

Entity Framework Core

Inhoudstafel

1	Inleiding.....	3
1.1	Doelstelling.....	3
1.2	Nodige software.....	3
1.3	Wat is Entity Framework Core?	3
1.4	ORM (object-relational mapping)	4
2	SQL Server	4
2.1	Een database server	4
2.2	SQL Server Management Studio	5
2.3	Een database	6
3	Database-first: Scaffolding	8
4	Migraties	8
5	Code-First aanpak.....	9
5.1	Aanmaken van .Core Solution met Console App en Class Library	9
5.2	EF Core toevoegen	9
5.3	Het entity data model	11
5.4	De entities.....	11
5.4.1	Associaties.....	12
5.5	De DbContext-class	12
5.6	De connectionstring	13
	Een hard-coded connectionstring (niet safe in productie!)	14
	Een connectionstring uit appsettings.json halen	15
6	Migraties	16
6.1	Een eerste migration	16
6.2	Data annotations.....	17
6.2.1	De naam van de tabel instellen.....	17
6.2.2	De naam van een kolom instellen	17
6.2.3	Een kolom instellen als (niet) verplicht in te vullen	18
6.2.4	Het maximum aantal tekens in een varchar-kolom instellen	18
6.2.5	Het kolomtype instellen.....	18
6.2.6	De primary key instellen	19
6.2.7	De foreign key instellen	19
6.2.8	Een property niet opnemen in de databasetable	19

6.2.9	Oefening	20
6.3	Enums en partial classes.....	22
6.4	Fluent API	23
6.4.1	Property mapping.....	23
6.4.2	Type mapping.....	25
6.5	Oefening	25
6.6	Seeding.....	25
6.7	Oefening	30
6.8	LINQ-to-Entity query's.....	30
6.9	Query-methods.....	32
6.10	LINQ-query's vs. query methods.....	32
6.11	Een entity zoeken op basis van de PK-waarde	34
6.12	Gedeeltelijke objecten ophalen.....	34
6.13	Lazy en Eager loading	35
6.13.1	Eager Loading.....	35
	(1) Je gebruikt de method Include op de property Docenten. Als parameter geef je de naam van de associatie mee.	36
6.13.2	Lazy Loading.	37
	Daarnaast moeten we alle navigation properties virtual maken.	38
	• Controleer of in onderstaande classes de navigation properties virtual zijn.	38
7	Entity's toevoegen	40
7.1	Eén entiteit toevoegen.....	40
7.1.1	Meerdere entiteiten toevoegen.....	41
7.2	Meerdere entiteiten van een verschillend type toevoegen.....	42
7.3	Entiteiten én bijhorende nieuwe geassocieerde entiteiten toevoegen.....	42
7.4	Entiteiten én bijhorende bestaande geassocieerde entiteiten toevoegen.....	43
8	Entity's wijzigen	46
8.1	Eén entity wijzigen	46
8.2	Meerdere entity's lezen en slechts enkele daarvan wijzigen	47
8.3	Entity's wijzigen die indirect zijn ingelezen via associaties	47
8.4	Een associatie van een entity wijzigen	48
8.4.1	De associatie wijzigen vanuit de veel-kant	48
8.4.2	De associatie wijzigen vanuit de één-kant.....	49
9	Entity's verwijderen.....	50
9.1	Voorbeeld	50
10	Transacties.....	51

10.1	Kenmerken van transacties	51
10.2	Isolation level.....	52
10.3	De method SaveChanges.....	53
10.4	Eigen transactiebeheer met TransactionScope	53
10.5	Voorbeeld	55
10.5.1	Ingebouwd transactiebeheer	56
10.6	Eigen transactiebeheer	58
11	Optimistic record locking	60
11.1	Probleemstelling	60
11.2	Oplossing zonder timestampveld	61
11.3	Oplossing met timestampveld	63
12	De change tracker	65
12.1	Statussen van de ChangeTracker	65
12.2	Een voorbeeld.....	66
12.3	Het StateChanged en Tracked event.....	68

1 Inleiding

1.1 Doelstelling

In deze cursus leer je het .NET Entity Framework gebruiken.

Het entity framework, afgekort EF, brengt objecten in het interne geheugen in verband met records in een relationele database. Op die manier kunnen we op een vlotte manier gegevens bewaren, ophalen, bewerken en verwijderen in een database.

1.2 Nodige software

Bij de aanvang van de cursus veronderstellen we dat volgende software geïnstalleerd is :

- Visual Studio 2019 mét de recentste update of Visual Studio Code
- .NET Core 3.0 (minimaal)
- SQL Server
- Microsoft SQL Server Management Studio

1.3 Wat is Entity Framework Core?

Entity Framework is een **object-relational mapper (ORM)** van Microsoft die het mogelijk maakt om .NET ontwikkelaars te laten werken met een database, gebruik makend van .NET objecten die worden gemapt aan database-tabellen. Concreet zal de programmeur wijzigingen aanbrengen in één of meerdere objecten in het interne geheugen, EF zorgt ervoor dat deze wijzigingen worden doorgevoerd in de bijhorende database. Dit verhoogt de productiviteit van de .NET-ontwikkelaar. Oudere technologieën zoals ADO.NET vergden heel wat codeerwerk en waren veel arbeidsintensiever. Wie EF gebruikt werkt standaard met de Code-First techniek. Dit betekent dat de ontwikkelaar eerst het data model maakt en op basis daarvan de tabellenstructuur laat maken door het EF. Heb je reeds een database dan kan je het EF het data model laten maken. We noemen dit ook scaffold en we spreken dan van Database-First. EF kan overweg met tal van verschillende soorten databases. Wij zullen enkel werken met SQL Server databases.

Je vindt alle informatie over EF Core op <https://docs.microsoft.com/nl-nl/ef/core/>

1.4 ORM (object-relational mapping)

Je stelt dus gegevens in C# op een andere manier voor dan in een database. Dit betekent dat je ergens een vertaalslag zal moeten doen tussen deze verschillende visies. Deze vertaalslag zelf uitschrijven is niet gemakkelijk en vergt veel tijd. ORM is juist het converteren van de object-georiënteerde visie naar de database-visie. Een ORMlibrary helpt je deze conversie te doen.

EF Core is een ORM-library van Microsoft.

Een ORM-library biedt volgende voordelen :

Productiviteit Zelf de code schrijven die de vertaalslag doet tussen de twee verschillende visies vraagt veel code en tijd. Een ORM-library neemt je veel werk uit handen.

Onderhoudbaarheid Het databaseschema wijzigt in de tijd. Het class diagram eveneens. Deze visies continu op mekaar afstemmen gaat gemakkelijker mét een ORM-library dan zonder.

Databasemerk onafhankelijk Een ORM-library neemt de verschillen tussen databases voor zijn rekening

2 SQL Server

2.1 Een database server

Zoals gezegd kan EF Core overweg met verschillende soorten database servers. In deze cursus zullen we telkens gebruik maken van de in Visual Studio ingebouwde database server localDb. LocalDb is een database server, bedoeld om te gebruiken tijdens de ontwikkeling van jouw software project. Van zodra jouw project in productie gaat zal je eerder een Microsoft SQL Server of SQL Server Express gebruiken. Dit zijn ook database servers van Microsoft.

