

Oefening 2:

Method syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Where(c => c == 'E')
```

Query syntax:

```
from c in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS" where c == 'E' select c
```

Method Syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Where(c => c == 'E' || c == 'Z')
```

Query syntax:

```
from c in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS" where c == 'E' || c == 'Z' select c
```

Method syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Where(c => c != ' ').Where(c => c <= 'E')
```

Of

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Where(c => c != ' ' && c <= 'E')
```

Query Syntax:

```
from c in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS" where c != ' ' where c <= 'E' select c
```

Of

```
from c in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS" where c != ' ' && c <= 'E' select c
```

Method syntax:

```
new int[] {1,20,5,32,2,8,77,100}.Where (n => n%5 == 0)
```

Query syntax:

```
from n in new int[] {1,20,5,32,2,8,77,100} where n%5 == 0 select n
```

Method syntax:

- ⇒ De **where** methode heeft ook een **overload** waarbij 2 parameters worden doorgegeven, de 2^e is de index. Merk op dat vanaf dat er meer dan 1 parameter is, er met ronde haken dient te worden gewerkt rond de parameters.

```
new int[] {1,20,5,32,2,8,77,100}.Where ((n,i) => i%2 != 0)
```

Query syntax:

- ⇒ Hierbij is er geen standaard manier voorzien om deze query te verwezenlijken
-

Method syntax:

```
Cars.Where(c => c.Year < 1994)
```

Query syntax:

```
from c in Cars where c.Year < 1994 select c
```

Method syntax:

⇒ Of je kan (voor beide syntaxen) ook met een EN (&&) constructie werken uiteraard (zie oef. Hierboven)

```
Cars.Where(c => c.Manufacturer.Name == "Aston Martin").Where(c => c.Fuel == "Hybrid")
```

Query Syntax:

```
from c in Cars where c.Manufacturer.Name == "Aston Martin" where c.Fuel == "Hybrid" select c
```

Oefening 3:

Method syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Where(c => c != ' ').OrderBy(c => c)
```

Query syntax:

```
from c in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS" where c != ' ' orderby c select c
```

Method syntax:

- ⇒ Distinct zorgt ervoor dat uit het resultaat van de query alle dubbele resultaten worden verwijderd.

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Where(c => c != ' ').OrderBy(c => c).Distinct()
```

Query Syntax:

- ⇒ Het **distinct** keyword is niet beschikbaar via de query syntax, je zou een combinatie kunnen maken met de Method syntax (je kan immers op het query resultaat van de ene syntax opnieuw een query bouwen met de andere syntax...

```
(from c in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS" where c != ' ' orderby c select c).Distinct()
```

Method syntax:

```
new int[] {1,20,5,32,2,8,77,100}.Where (n => n%5 == 0).OrderByDescending(n => n)
```

Query syntax:

```
from n in new int[] {1,20,5,32,2,8,77,100} where n%5 == 0 orderby n descending select n
```

Method syntax:

- ⇒ We sorteren eerst op jaar en vervolgens op Model

```
Cars.Where(c => c.Fuel == "Diesel").OrderByDescending(c => c.Year).ThenBy(c => c.Model)
```

Query syntax:

- ⇒ Merk op dat de volgorde van de orderby omgekeerd is dan bij de Method syntax !

```
from c in Cars where c.Fuel == "Diesel" orderby c.Model orderby c.Year descending select c
```

Method syntax:

```
Cars.Where(c => c.Fuel == "Diesel").OrderBy(c => c.Manufacturer.Name)
```

Query syntax:

```
from c in Cars where c.Fuel == "Diesel" orderby c.Manufacturer.Name select c
```

Oefening 4:

Method syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS"
    .Split()
    .Where(w => w.Length >= 3)
    .OrderBy(w => w)
    .GroupBy(w => w.Length)
```

Query Syntax:

- ⇒ **Split** is geen LINQ method (maar een gewone methode van de c# string klasse), dus er is hiervoor geen overeenkomstige Query syntax. Belangrijk is om dit onderscheid te maken (wat hoort bij LINQ en wat niet), want dat betekent dus dat we de Split methode ook niet noodzakelijk zullen kunnen gebruiken bij andere lijst vormen (array, list, DBSet,..).

