C# Advanced
LINQ

Koen Bloemen

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

LINQ

LINQ

- LINQ = **L**anguage **IN**tegrated **Q**uery
- Ingebouwd in C#
- Compiler en IntelliSense detecteren fouten ⇒ compile-time
- Zelfde syntax voor alle gegevensbronnen
  - Geen specifieke taal nodig zoals T-SQL, XPath, XQuery, for-lussen, if's,...
- Queries rechtstreeks in C# uitvoeren op collections (List, array,...) en databases
- Declaratief (lijkt op SQL)
- Namespace: `using System.Linq;`
- LINQ queries lijken op SQL queries, maar volgorde van clauses is anders!
- 2 soorten LINQ
  - Query syntax (SQL notatie)
  - Method syntax (gebruikt lamba expressies)

Voorbeeld 1: LINQ query via “query syntax”

LINQ queries bestaan uit 3 delen

```
// 1) Definieer gegevensbron. Waaruit data selecteren?
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// 2) Query opstellen met g als iteratie variabele.
// Query zelf wordt opgeslagen in query variabele, maar nog niet uitgevoerd.
var evenQuery1 =
  from g in getallen // Kijk voor elk getal g in de array
  where g % 2 == 0 // of getal g even is (delen door 2 geeft rest 0)
  select g; // Selecteer getallen die aan voorwaarde voldoen

// 3) Query uitvoeren. Pas wanneer we elementen ervan nodig hebben.
// Dit noemen we ook wel "lazy evaluation".
foreach (int getal in evenQuery1)
{
  Console.Write($"{getal} "); // Afdruk: 0 2 4 6 8
}
```

Voorbeeld 1: LINQ query via “method syntax”

De vorige LINQ query, maar nu korter via method syntax en lambda expressie

```
// 1) Definieer gegevensbron. Waaruit data selecteren?
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// 2) Query opstellen met g als iteratie variabele.
// g moet voldoen aan voorwaarde: delen door 2 geeft als rest 0
// Dus g moet een even getal zijn.
// Query zelf wordt opgeslagen in query variabele, maar nog niet uitgevoerd.
var evenQuery1 = getallen.Where(g => (g % 2 == 0));

// 3) Query uitvoeren. Pas wanneer we elementen ervan nodig hebben.
// Dit noemen we ook wel "lazy evaluation".
foreach (int getal in evenQuery1)
{
  Console.Write($"{getal} "); // Afdruk: 0 2 4 6 8
}
```

Voorbeeld 2: getallen > 3 (query syntax)

LINQ queries bestaan uit 3 delen

```
// 1) Definieer gegevensbron. Waaruit data selecteren?
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// 2) Query opstellen met g als iteratie variabele.
// Query zelf wordt opgeslagen in query variabele, maar nog niet uitgevoerd.
var getallenGroterDan3 =
  from g in getallen   // Kijk voor elk getal g in de array
  where g > 3          // of getal g > 3
  select g;            // Selecteer getallen die aan voorwaarde voldoen

// 3) Query uitvoeren. Pas wanneer we elementen ervan nodig hebben.
// Dit noemen we ook wel "lazy evaluation".
foreach (int getal in getallenGroterDan3)
{
  Console.Write($"{getal} "); // Afdruk: 4 5 6 7 8 9
}
```

Voorbeeld 2: getallen > 3 (method syntax)

De vorige LINQ query, maar nu korter via method syntax en lambda expressie

```
// 1) Definieer gegevensbron. Waaruit data selecteren?
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// 2) Query opstellen met g als iteratie variabele.
// g moet voldoen aan voorwaarde: g moet groter zijn dan 3
// Query zelf wordt opgeslagen in query variabele, maar nog niet uitgevoerd.
var getallenGroterDan3 = getallen.Where(g => g > 3);

// 3) Query uitvoeren. Pas wanneer we elementen ervan nodig hebben.
// Dit noemen we ook wel "lazy evaluation".
foreach (int getal in getallenGroterDan3)
{
  Console.Write($"{getal} "); // Afdruk: 4 5 6 7 8 9
}
```