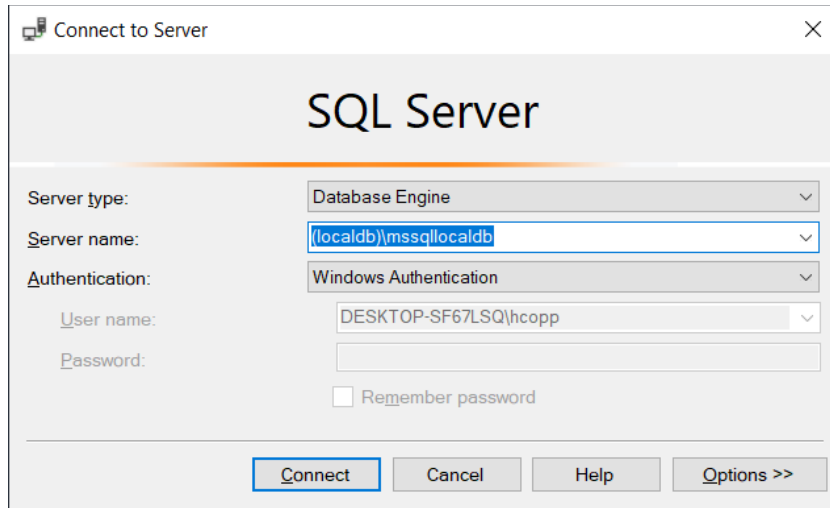
SQL Express is wel beperkter in zijn mogelijkheden :

- Een database is op SQL Express maximaal 4 GB groot
- SQL Express gebruikt maximaal 1GB RAM (ook als er meer beschikbaar is)
- SQL Express gebruikt maximaal 1 processor (ook als er meerdere beschikbaar zijn)

Er zijn twee manieren om jouw database te beheren. Ofwel gebruik je SQL Server Management Studio. Dit is een tool van Microsoft waarmee je niet alleen de inhoud van de tabellen in je database kan bekijken maar ook editen. Je kan de structuur van de tabellen veranderen, scripts uitvoeren,... De meeste van deze taken kan je ook vanuit Visual Studio uitvoeren. Je doet dit in het venster Server Explorer. Na het leggen van een Data Connection kan je ook van hieruit wijzigingen aanbrengen in de database.

2.2 SQL Server Management Studio

Zoals we eerder aangegeven hebben kunnen we de tool SQL Server Management Studio gebruiken om onze databases te beheren. We proberen nu één en ander uit met een database EFBieren. Deze database maken we aan met het databasescript EFcreateBieren.sql.



2.3 Een database

In deze cursus en in de bijhorende taken zullen we werken met enkele databases. Eén van die databases is de database **EFBieren**. Deze bevat volgende 3 tabellen :

- Brouwers Gegevens over de brouwers.
- Soorten Gegevens over de biersoorten, (alcoholvrij, alcoholarm, ...)
- Bieren Gegevens over de bieren zelf.

Eén van die databases is de database EFBieren. Deze bevat volgende 3 tabellen :

- **Brouwers** Gegevens over de brouwers.
- **Soorten** Gegevens over de biersoorten, (alcoholvrij, alcoholarm, ...)
- **Bieren** Gegevens over de bieren zelf.

De tabel Brouwers bevat volgende velden :

Veldnaam	Type	Lengte	Opmerkingen
BrouwerNr	Int		Primary Key én AutoNumber
BrNaam	Nvarchar	50	
Adres	Nvarchar	50	
PostCode	SmallInt		
Gemeente	Nvarchar	50	
Omzet	Int		Jaarlijkse omzet uitgedrukt in hectoliter

Enkele voorbeeldrecords van deze tabel :

BrouwerNr	BrNaam	Adres	PostCode	Gemeente	Omzet
1	Achouffe	Route du Village 32	6666	Achouffe-Wibrin	10000
2	Alken	Stationstraat 2	3570	Alken	950000
3	Ambly	Rue Principale 45	6953	Ambly-Nassogne	500
4	Anker	Guido Gezellelaan 49	2800	Mechelen	3000
6	Artois	Vaartstraat 94	3000	Leuven	4000000
8	Bavik	Rijksweg 33	8531	Bavikrove	110000
9	Belle Vue - Molenbeek	Henegouwenkaai 33	1080	Sint-Jans-Molenbeek	NULL
10	Belle Vue - Zuun	Steenweg naar Bergen	1600	Sint- Pieters-Leeuw	NULL
11	Belle Vue	Delaunoy-sstraat 58-60	1080	Sint-Jans-Molenbeek	300000
12	Bie (De)	Stoppelweg 26	8978	Watou	280
13	Binchoise	Faubourg St. Paul 38	7130	Binche	700

De tabel Soorten bevat volgende velden :

Veldnaam	Type	Lengte	Opmerkingen
SoortNr	Int		Primary Key én AutoNumber
Soort	Nvarchar	50	Geïndexeerd met toelating van dubbels

Enkele voorbeeldrecords van deze tabel :

SoortNr	Soort
2	Alcoholarm
3	Alcoholvrij
4	Ale
5	Alt
6	Amber
8	Bierette
11	Bitter

De tabel Bieren bevat volgende velden :

Veldnaam	Type	Lengte	Opmerkingen
BierNr	Int		Primary Key én AutoNumber
Naam	Nvarchar	100	Geïndexeerd met toelating van dubbels
BrouwerNr	Int		
SoortNr	Int		
Alcohol	Real		

Enkele voorbeeldrecords van deze tabel :

BierNr	Naam	BrouwerNr	SoortNr	Alcohol
4	A.C.O.	104	18	7
5	Aalbeeks St. Corneliusbier (=Kapittel pater (Het))	113	18	6,5
7	Aardbeien witbier	56	53	2,5
8	Aarschots kruikenbier (=St. Sebastiaan grand cru)	105	15	7,6
10	Abt Bijbier (Nen)	33	18	7
11	Adler	51	42	6,75
12	Aerts 1900	81	14	7
13	Affligem blond (Abdij)	100	33	7

Het relatiediagram van deze database :



Verderop in deze cursus zullen we deze database gebruiken om scaffolding te demonstreren.

3 Database-first: Scaffolding

In het vorige hoofdstuk bouwden we een database *EFBieren* op in Sql Server. Soms gebeurt het dat we reeds over een bestaande database beschikken. We kunnen dan op basis van deze database de entity classes laten aanmaken. Dit noemen we **scaffolding**.

We zullen deze database nu gebruiken in een nieuwe solution.

- Maak een nieuwe console app *UI* in een nieuwe solution *EFBierenScaffolding*.
- Voeg er ook een class library (.NET Core) aan toe met de naam *Model*.
- Voeg aan het project *Model* de nuget packages *Microsoft.EntityFrameworkCore*, *Microsoft.EntityFrameworkCore.SqlServer* en *Microsoft.EntityFrameworkCore.Design* toe. Kies telkens versie 3.1

We gaan nu de database *EFBieren* scaffolden.

- Build de solution.
- Start de package manager console en ga met de instructie `cd model` naar de map *Model*.
- Tik het commando

```
Scaffold-DbContext "Data Source=(LocalDB)\MSSQLLocalDB;Initial
Catalog=BierenMetUsersDb;Integrated Security=True"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Model
```

Meer informatie over het `ScaffoldDbContext` commando:

<https://docs.microsoft.com/en-us/ef/core/cli/powershell>

4 Migraties

Op dit moment kunnen we nog geen migrations toepassen op onze database.

Dit kan nochtans nodig zijn wanneer we in de toekomst bijvoorbeeld een extra tabel of extra veld in een tabel willen toevoegen.

- Tik in de Package Manager Console het commando

```
Add-Migration Initial
```

Het resultaat is een extra folder **Migrations** in het project *Model*

- Open in de folder *Migrations* de nieuwe migrationfile.

Deze start met een tijdstip en eindigt op `_initial.cs`. Zoals steeds bevat de migrationfile een method `Up()` en een method `Down()`.

Aangezien de database reeds bestaat mag alle code uit de method `Up()` verdwijnen.

- Wis alle code in de method `Up()`. We voeren nu de initiële migration uit. Op deze manier wordt in de database de tabel `__EFMigrationsHistory` aangemaakt.
- Tik in de Package Manager Console het commando

```
Update-database
```

De database is nu klaar om eventuele wijzigingen te ondergaan.

