable to store the results of a group, join, or select clause into a new variable.

var em = from e in emp

group e by new { e.DeptId }

into gEmp

where gEmp.Count() > 1

select new { gEmp.Key.DeptId, salary = gEmp.Sum(t => t.Salary) };

**Note**: Into is used when you want to perform an operation on grouped data.

var teenAgerStudents = from s in studentList

where s.age > 12 && s.age < 20

select s

into teenStudents

where teenStudents.StudentName.StartsWith("B")

select teenStudents;

In the above query, the 'into' keyword introduced a new range variable teenStudents, so the first range variable s goes out of scope. You can write a further query after the into keyword using a new range variable.

**Let**

The Let keyword allows storing the results of a query which can be used in a subsequent query; i.e., it creates a new variable and initializes it with the result of the expression you supply.

var em = from e in emp

group e by new { e.Salary, e.Id }

into gEmp

let avgsal = (gEmp.Sum(t => t.Salary) / gEmp.Count())

where gEmp.Key.Salary == avgsal

select new { gEmp.Key.Salary, gEmp.Key.Id };

The above query is used to find out employee(s) having salary more than avgSalary. The Let keyword allows to create a new variable avgsal that is used in further operations.

The following example use 'let' to introduce new variable 'lowercaseStudentName' that will be then used in everywhere. Thus, **let** keyword to make the query more readable.

var lowercaseStudentNames = from s in studentList

let lowercaseStudentName = s.StudentName.ToLower()

where lowercaseStudentName.StartsWith("r")

select lowercaseStudentName;

**Let vs. Into**

Most people find it difficult to decide which one to use when designing a LINQ query.

Into – Hides the previous variable when used in a query, as you see in the above example. Which means it hides the previous range variable and creates a temporary range variable which you can use in further operations.

Let – Doesn’t hide the previous variable and creates a new variable. Which means you create a new variable and you can also use the previous variable, so you can use both in further operations.