Voorbeeld 3: lijst sorteren (query syntax)

LINQ queries bestaan uit 3 delen

```
// 1) Definieer gegevensbron. Waaruit data selecteren?
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// 2) Query opstellen met g als iteratie variabele.
// Query zelf wordt opgeslagen in query variabele, maar nog niet uitgevoerd.
var gesorteerdeGetallen =
  from g in getallen   // Kijk voor elk getal g in de array
  orderby g            // sorteren van getallen van klein naar groot
  select g;            // Selecteer getallen die aan voorwaarde voldoen

// 3) Query uitvoeren. Pas wanneer we elementen ervan nodig hebben.
// Dit noemen we ook wel "lazy evaluation".
foreach (int getal in gesorteerdeGetallen)
{
  Console.Write($"{getal} "); // Afdruk: 0 1 2 3 4 5 6 7 8 9
}
```

Voorbeeld 3: lijst sorteren (method syntax)

De vorige LINQ query, maar nu korter via method syntax en lambda expressie

```
// 1) Definieer gegevensbron. Waaruit data selecteren?
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// 2) Query opstellen met g als iteratie variabele.
// OrderBy sorteert getallen van klein naar groot
// Query zelf wordt opgeslagen in query variabele, maar nog niet uitgevoerd.
var gesorteerdeGetallen = getallen.OrderBy(g => g);

// 3) Query uitvoeren. Pas wanneer we elementen ervan nodig hebben.
// Dit noemen we ook wel "lazy evaluation".
foreach (int getal in gesorteerdeGetallen)
{
  Console.Write($"{getal} "); // Afdruk: 0 1 2 3 4 5 6 7 8 9
}
```

Query onmiddellijk uitvoeren

LINQ query onmiddellijk uitvoeren en opslaan in List<T> of array

```
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

List<int> evenGetallenLijst =
    (from g in getallen
    where g % 2 == 0
    select g).ToList();

int[] evenGetallenArray =
    (from g in getallen
    where g % 2 == 0
    select g).ToArray();

int[] evenGetallenArray2 = getallen.Where(g => (g % 2 == 0)).ToArray();
```

Aggregaatfuncties

- Aggregaatfuncties returnen maar 1 waarde
  - Count(), Min(), Max(), Average(), First()
- Voeren de LINQ query ook onmiddellijk uit

```
int[] scores = { 90, 71, 82, 93, 75, 82 };

int aantal =
    (from score in scores
    where score > 80
    select score).Count(); // aantal scores groter dan 80: 4 (maar 1 waarde)

int hoogste =
    (from score in scores
    select score).Max(); // hoogste score: 93 (maar 1 waarde)
```

Aggregaatfuncties

```
int[] scores = { 90, 71, 82, 93, 75, 82 };

int eersteHogerDan92 =
    (from score in scores
    where score > 92
    select score).First(); // eerste score hoger dan 92: 93 (maar 1 waarde)

double gemiddelde =
    Math.Round((from score in scores
    select score).Average(), 2); // gemiddelde score
```

WHERE clause in LINQ

- Gebruikt om data te filteren uit de datasource

```
int[] reeks = { 20, 40, 10, 30 };

// Alle getallen groter dan of gelijk aan 30.
int[] resultaat1 = (from r in reeks
    where r >= 30
    select r).ToArray();

// Alle getallen gelijk aan 10 of gelijk aan 40.
int[] resultaat2 = (from r in reeks
    where r == 10 || r == 40
    select r).ToArray();

// Alle getallen groter dan 10 en kleiner dan 40.
// Hier bijvoorbeeld via method syntax in plaats van query syntax.
int[] resultaat3 = reeks.Where(r => (r > 10 && r < 40)).ToArray();
```