5 Code-First aanpak

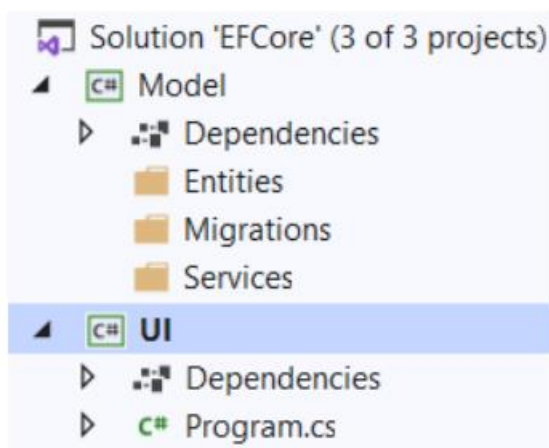
5.1 Aanmaken van .Core Solution met Console App en Class Library

We beginnen met het opbouwen van de solution.

- Maak in Visual Studio een nieuw project van het type Console App (.NET Core). Je noemt het project **UI** en de solution EFCore.

We voegen nog een tweede project toe, namelijk van het type Class Library (.NET Core).

- Voeg een .NET Core class library toe en noem deze Model.
- Verwijder de class Class1.cs. In het project Model komen de entities die gekoppeld zullen worden aan de tabellen in de database. We voegen nog enkele folders toe aan het project Model.
- Voeg in het project Model de folders Entities, Migrations en Services toe.



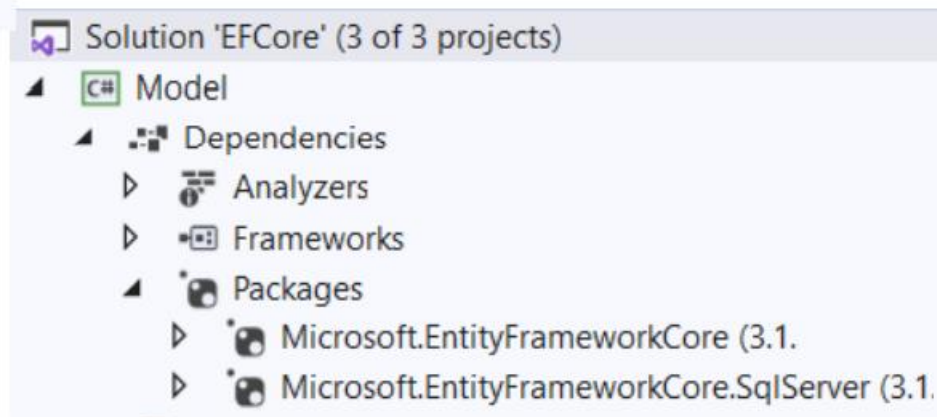
5.2 EF Core toevoegen

EF Core wordt standaard niet toegevoegd aan een project. We moeten dit zelf doen. In de volgende paragraaf zullen we in het project Model de entities toevoegen die we via het entity framework gaan koppelen met de databasetabellen.

Het is dus in dit project dat we EF Core zullen toevoegen.

- Kies in het menu voor Tools – NuGet Package Manager – Manage NuGet Packages for Solution...
- Klik links bovenaan op Browse...
- In het zoekveld tik je : microsoft.entityframeworkcore
- Zoek in de lijst de package Microsoft.EntityFrameworkCore en klik erop.
- Vink ernaast het project Model aan en kies naast Version: voor versie 3.1.11 klik op Install
- Vink in het venster Preview Changes de optie Do not show this again aan en klik op OK.
- Aanvaard de licentievoorwaarden met een klik op I Accept.

In de Solution Explorer kan je zien dan bij het project Model, onder Dependencies en Packages nu EFCore 3.1.11 is vermeld.



Op dezelfde manier voegen we nu ook de package Microsoft.EntityFrameworkCore.SqlServer toe. Deze package zorgt ervoor dat we EFCore kunnen toepassen op een SQL Server database. Een overzicht van andere EF Core database providers vind je op <https://docs.microsoft.com/enus/ef/core/providers>.

Tip Je kan een package ook nog op andere manieren installeren.

Met commando's in de package manager console

Je installeert de package Microsoft.EntityFrameworkCore in het project Model via het commando :
install-package microsoft.entityframeworkcore --version 3.1.11 -projectname Model

Via het .csproj-bestand

Je dubbelklikt in de Solution Explorer op de naam van het project Model. Een bestand met de naam Model.csproj wordt geopend. Voeg binnen een **ItemGroup** de nodige **packagereferences** toe.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="3.1.11" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="3.1.11">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.1.11">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="3.1.11">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
    </PackageReference>
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\Model\Model.csproj" />
  </ItemGroup>

</Project>
```

Via de command line interface (CLI)

Je start de Developer Command Prompt for VS2019. Verplaats je naar de folder waar jouw project opgeslagen is.

Tik het commando : dotnet add package Microsoft.EntityFrameworkCore --version 3.1.11

5.3 Het entity data model

Om te kunnen communiceren met de onderliggende database is het voor EF noodzakelijk om te beschikken over een entity data model. Dit entity data model (EDM) bestaat uit 3 delen :

het conceptueel model, het storage model en de mapping.

Het conceptueel model bevat de structuur van de entity classes. Een entity class is een class die een gegeven uit de werkelijkheid voorstelt.

De classes Docent en Campus zijn bijvoorbeeld entity classes.

De objecten die we maken op basis van deze classes noemen we entities.

Het conceptueel model bevat ook de associaties tussen de entity classes.

Het storage model bevat de structuur van iedere tabel en de relaties tussen deze tabellen.

De mapping bevat informatie over welke tabel bij welke entity class hoort. Het legt dus het verband tussen de entity classes en de database.

Conceptueel model, storage model en mapping worden door EF gegenereerd en geconfigureerd op basis van conventions. Je kan hier eventueel zelf wijzigingen of toevoegingen in aanbrengen.

5.4 De entities

We starten met een class Campus.

- Maak in de map **Entities** (in het project Model) een class Campus aan.
- Vul de class aan met onderstaande code :

```
public class Campus
{
    public int CampusId { get; set; }
    public string Naam { get; set; }
    public string Straat { get; set; }
    public string Huisnummer { get; set; }
    public string Postcode { get; set; }
    public string Gemeente { get; set; }
}
```

- Voeg in dezelfde folder ook een class Docent toe :

```
public class Docent
{
    public int DocentId { get; set; }
    public string Voornaam { get; set; }
    public string Familienaam { get; set; }
    public decimal Wedde { get; set; }
}
```

5.4.1 Associaties

Tussen de classes Campus en Docent bestaat een 1-op-veel relatie.

We drukken dit uit door in de classes Docent en Campus extra properties op te nemen:

