

# .NET Advanced

LINQ vervolg



AP HOGESCHOOL  
ANTWERPEN

AP.BE

# Topics

- Query operaties
  - Filteren
  - Sorteren
  - Groeperen
- Projection (Custom)
- Paging
- IQueryable en IEnumerable
- Gerelateerde entiteiten laden



# LINQ vervolg

Query operaties

# LINQ vervolg - Query operaties

---

## Leerdoelen

---

- De meest courante query operaties in LINQ kennen
- De meest courante query operaties in LINQ kunnen toepassen

# LINQ vervolg - Query operaties - Filteren

## Theorie

- `where` clause bij query syntax of `Where` method bij extension method syntax
- Beperken van resultaat aan de hand van filtercriteria (analoog aan SQL)
- Toepassen van booleaanse condities (predicate) op elk source element
  - Bv. `naam.Length > 10`
- Standaard operatoren AND (&&) en OR (||) gebruiken om combinaties te maken
- Elke AND kan vervangen worden door een `where` clause of `Where` method

<code>where [condition1]</code>	<code>.Where([condition1])</code>
<code>where [condition2]</code>	<code>.Where([condition2])</code>

- Mag eender waar in de query staan met uitzondering van de eerste en laatste zin

# LINQ vervolg - Query operaties - Filteren

---

## Demo

---

- Demo 1
- Voorbeeldcode op digitap

# LINQ vervolg - Query operaties - Sorteren

## Theorie

- `orderby` clause bij query syntax of `OrderBy` method bij extension method syntax
- Analooq aan SQL
- Gebeurt met default comparer van het datatype waarop gesorteerd wordt
- Klein naar groot
  - Bij query syntax `ascending` keyword gebruiken
  - Bij extension method syntax veldnaam als lambda meegeven aan `OrderBy` method
  - Default indien sorteerrichting niet wordt gespecificeerd
- Groot naar klein
  - Bij query syntax `descending` keyword gebruiken
  - Bij extension method syntax `OrderByDescending` method gebruiken

# LINQ vervolg - Query operaties - Sorteren

## Voorbeelden

```
var simpsons = from simpson in GetSimpsons()  
                where simpson.FirstName.Length < 16  
                orderby simpson.FirstName  
                select simpson;
```

Character: Milhouse Van Houten  
Character: Nelson Muntz  
Character: Nick Riviera  
Character: Ralph Wiggum  
Character: Reverend Lovejoy

```
var simpsons = from simpson in GetSimpsons()  
                where simpson.FirstName.Length < 16  
                orderby simpson.FirstName descending  
                select simpson;
```

Character: Reverend Lovejoy  
Character: Ralph Wiggum  
Character: Nick Riviera  
Character: Nelson Muntz  
Character: Milhouse Van Houten

```
var simpsons = GetSimpsons()  
                .Where(s => s.FirstName.Length < 16)  
                .OrderByDescending(s => s.FirstName);
```



# LINQ vervolg - Query operaties - Sorteren

---

## Theorie

---

- Sorteren op meerdere velden afhankelijk van gekozen syntax method
- Query syntax
  - Optie 1: Combineren van velden telkens gescheiden door een ,
  - Optie 2: Herhalen van **orderby** clause per veld waarop gesorteerd moet worden
- Extension method syntax
  - **OrderBy** method voor het eerste veld, **ThenBy** method voor elk volgend veld
  - Bij aflopend sorteren vanaf 2<sup>de</sup> veld gebruiken we de **ThenByDescending** method

# LINQ vervolg - Query operaties - Sorteren

## Voorbeelden

### Query syntax

```
var simpsons = from simpson in GetSimpsons()  
                where simpson.FirstName.Length < 16  
                orderby simpson.FirstName.First(), simpson.LastName  
                select simpson;
```

```
var simpsons = from simpson in GetSimpsons()  
                where simpson.FirstName.Length < 16  
                orderby simpson.FirstName.First(), simpson.LastName descending  
                select simpson;
```

```
var simpsons = from simpson in GetSimpsons()  
                where simpson.FirstName.Length < 16  
                orderby simpson.FirstName.First()  
                orderby simpson.LastName descending  
                select simpson;
```

