**The following operators belong to Set operators category**  
Distinct  
Union  
Intersect  
Except   
  
   
  
We discussed **Distinct**operator in [Part 26](http://csharp-video-tutorials.blogspot.com/2014/08/part-26-set-operators-in-linq.html). In this video we will discuss **Union**, **Intersect** and **Except**operators.  
  
**Union**combines two collections into one collection while removing the duplicate elements.  
  
**Example 1:**numbers1 and numbers2 collections are combined into a single collection. Notice that, the duplicate elements are removed.

int[] numbers1 = { 1, 2, 3, 4, 5 };

int[] numbers2 = { 1, 3, 6, 7, 8 };

var result = numbers1.Union(numbers2);

foreach (var v in result)

{

    Console.WriteLine(v);

}

**Output:**   
union in linq   
  
When **comparing elements**, just like **Distinct()**method, **Union(), Intersect()**and **Except()**methods work in a slightly different manner with **complex types**like **Employee, Customer**etc.   
  
**Example 2 :**Notice that in the output the duplicate employee objects are not removed. This is because, the default comparer is being used which will **just check for object references being equal**and not the individual property values.

List<Employee> list1 = new List<Employee>()

{

    new Employee { ID = 101, Name = "Mike"},

    new Employee { ID = 102, Name = "Susy"},

    new Employee { ID = 103, Name = "Mary"}

};

List<Employee> list2 = new List<Employee>()

{

    new Employee { ID = 101, Name = "Mike"},

    new Employee { ID = 104, Name = "John"}

};

var result = list1.Union(list2);

foreach (var v in result)

{

    Console.WriteLine(v.ID + "\t" + v.Name);

}

**Output :**   
   
  
**Example 3 :**To solve the problem in **Example 2**, there are 3 ways  
**1.** Use the other overloaded version of **Union()**method to which we can pass a custom class that implements **IEqualityComparer**  
**2.**Override **Equals()**and **GetHashCode()**methods in **Employee**class  
**3.** Project the properties into a new anonymous type, which overrides **Equals()**and **GetHashCode()**methods  
  
Project the properties into a new anonymous type, which overrides **Equals()**and **GetHashCode()**methods

List<Employee> list1 = new List<Employee>()

{

    new Employee { ID = 101, Name = "Mike"},

    new Employee { ID = 102, Name = "Susy"},

    new Employee { ID = 103, Name = "Mary"}

};

List<Employee> list2 = new List<Employee>()

{

    new Employee { ID = 101, Name = "Mike"},

    new Employee { ID = 104, Name = "John"}

};

var result = list1.Select(x => new { x.ID, x.Name })

                    .Union(list2.Select(x => new { x.ID, x.Name }));

foreach (var v in result)

{

    Console.WriteLine(v.ID + "\t" + v.Name);

}

**Output :**   
union in linq c#   
  
**Intersect()**returns the common elements between the 2 collections.  
  
**Example 4 :**Return common elements in numbers1 and numbers2 collections.

int[] numbers1 = { 1, 2, 3, 4, 5 };

int[] numbers2 = { 1, 3, 6, 7, 8 };

var result = numbers1.Intersect(numbers2);

foreach (var v in result)

{

    Console.WriteLine(v);

}

**Output :**   
intersect in linq   
  
**Except()** returns the elements that are present in the first collection but not in the second collection.  
  
**Example 5:**Return the elements that are present in the first collection but not in the second collection.

int[] numbers1 = { 1, 2, 3, 4, 5 };

int[] numbers2 = { 1, 3, 6, 7, 8 };

var result = numbers1.Except(numbers2);

foreach (var v in result)

{

    Console.WriteLine(v);

}

**Output :**   
except in linq