```
public class Campus
{
    public int CampusId { get; set; }
    public string Naam { get; set; }
    public string Straat { get; set; }
    public string Huisnummer { get; set; }
    public string Postcode { get; set; }
    public string Gemeente { get; set; }
    public virtual ICollection<Docent> Docenten { get; set; } (1)
}

public class Docent
{
    public int DocentId { get; set; }
    public string Voornaam { get; set; }
    public string Familienaam { get; set; }
    public decimal Wedde { get; set; }
    public int CampusId { get; set; } (2)
    public virtual Campus Campus { get; set; } (3)
}
```

(1) In een campus werken meerdere docenten.

We drukken dit uit door aan de Campus-class een ICollection toe te voegen.

(2) Een docent is verbonden aan één campus. We nemen de id van de campus op in de Docentclass.

(3) Bijkomend voegen we aan de Docent-class een virtual property Campus toe. Deze property is van het type Campus. De property's Docenten in de class Campus en Campus in de class Docent noemen we navigation property's.

5.5 De DbContext-class

In het project Model gaan we nu een nieuwe class aanmaken die een voorstelling is van de database. Zo een voorstelling noemen we een context class. We gebruiken hiervoor een class die erft van DbContext. Aan de hand van deze context class spreken we de database aan :

- gegevens opvragen uit de database
- gegevens toevoegen aan de database
- gegevens aanpassen in de database
- gegevens verwijderen uit de database
- connecteren met de database
- het model en de configuraties instellen
- configuratie van de change tracking
- caching
- transaction management

Een DbContext-class implementeert de interface IDisposable. Dit betekent dat je een DbContext-object moet 'opkuisen' na gebruik. Indien je het DbContext-object aanmaakt met het sleutelwoord using, dan gebeurt deze opkuis automatisch. Intern gebruikt een DbContext databaseconnecties. Bij de opkuis van de DbContext sluit .NET deze connecties. Het is belangrijk de DbContext niet langer dan noodzakelijk levend te houden, zodat .NET connecties vlot kan sluiten.

Onze context bevat **DbSet** property's, één voor elke entity die gemapt wordt aan een tabel uit de database. Een **DbSet** is **een collectie van objecten van een bepaalde entity**.

We proberen dit uit in ons project.

- Maak in het project *Model* in de map *Entities* een class met de naam *EFOpleidingenContext* :

```
public class EFOpleidingenContext : DbContext           (1)
{
    public DbSet<Campus> Campussen { get; set; }         (2)
    public DbSet<Docent> Docenten { get; set; }
}
```

- (1) Je maakt een class die afgeleid is van DbContext.
- (2) Je maakt per entity class een property in je DbContext class. Deze property is van het type DbSet<TEntity>.

5.6 De connectionstring

Om een connectie te kunnen maken met een database hebben we 3 gegevens nodig : de naam van de databaseserver, de naam van de database en de usergegevens. Deze 3 gegevens gebruiken we eerder al een connectie te maken met een database. In een programma kunnen we die gegevens niet intikken maar slaan we die op in een zogenaamde connectionstring. Op de website

<https://www.connectionstrings.com/>

krijg je een overzicht van connectionstrings voor diverse databaseservers.

Je kan in een programma de connectionstring hardcoded opnemen in de code of opslaan in een configuratiebestand.

Een hard-coded connectionstring (niet safe in productie!)

- Voeg aan de class EFOPleidingenContext.cs onderstaande method *OnConfiguring()* toe :

```
public class EFOPleidingenContext : DbContext
{
    ...
    protected override void OnConfiguring(DbContextOptionsBuilder
        optionsBuilder)
    {
        optionsBuilder.UseSqlServer(
            "Server=(localdb)\\mssqllocaldb;Database=EFOPleidingen;Trusted_Connection=true;" (1)
            ,options => options.MaxBatchSize(150)); (2) (3)
    }
}
```

- (1) Met UseSqlServer() geven we aan dat we een SQL Server database gaan gebruiken. Deze method geef je een connectionstring mee als parameter en eventuele options.
- (2) In de connectionstring geven we aan dat we localdb gebruiken als database server (Server=(localdb)\\mssqllocaldb), EFOPleidingen als database (Database=EFOPleidingen) en dat we inloggen met de momenteel ingelogde windowsgebruiker (Trusted_Connection=true).
- (3) Met de method MaxBatchSize() geven we het maximaal aantal SQL commando's aan dat in één keer naar de database kan gestuurd worden.

Een connectionstring uit appsettings.json halen

We nemen de connectionstring op in een appsettings.json bestand. Op die manier kan je de connectionstring wijzigen zonder dat je de code moet hercompileren.

Om het bestand appsettings.json te kunnen inlezen moet je een extra NuGet package installeren, namelijk de NuGet package `Microsoft.Extensions.Configuration.Json`.

- Voeg versie 3.1.5 van deze package toe aan het project Model.

Je voegt nu de appsettings.json file toe.

- Klik met de rechtermuistoets op het project Model en kies Add-New Item...
- Kies in de categorie Web voor een JSON File en noem deze appsettings.json.
- Voeg er onderstaande code aan toe :

```
{
  "ConnectionStrings": {
    "EFopleidingen":
      "Server=(localdb)\\MSSQLLocalDB;Database=EFopleidingen;Trusted_Connection=True;"
  }
}
```

Standaard wordt dit nieuwe configuratiebestand na een build van het project niet mee gekopieerd naar de outputdirectory. We passen dit aan.

- Selecteer in de Solution Explorer het bestand appsettings.json.
- Pas in het propertiesvenster de property Copy to Output Directory aan naar **Copy always**.

Je voegt nu de nodige code toe in de DbContext-class om de connectionstring in te lezen uit het appsettings.json bestand.

- Wijzig de class EFopleidingenContext.cs als volgt :

```
public class EFopleidingenContext : DbContext
{
    public static IConfigurationRoot configuration;                                (1)
    public DbSet<Campus> Campussen { get; set; }
    public DbSet<Docent> Docenten { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder
        optionsBuilder)
    {
        configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetParent(AppContext.BaseDirectory).FullName)
            .AddJsonFile("appsettings.json", false)
            .Build();                                                            (2)
        var connectionString =
            configuration.GetConnectionString("efopleidingen");                (3)
        if (connectionString != null)
        {
            optionsBuilder.UseSqlServer(connectionString,                        (4)
                options => options.MaxBatchSize(150));
        }
    }
}
```

- (1) Om de configuratiegegevens op te halen gebruiken we een IConfigurationRoot.
- (2) We geven de naam en de plaats op van het appsettings.json bestand.
- (3) We halen de connectionstring uit het configuratiebestand.
- (4) En gebruiken deze string i.p.v. de hardcoded connectionstring.

6 Migraties

We beschikken nu over een Entity Data Model. Op basis van dit model kunnen op de database server een nieuwe database laten maken. Ook wanneer dit model wijzigt kunnen we de nodige wijzigingen in de database laten doorvoeren. Wijzigingen worden bewaard in een migration. Dit is een C#-script waarin code staat die de wijzigingen doorvoert (een method Up) of terug ongedaan (een method Down) maakt.

6.1 Een eerste migration

We beginnen met het maken van een eerste migration die ervoor zal zorgen dat de database wordt aangemaakt. We gebruiken hiervoor een tool. Met het commando dotnet ef migrations add zullen we een eerste migration aanmaken en die vervolgens uitvoeren met het commando dotnet ef database update.

Om de Migratie te kunnen opzetten met commando's in de NuGet Package Manager console, moeten we nog een NuGet package toevoegen, namelijk de package Microsoft.EntityFrameworkCore.Design.

- Kies in het menu Tools – NuGet Package Manager – Manage NuGet Packages for Solution.
- Installeer de package Microsoft.EntityFrameworkCore.Design voor het project Model.
- Kies versie 3.1.11 en klik op Install. We kunnen nu de tool gebruiken om een eerste migration aan te maken.
- Build de solution.
- Start de Package Manager Console.
- Normaal bevind je je nu in de folderstructuur in de map waar de solutionfile zich bevindt.

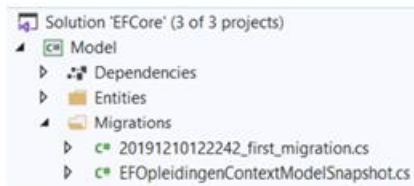
Het commando dat de migration aanmaakt moet uitgevoerd worden in de root van het project Model.

- Voer het onderstaande commando uit om de migration aan te maken :

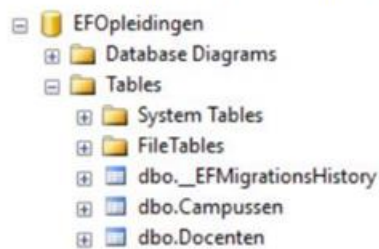
Add-Migration Initial

Het resultaat van deze opdracht is een bestand EFopleidingenContextModelSnapshot.cs en een bestand waarvan de naam begint met de huidige datum en tijd en eindigt met de naam van de migration.

Beide bestanden zijn opgeslagen in een folder **Migrations** in het project Model.



- Tik in de Package Manager Console het commando `Update-database` om de migration uit te voeren. De database wordt nu gemaakt.
- Start SSMS, refresh de databaselijst en bekijk het resultaat.



6.2 Data annotations

Bij de aanmaak van de eerste migration heeft EF Core migratie tool zich gebaseerd op standaard conventies om de namen van de tabellen te kiezen, de namen en types van de columns, de keuze van de primary key's, enz. Wil je van deze standaard conventies afwijken dan kan dat door gebruik te maken van **annotations** of **Fluent API code**

6.2.1 De naam van de tabel instellen

Standaard zal EF de naam van de DbSet in de DbContext gebruiken voor de naam van de bijhorende table in de database.

Indien je hiervan wil afwijken dan gebruik je de annotation **[Table]** om de tabel een andere naam te geven.

Voorbeeld:

```
[Table("Naties")]
public class Land
{
    ...
}
```

6.2.2 De naam van een kolom instellen

De naam van de kolom die hoort bij een property is standaard dezelfde als de naam van de property. Vind je deze naam niet goed dan kan je deze instellen met het attribuut **[Column]**.

Voorbeeld:

```
public class Docent
{
    ...
    [Column("Maandwedde")]
    public decimal Wedde { get; set; }
}
```

6.2.3 Een kolom instellen als (niet) verplicht in te vullen

Wil je de kolom, die bij een property hoort, instellen als verplicht in te vullen, dan voeg je het attribuut **[Required]** toe aan de property. Een kolom die hoort bij een property met een primitief data-type is standaard verplicht in te vullen. Je kan hiervan afwijken door de property **nullable** te maken. Voorbeeld :

```
public class Docent
{
    ...
    [Required]
    public string Voornaam { get; set; }
    public bool? HeeftRijbewijs { get; set; }
    ...
}
```

6.2.4 Het maximum aantal tekens in een varchar-kolom instellen

Je kan met het attribuut **[StringLength]** het maximum aantal tekens in een varchar-kolom instellen. Voorbeeld :

```
public class Campus
{
    ...
    [StringLength(50)]
    public string Gemeente { get; set; }
    ...
}
```

6.2.5 Het kolomtype instellen

Je kan met het attribuut **[Column]** een afwijkend kolomtype instellen. Bij een DateTime property hoort standaard een datetime-kolom in de databasetabel. Je kan dit veranderen in een date-kolom via onderstaande notatie.

Voorbeeld :

```
public class Docent
{
    ...
    [Column(TypeName = "date")]
    public DateTime InDienst { get; set; }
    ...
}
```

6.2.6 De primary key instellen

Als EF een property vindt met de naam `Id`, aanziet EF deze property als de primary key. Vindt EF deze kolom niet maar wel een property waarvan de naam gelijk is aan de naam van de class, gevolgd door `Id`, dan aanziet EF deze property als de primary key. Je kan hiervan afwijken door het attribuut `[Key]` te plaatsen voor de property die jij als primary key wil instellen.

Voorbeeld:

```
public class Land
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.None)]
    public string Code { get; set; }
    ...
}
```

6.2.7 De foreign key instellen

Wanneer jouw sleutels eindigen op `Id` zoals in ons Opleidingen-voorbeeld, bepaalt EF dus zelf welk veld de primary key is. EF doet hetzelfde voor de foreign key. Eindigt een potentiële foreign key niet op `Id` dan moet je zelf de foreign key aanduiden. Je doet dit door boven de navigation property een annotation te plaatsen met als parameter de naam van het foreign key veld.

Voorbeeld:

```
public class Inwoner
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.None)]
    public string RijksregisterNr { get; set; }
    ...
    public string Code { get; set; }
    [ForeignKey("Code")]
    public virtual Land Land { get; set; }
}
```

Er is een één-veel relatie tussen `Land` en `Inwoner`. In de class `Inwoner` geven we aan dat het veld `Code` de foreign key is door boven de navigation property `Land` de annotation `[ForeignKey("Code")]` te plaatsen. Als alternatief kan je ook een `ForeignKey`-annotation plaatsen boven het foreign key veld. Als parameter geef je dan de naam van de navigation property mee.

6.2.8 Een property niet opnemen in de databasetable

Je kan aangeven dat een property niet moet opgenomen worden in de databasetable door er het attribuut `[NotMapped]` boven te plaatsen.

Voorbeeld:

```
public class Campus
{
    ...
    [NotMapped]
    public string NietOpnemen { get; set; }
}
```

6.2.9 Oefening

Pas een aantal van deze attributen toe in de EFCore app.

- Wijzig de entiteiten Campus en Docent zoals hieronder weergegeven. Voeg ook de entiteit Land toe:

```
[Table("Campussen")]
public class Campus
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int CampusId { get; set; }
    [Required]
    [Column("CampusNaam")]
    public string Naam { get; set; }
    public string Straat { get; set; }
    public string Huisnummer { get; set; }
    public string Postcode { get; set; }
    [StringLength(50)]
    public string Gemeente { get; set; }
    [NotMapped]
    public string Commentaar { get; set; }
    public virtual ICollection<Docent> Docenten { get; set; }
}
```

```
[Table("Docenten")]
public class Docent
{
    [Key]
    public int DocentId { get; set; }
    [Required]
    [MaxLength(20)]
    public string Voornaam { get; set; }
    [Required]
    [MaxLength(30)]
    public string Familienaam { get; set; }
    [Column("Maandwedde", TypeName = "decimal(18,4)")]
    public decimal Wedde { get; set; }
    [Column(TypeName = "date")]
    public DateTime InDienst { get; set; }
    public bool? HeeftRijbewijs { get; set; }
    [ForeignKey("Land")]
    public string LandCode { get; set; }
    [ForeignKey("Campus")]
    public int CampusId { get; set; }
    public virtual Campus Campus { get; set; }
    public virtual Land Land { get; set; }
}
```

```
[Table("Landen")]
public class Land
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.None)]
    public string LandCode { get; set; }
    public string Naam { get; set; }
    public virtual ICollection<Docent> Docenten { get; set; }
}
```

- Voeg nu ook aan EF0pleidingenContext.cs een DbSet<Land> toe:

```
public class EF0pleidingenDbContext:DbContext
{
    public static IConfigurationRoot configuration;

    public DbSet<Campus> Campussen { get; set; }
    public DbSet<Docent> Docenten { get; set; }
    public DbSet<Land> Landen { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder
        optionsBuilder)
    {
        configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetParent(AppContext.BaseDirectory).FullName)
            .AddJsonFile("appsettings.json", false)
            .Build();
        var connectionString =
            configuration.GetConnectionString("efopleidingen");
        if (connectionString != null)
        {
            optionsBuilder.UseSqlServer(connectionString,
                options => options.MaxBatchSize(150));
        }
    }
}
```

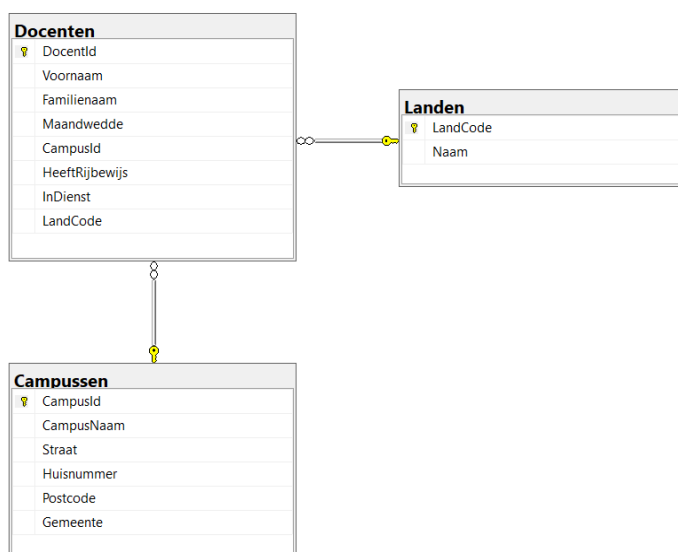
- Tik in de Package Manager Console onderstaand commando om een nieuwe migration aan te maken :

Add-Migration Oef_DataAnnotations

- Voer de migration door met het commando

Update-Database

- Open SSMS en open een connectie op (localdb)\mssqlLocalDb
- Maak een Database diagram met de 3 gecreëerde tabellen :



6.3 Enums en partial classes

EF kan ook overweg met enums en partial classes. We proberen één en ander uit. We voegen aan de class `Docent` een property **Geslacht** toe. Deze property is van het type `Geslacht`, een **enum**-type met `Man`, `Vrouw` en `X` als mogelijke waarden. Daarnaast maken we van de class `Docent` een partial class. We voegen aan de **class `Docent` in een tweede sourcefile, `DocentExtension.cs`**, een property `Naam` toe die bestaat uit de voor- en familienaam van de docent en een property `Geboortedatum`.

- Maak van de class `Docent.cs` een **partial class** en **voeg** een property **Geslacht** toe :