# LINQ vervolg - Query operaties - Sorteren

## Voorbeelden

Extension method syntax met lambda expressions

```
var simpsons = GetSimpsons()  
    .Where(s => s.FirstName.Length < 16)  
    .OrderBy(s => s.FirstName.First())  
    .ThenBy(s => s.LastName);
```

```
var simpsons = GetSimpsons()  
    .Where(s => s.FirstName.Length < 16)  
    .OrderBy(s => s.FirstName.First())  
    .ThenByDescending(s => s.LastName);
```

# LINQ vervolg - Query operaties - Groeperen

---

## Theorie

---

- `group` clause bij query syntax of `GroupBy` method bij extension method syntax
- Analooq aan SQL
- Samennemen van bij elkaar horende gegevens op basis van een key-waarde
- Geeft een collection terug van `IGrouping<TKey,TElement>` elementen, waarbij
  - `TKey` de gemeenschappelijke key is van de groep
  - `TElement` het element is dat hoort bij de key
- Geen `select` clause meer nodig bij query syntax
- Is laatste statement van een query
- Gebruik `var` voor betere leesbaarheid

# LINQ vervolg - Query operaties - Groeperen

## Voorbeelden

Query syntax

De waarde van de key van de groepering is een karakter dus DataType wordt char

```
IEnumerable<IGrouping<char, SimpsonCharacter>> simpsons = from simpson in GetSimpsons()  
group simpson by simpson.FirstName[0];
```

De brongegevens bevatten SimpsonCharacter objecten en zijn bijgevolg de elementen binnen de groep

We groeperen op het eerste karakter van de voor naam van elk karakter.



# LINQ vervolg - Query operaties - Groeperen


## Voorbeelden

Query syntax

```
IEnumerable<IGrouping<char, SimpsonCharacter>> simpsons = from simpson in GetSimpsons()  
                                                         group simpson by simpson.FirstName[0];
```

Of

```
var simpsons = from simpson in GetSimpsons()  
              group simpson by simpson.FirstName[0];
```



IGrouping	TKey (char)	TElement (SimpsonCharacter)
	N	Nelson Muntz
		Nick Riviera
	M	Moe Sizlack
		Milhouse Van Houten
	...	

# LINQ vervolg - Query operaties - Groeperen

## Voorbeelden

Extension method syntax met lambda expressions

```
IEnumerable<IGrouping<char, SimpsonCharacter>> simpsons = GetSimpsons()  
    .GroupBy(s => s.FirstName[0]);
```

```
var simpsons = GetSimpsons()  
    .GroupBy(s => s.FirstName[0]);
```

# LINQ vervolg - Query operaties - Groeperen

## Theorie

- Itereren door geneste **foreach** toe te passen

Buitenste iteratie om door de key values (outer list) te itereren

```
foreach ([outer list])  
{  
    foreach ([inner list])  
    {  
    }  
}
```

IGrouping	TKey (outer list)	TElement (inner list)
	N	Nelson Muntz
		Nick Riviera
	M	Moe Sizlack
		Milhouse Van Houten
	...	

Binnenste iteratie om door de elementen per key value (inner list) te itereren

# LINQ vervolg - Query operaties - Groeperen

---

## Demo

---

- Demo 2
- Voorbeeldcode op digitap

# LINQ vervolg - Query operaties - Groeperen

---

## Theorie

---

- Groeperen op basis van voorwaarde mogelijk (Equivalent van HAVING in SQL)
- Query syntax
  - **into** clause gebruiken
  - Verder zetten van query verplicht te doen met een nieuwe range variabele
  - **where** clause gebruiken m.b.v. nieuwe range variabele om te filteren op de groep
  - Eindigen met **select** statement of een **group** clause
- Extension method syntax
  - **Where** method gebruiken



# LINQ vervolg - Query operaties - Groeperen

## Voorbeelden

Query syntax

Definiëren van een nieuwe range variabele waarmee de query verder wordt gezet

```
var simpsons = from simpson in GetSimpsons()  
                orderby simpson.FirstName, simpson.LastName  
                group simpson by simpson.FirstName[0] into gSimpson  
                where gSimpson.Count() > 1  
                select gSimpson;
```