```
from w in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Split()
where w.Length >= 3
orderby w
group w by w.Length
```

Method syntax:

- ⇒ Groepen hebben een Key, dat is het element waarop de groep werd gemaakt (dus in dit geval de lengte van de woorden)

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS"
    .Split()
    .Where(w => w.Length >= 3)
    .OrderBy(w => w)
    .GroupBy(w => w.Length)
    .OrderBy(g => g.Key)~
```

Query Syntax:

- ⇒ Het is niet mogelijk om na een **group** nog een **orderby** te doen, maar je kan wel de query verpakken tussen ronde haakjes en op het resultaat opnieuw een query doen (die dan bijkomend de orderby doet op basis van de Key van de groepen)

```
from g in
(from w in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Split()
where w.Length >= 3
orderby w
group w by w.Length) orderby g.Key select g
```

Method Syntax:

```
new int[] {1,20,5,32,2,8,77,100}
    .OrderBy(n => n)
    .GroupBy(n => n/10)~
```

Query Syntax:

```
from n in new int[] {1,20,5,32,2,8,77,100} orderby n group n by n/10
```

Method Syntax:

```
new int[] {1,20,5,32,2,8,77,100}
    .OrderBy(n => n)
    .GroupBy(n => n%10)
    .OrderBy(g => g.Key)
```

Query Syntax:

```
from g in
  (from n in new int[] {1,20,5,32,2,8,77,100} orderby n group n by n%10)
orderby g.Key
select g
```

Method Syntax:

```
Cars.GroupBy(c => new {c.Fuel, c.Year})
```

Query Syntax:

```
from c in Cars group c by new {c.Fuel, c.Year}
```

Method Syntax:

```
Cars
    .OrderBy(c => c.Model)
    .GroupBy(c => new {c.Fuel, c.Year})
    .OrderByDescending(g => g.Key.Year).ThenBy(g => g.Key.Fuel)
```

Query Syntax:

```
from g in
  (from c in Cars orderby c.Model group c by new {c.Fuel, c.Year})
orderby g.Key.Fuel orderby g.Key.Year descending select g
```

Oefening 5:

Method syntax

Cars

```
.Where(c => c.Model.Length > 5)
.OrderBy(c => c.Model)
.Select (c => c.Model)
.Distinct()
```

Query Syntax:

```
(from c in Cars where c.Model.Length > 5 orderby c.Model select c.Model)
.Distinct()
```

Method syntax:

Cars

```
.Where(c => c.Fuel == "Hybrid" && c.Year == 2023)
.Select (c => c.Manufacturer.Name).Distinct()
.OrderByDescending(mn => mn)
```

Query Syntax:

```
from mn in ((from c in Cars where c.Fuel == "Hybrid" && c.Year == 2023
select c.Manufacturer.Name)
.Distinct()) orderby mn descending select mn
```

Method syntax:

Cars

```
.Where(c => c.Fuel == "Hybrid" && c.Year == 2023)
.Select (c => c.Manufacturer.Name).Distinct()
.GroupBy(mn => mn[0])
```

Query Syntax:

```
from mn in ((from c in Cars where c.Fuel == "Hybrid" && c.Year == 2023
select c.Manufacturer.Name)
.Distinct()) group mn by mn[0]
```

Method syntax:

Cars

```
.Where(c => c.Fuel == "Hybrid" && c.Year == 2023)
.Select (c => new {Manufacturer = c.Manufacturer.Name, c.Model}).Distinct()
.GroupBy(mn => mn.Manufacturer[0])
```

Query Syntax:

```
from mn in ((from c in Cars where c.Fuel == "Hybrid" && c.Year == 2023
select new {Manufacturer = c.Manufacturer.Name, c.Model})
.Distinct()) group mn by mn.Manufacturer[0]
```

Oefening 6:

Method Syntax:

⇒ Opvragen van page 1 (met `pageLength = 5`)

Cars

```
.OrderBy(c => c.Year).ThenBy(c => c.Model)
.Select(c => new {c.Year, c.Model, c.Fuel})
.Skip(0).Take(5)
```

Query Syntax:

⇒ Voor **Skip & Take** is er eveneens geen equivalent in de Query Syntax, daardoor gaan we hiervoor terug over naar de Method Syntax.