```
public partial class Docent
{
    ...
    public Geslacht Geslacht { get; set; }
}
```

- Voeg in een nieuwe (**partial**) class `DocentExtension.cs` de property's `Naam` en `Geboortedatum` toe :

```
public partial class Docent
{
    public string Naam
    {
        get { return Voornaam + " " + Familienaam; }
    }
    [Column(TableName = "date")]
    public DateTime Geboortedatum { get; set; }
}
```

- Maak in het project `Model` in de folder `Entities` een nieuwe codefile `Geslacht.cs` aan en voeg er volgende code aan toe :

```
public enum Geslacht
{
    Man,
    Vrouw,
    X
}
```

We maken een nieuwe migration aan en voeren die door.

- Tik in de Package Manager Console het commando

Add-Migration `geslachtenaam`.

Een nieuwe migration wordt aangemaakt.

- Voer de migration door met het commando

Update-database

Bekijk het resultaat in SSMS :

in de tabel Docenten is een veld Geslacht toegevoegd met als kolomtype een **int**. Daarnaast is er **geen** extra veld **Naam** toegevoegd maar wel een veld **Geboortedatum**.

6.4 Fluent API

Fluent API is een alternatief én een uitbreiding van annotations om de mapping tussen entityclasses en tabellen uit te drukken. Je configureert de mappings in de contextclass door de method **OnModelCreating()** te **overriden**. De configuratie die je via Fluent API uitdrukt heeft voorrang op eventuele annotations.

6.4.1 Property mapping

We starten met de mapping van property's. Zoals gezegd kunnen we het standaard gedrag van EF overschrijven door de mapping via code uit te drukken in de method **OnModelCreating()**. Het instellen van de primary key gebeurt via conventie door de naam van de property te bekijken en vast te stellen of deze bestaat uit de naam van de entity en 'id'. Een alternatief is de annotation **[Key]**. Met fluent API doe je dit met de volgende code :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Campus>()
        .HasKey(c => c.CampusId);
}
```

Met bovenstaande code geven we aan dat de entiteit Campus een sleutel heeft die bestaat uit de CampusId. Willen we op een entiteit een **samengestelde sleutel** toepassen dan gaat dit met volgende code :

```
modelBuilder.Entity<Orderlijn>()
    .HasKey(o => new { o.Order, o.Orderlijn });
```

We kunnen via de annotation **[DatabaseGenerated(...)]** aangeven of op een sleutelveld al dan niet **autonummering** moet toegepast worden.

Dit geven we aan in fluent API met volgende code :

```
modelBuilder.Entity<Land>()
    .Property(l => l.LandCode)
    .ValueGeneratedNever();    // geen autonummering

modelBuilder.Entity<Campus>()
    .Property(c => c.CampusId)
    .ValueGeneratedOnAdd();    // wel autonummering
```

De annotation `[StringLength]` kunnen we vervangen door de method `HasMaxLength()` :

```
modelBuilder.Entity<Docent>()  
    .Property(d => d.Voornaam)  
    .HasMaxLength(20);
```

Met de method `IsRequired()` geven we aan dat het veld verplicht is in te vullen :

```
modelBuilder.Entity<Docent>()  
    .Property(d => d.Voornaam)  
    .IsRequired();
```

Je kan op één of meerdere velden een index plaatsen zodat er sneller op kan gezocht worden. Je kan deze index bovendien uniek maken. Met onderstaande code voeg je een **unieke index** toe.

```
modelBuilder.Entity<Docent>()  
    .HasIndex(d => d.Emailadres)  
    .IsUnique();
```

Je kan met fluent API ook aangeven dat een property **niet gemapt** moet worden :

```
modelBuilder.Entity<Campus>()  
    .Ignore(c => c.Commentaar);
```

De method `HasColumnName` geeft een property een afwijkende veldnaam.

```
modelBuilder.Entity<Docent>()  
    .Property(d => d.Wedde)  
    .HasColumnName("Maandwedde");
```

Tenslotte kunnen we ook het gegevenstype aangeven met de method `HasColumnType()`

```
modelBuilder.Entity<Docent>()  
    .Property(d => d.Wedde)  
    .HasColumnType("decimal(18, 4)");
```


6.4.2 Type mapping

Met type mapping overschrijven we het standaard gedrag van EF Core voor de mapping van een class. Onderstaande code zorgt ervoor dat een entity **Test** niet wordt gemapt :

```
modelBuilder.Ignore<Test>();
```

We geven een tabel een afwijkende naam met volgende code :

```
modelBuilder.Entity<Campus>().ToTable("Campussen");
```

6.5 Oefening

Creëer in de solution Taken op basis van het entity data model de tabellen Banken en Rekeningen in de database **EFBank**. Voor de veld eigenschappen en de mapping kan je een mix gebruiken van de standaard conventies, annotations en fluent api.

Klant

Property	type	Omschrijving
KlantNr	integer	PK
Voornaam	string	Not null

Rekening

Property	type	Omschrijving
RekeningNr	string	PK
KlantNr	integer	FK, not null
Saldo	decimal	not null
Soort	char	not null

6.6 Seeding

We kunnen de tabellen van de database via code voorzien van initiële gegevens. Dit kunnen eventueel ook testgegevens zijn. We noemen dit seeding van de database. Om één of meerdere tabellen te seeden voeg je code toe in de contextclass in de method **OnModelCreating()**. Je past de method **HasData()** toe op de modelBuilder om een record toe te voegen aan een tabel.

Opgelet : sleutelwaarden moeten altijd expliciet meegegeven worden, ook wanneer er autonummering voorzien is voor een veld.

