

# LINQ

## Sorguları

# Ele Alınacak Başlıklar

- Dizi Şeklindeki İşlemler (\*Sequences)
- Ertelenmiş Yürütme (\*Deferred Execution)
- Seçme, Filtreleme, Gruplama, Join İşlemleri  
(\*Selection, Filtering, Grouping and Joining)

Bir LINQ  
Sorgusu Nasıldır?

1 "Sequence" olarak isimlendirilir.

2 "Range" olarak isimlendirilir.

3 Sorgu operatörü

Bu tarz sorgu "comprehension query syntax" olarak adlandırılır.

```
var employees = new EmployeeRepository().GetAll();  
int maxNameLength = 5;
```

```
var employeesWithShortNames =  
    from employee in employees  
    where employee.Name.Length <= maxNameLength  
    select employee;
```

# Comprehension Query & Lambda Query

- Lambda sorguları daha fazla kontrol ve esneklik sağlamaktadır.
- Lambda sorgularında, operatörlerin birbirlerine bağlanması "pipeline" şeklinde gerçekleşir.
- Lambda sorgularında ("Projection" yapılmadığı sürece) "Select" operatörünün kullanımı seçimlidir.
- Bir çok Lambda operatörünün "comprehension query" yazarken karşılığı bulunmamaktadır.

```
var employeesWithShortNames =  
    employees.Where(e => e.Name.Length <= maxNameLength)  
        .Select(e => e);
```

---

```
var sortedEmployees =  
    from employee in employees  
    where employee.Name.Length <= 4  
    orderby employee.Name ascending  
    select employee;
```

# "let" Anahtar Sözcüğü

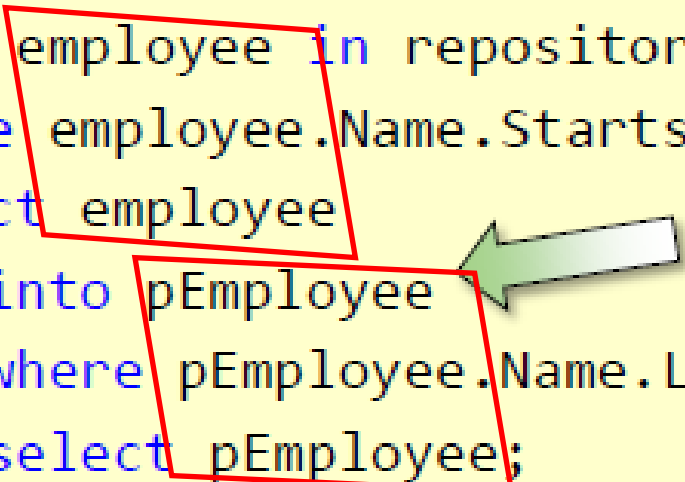
- Var olan "range" ı, yeni "range" değişkenine dönüştürür. (projection)

```
var employees =  
    from employee in repository.GetAll()  
        let lowercaseName = employee.Name.ToLower()  
    where lowercaseName.StartsWith(lowercaseName.Substring(  
                                                lowercaseName.Length - 1))  
    select employee;
```

# "into" Anahtar Sözcüğü

- "into" anahtar sözcüğü projeksiyon işleminden sonra sorguya devam eder. Genellikle gruplamak amaçlı olarak bu anahtar sözcük kullanılır.
- Önceki range değişkeni etki alanının dışına çıkar.

```
var employees =  
    from employee in repository.GetAll()  
    where employee.Name.StartsWith("P")  
    select employee  
    into pEmployee  
    where pEmployee.Name.Length < 5  
    select pEmployee;
```

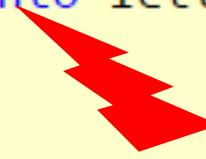




# Gruplama (Grouping)

Bu operator sorguyu sonlandırır.

```
var groupedEmployees =  
    from employee in repository.GetAll()  
    group employee by employee.Name[0] into letterGroup  
    orderby letterGroup.Key ascending  
    select letterGroup;
```



Sorguya devam edilmek isteniyorsa "*into*" operatörünün kullanılması gerekir.

```
foreach (var group in groupedEmployees)  
{  
    Console.WriteLine(group.Key);  
    foreach (var employee in group)  
    {  
        Console.WriteLine("\t{0}", employee.Name);  
    }  
}
```

Gruplar, anahtar-değer çiftleri şeklinde tutulurlar.

Daha karmaşık kriterler kullanarak da gruplandırma yapmak mümkündür.

```
var groupedEmployees =  
    from employee in repository.GetAll()  
    group employee by new { employee.DepartmentID,  
                            FirstLetter = employee.Name[0] };
```

```
foreach(var group in groupedEmployees)  
{  
    Console.WriteLine("\t{0} - {1}",  
        group.Key.DepartmentID,  
        group.Key.FirstLetter);  
    foreach (var employee in group)  
    {  
        Console.WriteLine(employee.Name);  
    }  
}
```

# Grouping ve Projecting

```
var groupedEmployees =  
    from employee in repository.GetAll()  
    group employee  
        by new { employee.DepartmentID,  
                 FirstLetter = employee.Name[0] }  
    into gEmployee  
    where gEmployee.Count() > 1  
    select new {  
        DepartmentID = gEmployee.Key.DepartmentID,  
        FirstLetter = gEmployee.Key.FirstLetter,  
        Count = gEmployee.Count()  
    };
```

Grupping ve Projecting işlemlerini Lambda ifadelerini kullanarak da yapmak mümkündür.

```
var groupedEmployees =  
    repository.GetAll()  
        .GroupBy(e => new { e.DepartmentID,  
                             FirstLetter = e.Name[0]})  
        .Where(g => g.Count() > 1)  
        .Select(g => new {  
            g.Key.DepartmentID,  
            g.Key.FirstLetter,  
            Count = g.Count()  
        });
```

## İç-içe Geçmiş Sorgular (Nested Queries)

```
var engineeringEmployees =  
    from employee in employeeRepository.GetAll()  
    where employee.DepartmentID ==  
        (from department in departmentRepository.GetAll()  
         where department.Name == "Engineering"  
         select department).First().ID  
    select employee;
```

```
var employees =  
    from employee in employeeRepository.GetAll()  
    select new  
    {  
        Name = employee.Name,  
        Department = (from department in departmentRepository.GetAll()  
                       where department.ID == employee.DepartmentID  
                       select department).First().Name  
    };
```

Bu örnek için Join işlemi yapmak daha anlamlı olacaktır.

# Joining

- İki veri topluluğunu (inner ve outer sequences) "equals" anahtar sözcüğünü kullanarak birbirine bağlar.
- (==) kullanılmaz.
- SQL sorgusundaki INNER Join işlemine denktir.

```
var employees =  
    from employee in employeeRepository.GetAll()  
    join department in departmentRepository.GetAll()  
    on employee.DepartmentID equals department.ID  
    select new { employee.Name, Department = department.Name };
```



# "orderby" Anahtar Sözcüğü (Sorting)

- "orderby" anahtar sözcüğünden sonra çoklu ifadeler kullanılabilir.
- Varsayılan sıralama artan (ascending) sıralamadır.

```
var employees =  
    from employee in employeeRepository.GetAll()  
    orderby employee.DepartmentID ascending,  
             employee.Name descending  
    select employee;
```

```
var employees = employeeRepository.GetAll()  
    .OrderBy(e => e.ID)  
    .ThenByDescending(e => e.Name);
```