Distinct() method in LINQ

- Gebruikt om dubbele data uit de datasource weg te filteren

```
int[] getallen = { 10, 20, 30, 20, 40, 30, 50 };

// Distinct: we houden enkel de unieke gegevens over, geen duplicaten
var resultaat =
    (from g in getallen
    where g > 10
    orderby g descending
    select g).Distinct();

// Afdruk.
foreach (int getal in resultaat)
{
    Console.WriteLine(getal); // Afdruk: 50 40 30 20
}
```

GROUPBY clause in LINQ

- Gebruikt om data uit de datasource te groeperen

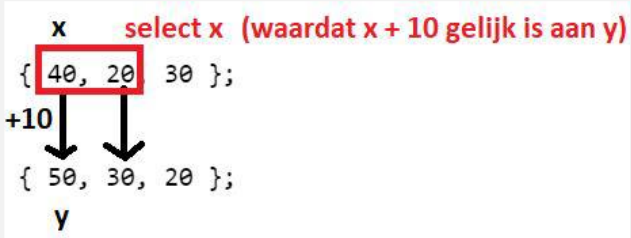
```
int[] getallen = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
// Groepeer de getallen in 2 groepen:
// even getallen:
// * hebben als Key true
// * 2 4 6 8
// oneven getallen:
// * hebben als Key false
// * 1 3 5 7 9

// method syntax
var resultaat = getallen.GroupBy(g => (g % 2 == 0));

// query syntax
var resultaat =
    from g in getallen
    group g by g % 2 == 0 into evenGroep
    select evenGroep;
```

GROUPBY clause in LINQ

```
// Voor elke groep in het resultaat
foreach (var group in resultaat)
{
    Console.WriteLine($"Is even? {group.Key}: "); // Key is true of false
    // Voor elk item in de groep
    foreach (var item in group)
    {
        Console.Write($"{item} ");
    }
    Console.WriteLine();
}

// Afdruk
Is even? False:
1 3 5 7 9
Is even? True:
2 4 6 8
```

JOIN clause in LINQ

```
int[] getallen1 = new int[3] { 40, 20, 30 }; // Array 1
int[] getallen2 = new int[3] { 50, 30, 20 }; // Array 2

// query syntax:
var resultaat =
    from x in getallen1
    join y in getallen2
    on (x + 10) equals y
    select x;

// method syntax:
var resultaat =
    getallen1.Join(getallen2,
        x => x + 10,
        y => y,
        (x, y) => x);
// Afdruk.
foreach (int getal in resultaat)
{
    Console.WriteLine(getal); // 40 20, want 40+10=50, 20+10=30, 30+10!=20
}
```



Classes en LINQ

```
// Student class maken met properties
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public int Age { get; set; }
}
// Student objecten toevoegen aan array
static void Main(string[] args)
{
    Student[] studenten = {
      new Student() { StudentID = 1, StudentName = "Joanna Pollers", Age = 28 },
      new Student() { StudentID = 2, StudentName = "Noah Janssens", Age = 31 },
      new Student() { StudentID = 3, StudentName = "Emma Pieters", Age = 35 },
      new Student() { StudentID = 4, StudentName = "Dries Gerris" , Age = 30 },
      new Student() { StudentID = 5, StudentName = "Tiziana Delaet" , Age = 41 },
      new Student() { StudentID = 6, StudentName = "Jonas Wellens", Age = 27 },
      new Student() { StudentID = 7, StudentName = "Joke Vermeersch",Age = 29 }};
}
```

Classes en LINQ

```
// LINQ queries op array van Student objecten

// Zoek de twintigers.
Student[] stud20 = studenten.Where(s => s.Age > 19 && s.Age < 30).ToArray();

// Zoek Dries.
// FirstOrDefault() returnt eerste element of standaard waarde
// als geen element gevonden.
// De standaardwaarde is afhankelijk van het datatype. Voor een object van een
// bepaalde class is dit null. Voor getallen bijvoorbeeld 0. En voor bool is
// dit false.
Student zoekNaamDries = studenten.Where(
    s => s.StudentName.Contains("Dries")).FirstOrDefault();

// Zoekt studentennummer 5
Student student5 = studenten.Where(s => s.StudentID == 5).FirstOrDefault();
```