- Voeg onderstaande code toe in de contextclass **EFOpleidingenContext.cs** :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Campus>().HasData(
        new Campus
        {
            CampusId = 1,
            Naam = "Andros",
            Straat = "Somersstraat",
            Huisnummer = "22",
            Postcode = "2018",
            Gemeente = "Antwerpen"
        },
        new Campus
        {
            CampusId = 2,
            Naam = "Delos",
            Straat = "Oude Vest",
            Huisnummer = "17",
            Postcode = "9200",
            Gemeente = "Dendermonde"
        },
        new Campus
        {
            CampusId = 3,
            Naam = "Gavdos",
            Straat = "Europalaan",
            Huisnummer = "37",
            Postcode = "3600",
            Gemeente = "Genk"
        },
        new Campus
        {
            CampusId = 4,
            Naam = "Hydra",
            Straat = "Interleuvenlaan",
            Huisnummer = "2",
            Postcode = "3001",
            Gemeente = "Heverlee"
        },
        new Campus
        {
            CampusId = 5,
            Naam = "Ikaria",
            Straat = "Vlamingstraat",
            Huisnummer = "10",
            Postcode = "8560",
            Gemeente = "Wevelgem"
        },
        new Campus
        {
            CampusId = 6,
            Naam = "Oinouses",
            Straat = "Akkerstraat",
            Huisnummer = "4",
            Postcode = "8400",
            Gemeente = "Oostende"
        }
    );
    modelBuilder.Entity<Land>().HasData(
        new Land { LandCode = "BE", Naam = "België" },

```

```

new Land { LandCode = "NL", Naam = "Nederland" },
new Land { LandCode = "DE", Naam = "Duitsland" },
new Land { LandCode = "FR", Naam = "Frankrijk" },
new Land { LandCode = "IT", Naam = "Italië" },
new Land { LandCode = "LU", Naam = "Luxemburg" }
);
modelBuilder.Entity<Docent>().HasData(
new Docent
{
    DocentId = 001,
    Voornaam = "Willy",
    Familiennaam =
"Abbeloos",
    Wedde = 1400m,
    HeeftRijbewijs = new Nullable<bool>(),
    InDienst = new DateTime(2019, 1, 1),
    CampusId = 4
},
new Docent
{
    DocentId = 002,
    Voornaam = "Joseph",
    Familiennaam =
"Abelshausen",
    Wedde = 1800m,
    HeeftRijbewijs = true,
    InDienst =
new DateTime(2019, 1, 2),
    CampusId = 2
},
new Docent
{
    DocentId = 003,
    Voornaam = "Joseph",
    Familiennaam =
"Achten",
    Wedde = 1300m,
    HeeftRijbewijs = false,
    InDienst =
new DateTime(2019, 1, 3),
    CampusId = 3
},
new Docent
{
    DocentId = 004,
    Voornaam = "François",
    Familiennaam =
"Adam",
    Wedde = 1700m,
    HeeftRijbewijs = new Nullable<bool>(),
    InDienst = new DateTime(2019, 1, 4),
    CampusId = 1
},
new Docent
{
    DocentId = 309,
    Voornaam = "Jozef",
    Familiennaam =
"Wouters",
    Wedde = 1100,
    HeeftRijbewijs = true,

```

```

        InDienst =
            new DateTime(2019, 11, 7),
            CampusId = 1
    }
    );
}

```

Hierboven zijn slechts een aantal docentgegevens afgebeeld. Je vindt de code voor de volledige lijst in het bestand EFOpleidingenContext.cs bij de oefenbestanden. Neem ze zeker allemaal over, je hebt ze later nodig om een aantal query's uit te proberen.

We voegen nu een migration toe die deze data zal toevoegen in de database en updaten daarna de database.

- Tik in de Package Manager Console het commando

Add-Migration seeding

Een nieuwe migration wordt aangemaakt.

- Voer de migration door met het commando

Update-Database

- Je kan de inhoud van de tabellen bekijken in SSMS.

De tabellen beschikken nu over de gewenste startdata.

Wanneer je de data in de method **OnModelCreating()** verandert door gegevens te wijzigen, te verwijderen of toe te voegen en een nieuwe migration aanmaakt dan worden de gegevens in de bijhorende databasetabellen navenant aangepast. Wis dus zeker niet zomaar alle data in **OnModelCreating()** of alle gegevens worden bij een volgende migration gewist !

We proberen enkele wijzigingen uit.

- Voeg in de lijst met landen Groot-Brittannië toe met landcode GB :

```

modelBuilder.Entity<Land>().HasData(
    ...
    new Land { LandCode = "GB", Naam = "Groot-Brittannië" }
);

```

- Schrap de code voor docent 309, Jozef Wouters in de lijst met docenten :

```

modelBuilder.Entity<Docent>().HasData(
    ...
new Docent { DocentId = 309, Voornaam = "Jozef", ... }
);

```

- Wijzig de gemeentenaam voor campus 2, Delos, in Wondelgem, de postcode in 9032 :

```
modelBuilder.Entity<Campus>().HasData(
    ...
    new Campus
    {
        CampusId = 2,
        Naam = "Delos",
        Straat = "Oude Vest",
        Huisnummer = "17",
        Postcode = "9032",
        Gemeente = "Wondelgem"
    },
    ...
);
```

- Tik in de Package Manager Console het commando

Add-Migration seeding_change

Een nieuwe migration wordt aangemaakt.

- Bekijk de inhoud van de nieuwe migrationsfile in de folder Migrations.

Enkel de wijzigingen t.o.v. de vorige migration zijn opgenomen.

- Voer de migration door met het commando

Update-database

Wanneer je de inhoud van een sleutelveld verandert moet je oppassen. De migration wijzigt enkel de waarde in de primary key maar niet de foreign key in gerelateerde tabellen.

Méér nog : de gerelateerde records in de docententabel worden eerst verwijderd en vervolgens terug toegevoegd! Dit komt omdat de campus met id 6 eerst wordt verwijderd en daarvoor mogen er geen bijhorende docenten bestaan. Die worden dus ook gewist. Na de verwijdering van de campus wordt deze terug toegevoegd met de correcte id. De gewiste docenten worden terug toegevoegd maar met de oude id!

- Wijzig de campusid voor campus 6, Oinouses, in 7 :

```
modelBuilder.Entity<Campus>().HasData(
    ...
    new Campus
    {
        // CampusId = 6
        CampusId = 7,
        Naam = "Oinouses",
        ...
        Gemeente = "Oostende"
    }
);
```

6.7 Oefening

Voeg in EFBank via seeding onderstaande data toe aan de tabellen Klanten en Rekeningen.

Klanten:

KlantNr	KlantVoornaam
1	Marge
2	Homer
3	Lisa
4	Maggie
5	Bart

Rekeningen:

RekeningNr	KlantNr	Saldo	Soort
123-4567890-02	1	1000	Z
234-5678901-69	1	2000	S
345-6789012-12	2	500	S

6.8 LINQ-to-Entity query's

Records lezen met een foreach zoals in de vorige paragraaf is beperkt in zijn mogelijkheden : je kan niet sorteren of filteren. We kunnen wel een LINQ query uitvoeren op de property Docenten. EF Core zet dan volgende stappen :

- EF Core vertaalt de LINQ query naar een SQL select statement
- EF Core stuurt dit SQL statement naar de database om records uit de bijhorende tabel op te vragen
- EFCore maakt van ieder gevonden record een entity
- EFCore verzamelt deze entity's in een verzameling

Je kan met een foreach itereren over deze verzameling We proberen dit uit.

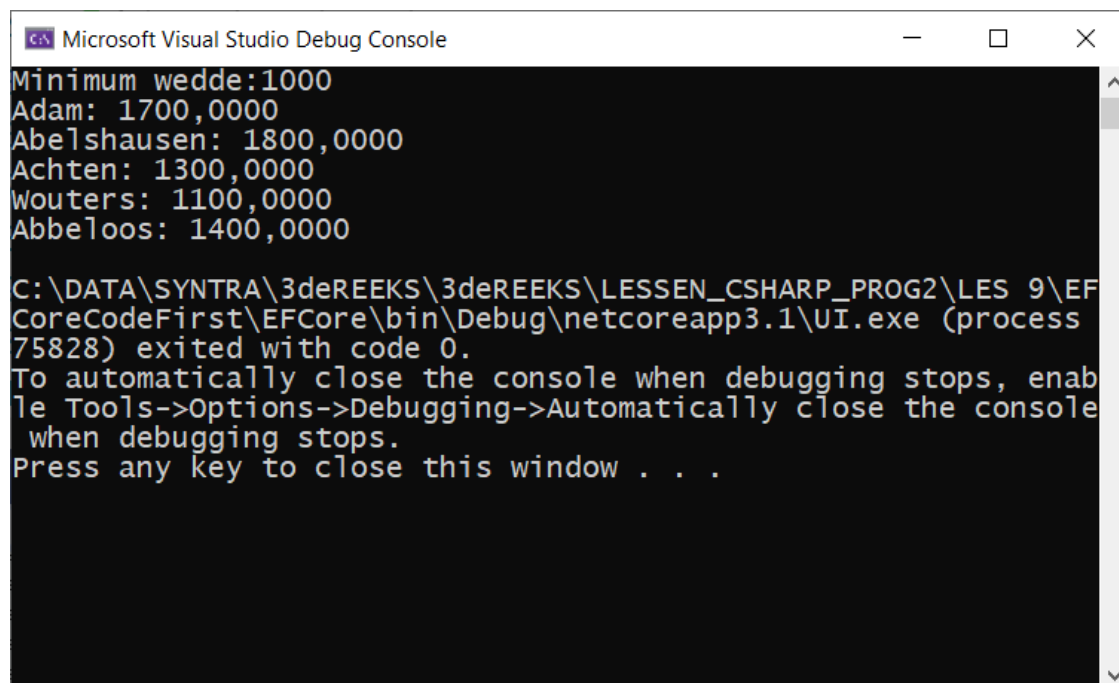
- Zet de aanwezige code in de Main()-method in Program.cs in commentaar.
- Voeg in Main() onderstaande code toe :

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Minimum wedde:");
        if (decimal.TryParse(Console.ReadLine(), out decimal minWedde))
        {
            using (var context = new EF0pleidingenDbContext())
            {
                var query = from docent in context.Docenten
                            where docent.Wedde >= minWedde
                            orderby docent.Voornaam, docent.Familienaam
                            select docent;
                foreach (var docent in query)
                {
                    Console.WriteLine("{0}: {1}", docent.Familienaam, docent.Wedde);
                }
            }
        }
        else
        {
            Console.WriteLine("Geef een getal in !");
        }
    }
}
```