Filtering op groep gebeurt niet met having maar met **where** clause

# LINQ vervolg - Query operaties - Groeperen

## Voorbeelden

Extension method syntax met lambda expressions

```
var simpsons = GetSimpsons()  
    .OrderBy(s => s.FirstName)  
    .ThenBy(s => s.LastName)  
    .GroupBy(s => s.FirstName[0])  
    .Where(g => g.Count() > 1);
```

Filtering op groep gebeurt niet met having maar met **Where** method

# LINQ vervolg - Query operaties - Groeperen

---

## Theorie

---

- Groeperen op meerdere velden
- Query syntax
  - Nieuw anonymous object aanmaken in **group** clause
  - Daarin gewenste groeperingsvelden opnemen
- Extension method syntax
  - Via lambda in **GroupBy** method anonymous object aanmaken
  - Daarin gewenste groeperingsvelden opnemen

# LINQ vervolg - Query operaties - Groeperen

## Voorbeelden

### Query syntax

```
List<HockeyPlayer> players = new List<HockeyPlayer>
{
    new HockeyPlayer{Name="JSON Statham", Position="Defence", Team="Team good guys"},
    new HockeyPlayer{Name="Bruised Lee", Position="Forward", Team="Team good guys"},
    new HockeyPlayer{Name="John Woosh", Position="Defence", Team="Team bad guys"},
    new HockeyPlayer{Name="Get Set Lee", Position="Forward", Team="Team good guys"},
    new HockeyPlayer{Name="Puck Norris", Position="Defence", Team="Team bad guys"},
    new HockeyPlayer{Name="Random Lee", Position="Forward", Team="Team bad guys"},
    new HockeyPlayer{Name="Steven Smeagal", Position="Forward", Team="Team good guys"},
    new HockeyPlayer{Name="Snackie Chan", Position="Defence", Team="Team bad guys"},
};

var hockeyPlayers = from player in players
                    group player by new { player.Team, player.Position };
```

Groeperen op nieuw anonymous object waarin groeperingsvelden zijn opgenomen

# LINQ vervolg - Query operaties - Groeperen


## Voorbeelden

### Query syntax

```
foreach (var group in hockeyPlayers)
{
    Console.WriteLine($"{group.Key.Team} - {group.Key.Position}");
    Console.WriteLine("=====");

    foreach (var player in group)
    {
        Console.WriteLine($"- {player.Name}");
    }

    Console.WriteLine("");
}
```



```
Team good guys - Defence
=====
- JSON Statham

Team good guys - Forward
=====
- Bruised Lee
- Get Set Lee
- Steven Smeagal

Team bad guys - Defence
=====
- John Woosh
- Puck Norris
- Snackie Chan

Team bad guys - Forward
=====
- Random Lee
```



# LINQ vervolg - Query operaties - Groeperen

## Voorbeelden


Extension method syntax met lambda expressions

```
var hockeyPlayers = players
    .GroupBy(p => new { p.Team, p.Position });

foreach (var group in hockeyPlayers)
{
    Console.WriteLine($"{group.Key.Team} - {group.Key.Position}");
    Console.WriteLine("=====");

    foreach (var player in group)
    {
        Console.WriteLine($"- {player.Name}");
    }

    Console.WriteLine("");
}
```



```
Team good guys - Defence
=====
- JSON Statham

Team good guys - Forward
=====
- Bruised Lee
- Get Set Lee
- Steven Smeagal

Team bad guys - Defence
=====
- John Woosh
- Puck Norris
- Snackie Chan

Team bad guys - Forward
=====
- Random Lee
```

# LINQ vervolg

Projection

# LINQ vervolg - Projection

---

## Leerdoelen

---

- Het concept van projection kennen
- Projection kunnen toepassen

# LINQ vervolg - Projection

## Theorie

- Resultaat teruggeven waarbij het type van het object verschilt van bronoject type
- Via **select** clause bij query syntax of **Select** method bij extension method syntax
- Kan geheel nieuw type zijn of deel van bronoject type
- Kan met transformaties gebeuren (bv. concatenatie van 2 velden in 1)
- Equivalent in SQL
  - `"SELECT * FROM T"`  
vs
  - `"SELECT column2, UCASE(column3) FROM T"`
- Belangrijk bij het werken met EF Core en DTO's