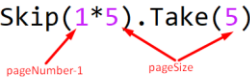
```
(from c in Cars orderby c.Model orderby c.Year select new {c.Year, c.Model, c.Fuel})
.Skip(0).Take(5)
```

Method Syntax:

⇒ Opvragen van page 2 (met `pageLength = 5`)

Cars

```
.OrderBy(c => c.Year).ThenBy(c => c.Model)
.Select(c => new {c.Year, c.Model, c.Fuel})
.Skip(1*5).Take(5)
```



The diagram shows two red arrows pointing from labels below to the expression '1*5' in the code. The label 'pageNumber-1' has an arrow pointing to the '1', and the label 'pageSize' has an arrow pointing to the '5'.

Query Syntax:

```
(from c in Cars orderby c.Model orderby c.Year
select new {c.Year, c.Model, c.Fuel})
.Skip(1*5).Take(5)
```

Oefening 7:

⇒ Voor deze oefeningen is er geen query syntax variante

Method Syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Count()
```

Het Count keyword bestaat niet in de Query Syntax.

Method Syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS"  
.Split().Where(w => w.Length >=2)  
.Count()
```

Query Syntax:

⇒ Zowel voor **Split** als **Count** moeten we terugvallen op de method Syntax

```
(from w in  
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Split()  
where w.Length >= 2 select w).Count()
```

Method Syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS"  
.Split()  
.Max(w => w.Length)
```

Query Syntax:

Ook hier is er geen alternatief voor de query syntax, aangezien zowel **Split** als **Max** niet mogelijk zijn.

Method Syntax:

```
"DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS"  
.Split()  
.GroupBy(w => w.Length)  
.Select(g => new {Length = g.Key, Count = g.Count()})  
.OrderBy(g => g.Length)
```

Query Syntax:

```
from g in (  
from w in "DIT IS EEN ZIN MET 8 WOORDEN EN 2 CIJFERS".Split()  
group w by w.Length)  
orderby g.Key  
select new {Length = g.Key, Count = g.Count()}
```

Method syntax:

```
new int[] {1,20,5,32,2,8,77,100}.Where(n => n>10).Sum()
```

Query Syntax:


```
(from n in new int[] {1,20,5,32,2,8,77,100} where n > 10 select n)
.Sum()
```

Method Syntax:

```
new int[] {1,20,5,32,2,8,77,100}
.GroupBy(n => n%2)
.Select(g => new {
    Even = g.Key == 1,
    Count = g.Count(),
    Max = g.Max(),
    Avg = g.Average()
})
```

Query Syntax:

⇒ Aangezien Count, Max, Average,.. niet beschikbaar zijn in de query syntax moeten we voor deze select ook overschakelen naar de Method syntax.

```
(from n in new int[] {1,20,5,32,2,8,77,100} group n by n%2)
.Select(g => new {
    Even = g.Key == 1,
    Count = g.Count(),
    Max = g.Max(),
    Avg = g.Average()
})
```

Oefening 8:

Query Syntax:

- ⇒ We kunnen een join voorzien om de relatie aan te geven.
- ⇒ ofwel speciëren we via de select welke properties we wensen op te vragen van beide tabellen

```
var query2 = from c in ctxt.Cars
             join m in ctxt.Manufacturers on c.ManufacturerId equals m.Id where c.Id == -1
             select new { Car = c, ManName = m.Name };

foreach (var c in query2)
{
    Console.WriteLine($"Id: {c.Car.Id}\t Year:{c.Car.Year}\tFuel:{c.Car.Fuel}" +
        $"\tManId:{c.Car.ManufacturerId}\tModel:{c.Car.Model}" +
        $"\tManId:{c.ManName}");
}
```

- ⇒ Kan echter ook op deze manier zonder de join expliciet te voorzien (EF weet immers dat er een join moet gebeuren door de navigation property van een Car naar de Manufacturer)

