**The following operators belong to Set operators category**  
Distinct  
Union  
Intersect  
Except   
  
   
  
In this video we will discuss **Distinct**operator. This operator returns distinct elements from a given collection.  
  
**Example 1:**Return **distinct**country names. In this example the default comparer is being used and the comparison is case-sensitive, so in the output we see country USA 2 times. 

string[] countries = { "USA", "usa", "INDIA", "UK", "UK" };

var result = countries.Distinct();

foreach (var v in result)

{

    Console.WriteLine(v);

}

**Output:**   
distinct in linq   
  
**Example 2:** For the **comparison to be case-insensitive**, use the other overloaded version of **Distinct()**method to which we can pass a class that implements **IEqualityComparer**as an argument. In this case we see country USA only once in the output.

string[] countries = { "USA", "usa", "INDIA", "UK", "UK" };

var result = countries.Distinct(StringComparer.OrdinalIgnoreCase);

foreach (var v in result)

{

    Console.WriteLine(v);

}

**Output:**   
distinct in linq example   
  
When comparing elements, **Distinct**() works in a slightly different manner with **complex types**like Employee, Customer etc.   
  
**Example 3:**Notice that in the output we don't get unique employees. 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> list = new List<Employee>()

{

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

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

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

};

var result = list.Distinct();

foreach (var v in result)

{

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

}

**Output:**   
linq distinct c# example   
  
**To solve the problem in Example 3, there are 3 ways**  
**1.** Use the other overloaded version of **Distinct()** 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  
  
**Example 4 :**Using the overloaded version of **Distinct()**method to which we can pass a custom class that implements **IEqualityComparer**  
  
**Step 1 :**Create a custom class that implements **IEqualityComparer<T>** and implement **Equals()**and **GetHashCode()**methods

public class EmployeeComparer : IEqualityComparer<Employee>

{

    public bool Equals(Employee x, Employee y)

    {

        return x.ID == y.ID && x.Name == y.Name;

    }

    public int GetHashCode(Employee obj)

    {

        return obj.ID.GetHashCode() ^ obj.Name.GetHashCode();

    }

}

**Step 2 :** Pass an instance of **EmployeeComparer**as an argument to **Distinct()**method

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

{

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

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

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

};

var result = list.Distinct(new EmployeeComparer());

foreach (var v in result)

{

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

}

**Output:**   
iequalitycomparer example   
  
**Example 5 :**Override **Equals()**and **GetHashCode()**methods in **Employee**class

public class Employee

{

    public int ID { get; set; }

    public string Name { get; set; }

    public override bool Equals(object obj)

    {

        return this.ID == ((Employee)obj).ID && this.Name == ((Employee)obj).Name;

    }

    public override int GetHashCode()

    {

        return this.ID.GetHashCode() ^ this.Name.GetHashCode();

    }

}

**Example 6 :** Project the properties into a **new anonymous type**, which overrides **Equals()**and **GetHashCode()**methods

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

{

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

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

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

};

var result = list.Select(x => new { x.ID, x.Name }).Distinct();

foreach (var v in result)

{

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

}