# LINQ vervolg - Projection

---

## Theorie

---

- Projection kan met
  - Het data source type
    - Zelfde type als data source gebruiken
    - Niet opgevraagde properties in de query zullen lege/default waardes bevatten
  - Een ander type
  - Een anonymous type



# LINQ vervolg - Projection

---

## Demo

---

- Demo 3
- Voorbeeldcode op digitap

# LINQ vervolg

## Paging

# LINQ vervolg - Paging

---

## Leerdoelen

---

- Het concept van paging kennen
- Het mechanisme van paging kennen
- Paging kunnen toepassen

# LINQ vervolg - Paging

---

## Theorie

---

- Alle gegevens ophalen in 1 keer kan lang duren en veel resources verbruiken
  - RAM geheugen
  - Bandbreedte
  - Rekenkracht om webpagina's op te bouwen
- Vermijden van overmatig resourcegebruik door paging toe te passen
  - Opdelen van grote dataset in delen = pages
  - Page start vanaf index 0
  - Vast aantal rijen per page

# LINQ vervolg - Paging

---

## Theorie

---

- Gebruik van operatoren Skip() en Take() in deze volgorde
- Skip()
  - Slaat elementen over tot op een bepaalde positie in de lijst
  - Bereken het aantal elementen met behulp van de pagina index
  - Pagina index start steeds vanaf 0
  - In geval pagina index > aantal pagina's geen resultaat meer
- Take()
  - Neemt elementen uit de lijst tot op een bepaalde positie
  - Dit aantal mag niet veranderen!
- Vergeet niet om je lijst te **sorteren** bij gebruik van paging!

# LINQ vervolg - Paging

## Voorbeelden

```
public static void DisplayPage()
{
    List<SimpsonCharacter> simpsons = GetSimpsons();

    var page = simpsons
        .Skip(10)
        .Take(10);

    foreach (SimpsonCharacter simpson in page)
    {
        Console.WriteLine($"Id: {simpson.Id}, Firstname: {simpson.FirstName}, Lastname: {simpson.LastName}");
    }
}
```

```
Id: 11, Firstname: Kent, Lastname: Brockman
Id: 12, Firstname: Martin, Lastname: Prince
Id: 13, Firstname: Marvin, Lastname: Monroe
Id: 14, Firstname: Milhouse, Lastname: Van Houten
Id: 15, Firstname: Moe, Lastname: Szyslak
Id: 16, Firstname: Mr., Lastname: Burns
Id: 17, Firstname: Ned, Lastname: Flanders
Id: 18, Firstname: Otto, Lastname: Mann
Id: 19, Firstname: Patty, Lastname: Bouvier
Id: 20, Firstname: Ralph, Lastname: Wiggum
```

- We slaan de eerste 10 elementen over uit de brongegevens m.b.v. Skip(10)
- We nemen vervolgens de eerstvolgende 10 elementen via m.b.v. Take(10)

# LINQ vervolg - Paging

---

## Aandachtspunten

---

- Wanneer de datasource een database is
  - Pas de operatoren toe op een object dat interface IQueryable implementeert
  - Logica wordt dan op de database uitgevoerd en niet op een in-memory collection!

# LINQ vervolg - Paging

---

## Demo

---

- Demo 4
- Voorbeeldcode op digitap



# LINQ vervolg

IQueryable en IEnumerable

# LINQ vervolg - IQueryable en IEnumerable

---

## Leerdoelen

---

- Het concept van de IQueryable interface kennen
- Het verschil tussen de IQueryable en IEnumerable interfaces kennen

# LINQ vervolg - IQueryable en IEnumerable

## Theorie

- DbSet implementeert de interface IQueryable
  - Een LINQ query die wordt uitgevoerd op een IQueryable
    - Wordt onderliggend vertaald naar een SQL query
    - Wordt rechtstreeks uitgevoerd op de database en niet in-memory
    - Heeft minder geheugen nodig omdat enkel opgehaald wordt wat nodig is
  - Een LINQ query die wordt uitgevoerd op een IEnumerable
    - Wordt uitgevoerd op een in-memory collection ongeacht de bron
    - Acties zoals filteren worden niet uitgevoerd op de database maar in-memory
    - Potentieel meer data in het geheugen geladen dan nodig als bron een database is
- ⇒ Bij database als bron voer je eerst zoveel mogelijk acties op IQueryable uit