```
var query3 = from c in ctxt.Cars
             where c.Id == -1
             select new { Car = c, c.Manufacturer };

```

- ⇒ Ofwel gebruiken we **Include** (method syntax) zodat alle properties van beide objecten worden opgehaald en dan wordt eveneens de join overbodig en hoeven we geen aparte properties te speciëren via de select (zowel de Car als bijhorende Manufacturer object worden opgehaald)

```
var query2 = from c in ctxt.Cars.Include(c => c.Manufacturer)
             where c.Id == -1
             select c;

foreach (var c in query2)
{
    Console.WriteLine($"Id: {c.Id}\t Year:{c.Year}\tFuel:{c.Fuel}" +
        $"\tManId:{c.ManufacturerId}\tModel:{c.Model}" +
        $"\tManId:{c.Manufacturer.Name}");
}
```

⇒

Method syntax:

- ⇒ Ook hier kunnen we de Include weglaten indien we in een Select zouden aangeven welke properties of objecten we wensen op te halen.

```
var ctxt = new LabContext();

var query = ctxt.Cars.Include(c => c.Manufacturer)
               .Where(c => c.Id == -1);

foreach (var c in query)
{
    Console.WriteLine($"Id: {c.Id}\t Year:{c.Year}\tFuel:{c.Fuel}" +
        $"\tManId:{c.ManufacturerId}\tModel:{c.Model}" +
        $"\tManId:{c.Manufacturer.Name}");
}
```

Query Syntax:

- ⇒ pagina 3 ophalen, dus we skippen de eerste 2 pagina's (2 x pagina lengte)
- ⇒ Merk op dat de join hier in principe dus mag worden weggelaten

```
var query2 = (from c in ctxt.Cars
              join m in ctxt.Manufacturers on c.ManufacturerId equals m.Id
              orderby c.Manufacturer.Name, c.Year, c.Model
              select new { c.Year, ManuName = c.Manufacturer.Name, c.Model })
              .Skip(2*10).Take(10);

foreach (var c in query2)
{
    Console.WriteLine($"Year:{c.Year}\tManufacturer:{c.ManuName}\tModel:{c.Model}");
}
```

Method syntax:

- ⇒ Aangezien we hier met een specifieke select clause werken, hoeven we de include dus niet te doen.

```
var query = ctxt.Cars
              .OrderBy(c => c.Manufacturer.Name).ThenBy(c => c.Year).ThenBy(c => c.Model)
              .Skip(2 * 10).Take(10)
              .Select(c => new { c.Year, ManuName = c.Manufacturer.Name, c.Model });

foreach (var c in query)
{
    Console.WriteLine($"Year:{c.Year}\tManufacturer:{c.ManuName}\tModel:{c.Model}");
}
```

Query syntax:

```
var query = (from m in ctxt.Manufacturers select new { m.Name, m.Cars.Count }).Distinct();

foreach (var m in query)
{
    Console.WriteLine($"Manufacturer:{m.Name}\tCars:{m.Count}");
}
```

Method syntax:

```
var query = ctxt.Manufacturers.Select(m => new { m.Name, m.Cars.Count }).Distinct();

foreach (var m in query)
{
    Console.WriteLine($"Manufacturer:{m.Name}\tCars:{m.Count}");
}
```

Method syntax:

```
Manufacturers.OrderBy(m => m.Name).GroupBy(c => c.Name)
.Select(c => new { c.Key, Cars = c.Sum(c => c.ManufacturerCars.Count()) })
```

Query Syntax:

```
from m in (from m in Manufacturers orderby m.Name group m by m.Name)
select new { m.Key, Cars = m.Sum(c => c.ManufacturerCars.Count()) }
```