# LINQ vervolg - IQueryable en IEnumerable

---

## Aandachtspunten

---

- Let op met IQueryable's terug te geven naar buiten toe
  - Laten toe om van buiten de data laag query's uit te voeren
  - Risico van complexe en resource intensieve query's op te bouwen (query op query op query op ...)



# LINQ vervolg

Gerelateerde entiteiten laden

# LINQ vervolg - Gerelateerde entiteiten laden

---

## Leerdoelen

---

- Het concept van elk loading pattern kennen
- De Include() en aanverwante methoden kunnen toepassen
- De Entry() methode kunnen toepassen

# LINQ vervolg - Gerelateerde entiteiten laden

---

## Theorie

---

- Kan volgens 3 patterns
  - Eager loading
  - Explicit loading
  - Lazy loading

# LINQ vervolg - Gerelateerde entiteiten laden

---

## Theorie

---

### Eager loading

- Gerelateerde data wordt ingeladen als een integraal deel van de initiële query
- Om gerelateerde data op te halen via Eager loading:
  - ⇒ Gebruik je de Include() method
  - ⇒ Gebruik de ThenInclude() method om verdere dependent entiteiten mee op te halen
- Resulteert in een query met een of meerdere joins



# LINQ vervolg - Gerelateerde entiteiten laden

## Voorbeelden

We halen de EventRegistrations op voor elk event m.b.v. de Include() method

```
var result = _context.EventDefinitions
    .Include(e => e.EventRegistrations);
```

Dit resulteert in onderstaande query

```
Executed DbCommand (58ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
SELECT [t].[Id], [t].[Planning], [t].[Venue], [t0].[Id], [t0].[EventDefinitionId], [t0].[IsPreRegistered], [t0].[PersonId], [t0].[RegisteredAt]
FROM [Events].[tblEventDefinitions] AS [t]
LEFT JOIN [Events].[tblEventRegistrations] AS [t0] ON [t].[Id] = [t0].[EventDefinitionId]
ORDER BY [t].[Id], [t0].[Id]
```

# LINQ vervolg - Gerelateerde entiteiten laden

## Voorbeelden

We willen bovenop de registratie gegevens ook de persoonsgegevens ophalen

```
var result = _context.EventDefinitions
    .Include(e => e.EventRegistrations)
    .ThenInclude(r => r.Person);
```

Dit resulteert in onderstaande query

```
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
SELECT [t].[Id], [t].[Planning], [t].[Venue], [t2].[Id], [t2].[EventDefinitionId], [t2].[IsPreRegistered], [t2].[PersonId], [t2].[RegisteredAt], [t2].[Id0], [t2].[EmailAddress], [t2].[F
FROM [Events].[tblEventDefinitions] AS [t]
LEFT JOIN (
    SELECT [t0].[Id], [t0].[EventDefinitionId], [t0].[IsPreRegistered], [t0].[PersonId], [t0].[RegisteredAt], [t1].[Id] AS [Id0], [t1].[EmailAddress], [t1].[FirstName], [t1].[LastName]
    FROM [Events].[tblEventRegistrations] AS [t0]
    INNER JOIN [Person].[tblPersons] AS [t1] ON [t0].[PersonId] = [t1].[Id]
) AS [t2] ON [t].[Id] = [t2].[EventDefinitionId]
ORDER BY [t].[Id], [t2].[Id], [t2].[Id0]
```

# LINQ vervolg - Gerelateerde entiteiten laden

## Theorie

### Explicit loading

- Expliciet inladen op een later tijdstip dan het tijdstip waarop de hoofdquery werd uitgevoerd
- Om gerelateerde data op te halen via Explicit loading
  - ⇒ Gebruik de `Entry()` method van de `DbContext` om een entiteit op te halen
  - ⇒ Roep op het resultaat de `Collection()` method aan wanneer je een lijst van gerelateerde entiteiten wil ophalen
  - ⇒ Of roep op het resultaat de `Reference()` method aan om een single entity op te halen
  - ⇒ Roep op het resultaat van de `Collection()/Reference()` method de `Load()` method aan
- Resulteert in x-aantal aparte query's die uitgevoerd worden

# LINQ vervolg - Gerelateerde entiteiten laden

## Voorbeelden

We halen de EventRegistrations op voor een event m.b.v. de Collection() en Load() methods

```
var eventDefinition = _context.EventDefinitions.Single(ed => ed.Id == 1);  
_context.Entry(eventDefinition)  
    .Collection(ed => ed.EventRegistrations)  
    .Load();
```

Dit resulteert in onderstaande aparte query's

```
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']  
SELECT TOP(2) [t].[Id], [t].[Planning], [t].[Venue]  
FROM [Events].[tblEventDefinitions] AS [t]  
WHERE [t].[Id] = 1
```

```
Executed DbCommand (1ms) [Parameters=[@__p_0='?' (DbType = Int32)], CommandType='Text', CommandTimeout='30']  
SELECT [t].[Id], [t].[EventDefinitionId], [t].[IsPreRegistered], [t].[PersonId], [t].[RegisteredAt]  
FROM [Events].[tblEventRegistrations] AS [t]  
WHERE [t].[EventDefinitionId] = @__p_0
```

# LINQ vervolg - Gerelateerde entiteiten laden

## Voorbeelden

We halen de persoonsgegevens van een registratie op m.b.v. de Reference() en Load() methods

```
var eventRegistration = _context.EventRegistrations.Single(er => er.Id == 1);  
  
_context.Entry(eventRegistration)  
    .Reference(er => er.Person)  
    .Load();
```

Dit resulteert in onderstaande aparte query's

```
Executed DbCommand (19ms) [Parameters=[], CommandType='Text', CommandTimeout='30']  
SELECT TOP(2) [t].[Id], [t].[EventDefinitionId], [t].[IsPreRegistered], [t].[PersonId], [t].[RegisteredAt]  
FROM [Events].[tblEventRegistrations] AS [t]  
WHERE [t].[Id] = 1
```

```
Executed DbCommand (12ms) [Parameters=[@__p_0='?' (DbType = Int32)], CommandType='Text', CommandTimeout='30']  
SELECT [t].[Id], [t].[EmailAddress], [t].[FirstName], [t].[LastName]  
FROM [Person].[tblPersons] AS [t]  
WHERE [t].[Id] = @__p_0
```

# LINQ vervolg - Gerelateerde entiteiten laden

---

## Theorie

---

### Lazy loading

- Automatisch inladen van gerelateerde data wanneer een navigation property aangesproken wordt
- Enkel bruikbaar mits installatie onderstaande NuGET package

`Microsoft.EntityFrameworkCore.Proxies`

- Instellen in `OnConfiguring()` method van de `DbContext`
  - `UseLazyLoadingProxies()` method aanroepen op `DbContextOptionsBuilder`

# Oefeningen

---

## Deel 5 - Oefenbundels 1 - 2

---

- Surf naar Digitap
- Download de opgaves
- Los de vragen op

# LINQ vervolg



## Bronnen

- Pluralsight
  - [LINQ Fundamentals](#) (Scott Allen)
- Microsoft
  - <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/return-or-skip-elements-in-a-sequence?redirectedfrom=MSDN>
  - <https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.skip?view=net-5.0>
  - <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/group-clause>
  - <https://docs.microsoft.com/en-us/dotnet/csharp/linq/group-query-results>
  - <https://docs.microsoft.com/en-us/dotnet/api/system.linq.igrouping-2?view=net-5.0>
  - <https://docs.microsoft.com/en-us/dotnet/csharp/linq/perform-inner-joins>
  - <https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.join?view=net-5.0>
  - <https://docs.microsoft.com/en-us/dotnet/api/system.linq.iqueryable?view=net-5.0>
  - <https://www.c-sharpcorner.com/article/iqueryable-vs-ienumerable/>
  - <https://entityframeworkcore.com/queries-data-loading-eager-lazy>