- Voeg bovenaan de code onderstaand using-statement toe :

```
using System.Linq;
```

- Probeer het programma uit en bekijk het SQL statement via de logging-informatie.



```
Microsoft Visual Studio Debug Console
Minimum wedde:1000
Adam: 1700,0000
Abelshausen: 1800,0000
Achten: 1300,0000
Wouters: 1100,0000
Abbeloos: 1400,0000

C:\DATA\SYNTRA\3deREEKS\3deREEKS\LESSEN_CSHARP_PROG2\LES 9\EF
CoreCodeFirst\EFCore\bin\Debug\netcoreapp3.1\UI.exe (process
75828) exited with code 0.
To automatically close the console when debugging stops, enable
Tools->Options->Debugging->Automatically close the console
when debugging stops.
Press any key to close this window . . .
```

6.9 Query-methods

Naast LINQ kan je ook query-methods zoals **Where** en **OrderBy** gebruiken om een query te definiëren.

- Wijzig in het programma de query-definitie als volgt :

```
var query = context.Docenten
    .Where(docent => docent.Wedde >= minWedde)    (1)
    .OrderBy(docent => docent.Voornaam)           (2)
    .ThenBy(docent => docent.Familienaam);        (3)
```

- (1) Je gebruikt de method Where om records te filteren. Je geeft een lambda-expressie mee, waarin je de filter definieert.
- (2) Je gebruikt de method OrderBy om records te sorteren. Je geeft een lambda-expressie mee, waarin je sortering definieert.
- (3) Als je op meerdere properties wil sorteren (in dit voorbeeld op voornaam én familienaam), pas je op het resultaat van de OrderBy-method de method ThenBy toe.

- Probeer opnieuw uit.

De methods OrderBy en ThenBy sorteren oplopend. De methods OrderByDescending en ThenByDescending sorteren aflopend.

6.10 LINQ-query's vs. query methods

LINQ-query's zijn meestal leesbaarder dan query's met querymethods. Sommige programmaonderdelen zijn meer onderhoudbaar met query-methods dan met LINQqueries. In het volgende voorbeeld ziet de gebruiker de docenten met een wedde vanaf een in te tikken grens. De gebruiker kiest daarna hoe hij die docenten sorteert. We tonen eerst het programma met een LINQ-query's. Het bevat drie sterk gelijkaardige query's

```
static void Main(string[] args)
{
    Console.WriteLine("Minimum wedde:");
    if (decimal.TryParse(Console.ReadLine(), out decimal minWedde))
    {
        Console.WriteLine("Sorteren:1=op wedde, 2=op familienaam, 3=op voornaam:");
        var sorterenOp = Console.ReadLine();
        using (var context = new EF0pleidingenDbContext())
        {
            IQueryable<Docent> query; //(1)
            switch (sorterenOp)
            {
                case "1":
                    query = from docent in context.Docenten //(2)
                        where docent.Wedde >= minWedde
                        orderby docent.Wedde
                        select docent;

                    break;
                case "2":
                    query = from docent in context.Docenten //(2)
                        where docent.Wedde >= minWedde
                        orderby docent.Familienaam
                        select docent;

                    break;
                case "3":
                    query = from docent in context.Docenten //(2)
                        where docent.Wedde >= minWedde
                        orderby docent.Voornaam
                        select docent;

                    break;
                default:
            }
        }
    }
}
```



```

        Console.WriteLine("Verkeerde keuze");
        query = null;
        break;
    }
    if (query != null)
        foreach (var docent in query)
            Console.WriteLine("{0}: {1}", docent.Familienaam, docent.Wedde);
    else
        Console.WriteLine("U tikte geen getal");
    }
}
}

```

- (1) Het type van de variabele query is een LINQ query die Docent-entity's teruggeeft.
- (2) De drie query's in het switch-statement lijken sterk op elkaar.

De versie met query-methods bevat maar één query-definitie :

```

static void Main(string[] args)
{
    Console.Write("Minimum wedde:");
    if (decimal.TryParse(Console.ReadLine(), out decimal minWedde))
    {
        Console.Write("Sorteren:1=op wedde, 2=op familienaam, 3=op voor naam:");
        var sorterenOp = Console.ReadLine();
        Func<Docent, Object> sorteerLambda; //(1)
        switch (sorterenOp)
        {
            case "1":
                sorteerLambda = (docent) => docent.Wedde; //(2)
                break;
            case "2":
                sorteerLambda = (docent) => docent.Familienaam; //(2)
                break;
            case "3":
                sorteerLambda = (docent) => docent.Voor naam; //(2)
                break;
            default:
                Console.WriteLine("Verkeerde keuze");
                sorteerLambda = null;
                break;
        }
        if (sorteerLambda != null)
        {
            using (var context = new EF0pleidingenDbContext())
            {
                var query = context.Docenten
                    .Where(docent => docent.Wedde >= minWedde)
                    .OrderBy(sorteerLambda);
                foreach (var docent in query)
                    Console.WriteLine("{0} {1}: {2}", docent.Voor naam, docent.Familienaam, docent.Wedde);
            }
        }
        else
            Console.WriteLine("U tikte geen getal");
    }
}

```

- (1) We definiëren een Func-delegate.
- (2) We kennen er telkens een andere lambda aan toe

6.11 Een entity zoeken op basis van de PK-waarde

Je hoeft geen LINQ-query te schrijven om een entity te zoeken op zijn primary-key-waarde. Voor zo'n zoekoperatie kan je de **Find**-method gebruiken. Deze method retourneert de gezochte entiteit als de meegegeven keywaarde gevonden wordt, zoniet dan wordt er een null-waarde teruggegeven. In onderstaand voorbeeld gebruiken we de Find-method op de property Docenten, om een docent te zoeken op zijn docentnummer.

- Neem onderstaande code over :

```
static void Main(string[] args)
{
    using (var context = new EFOPLEIDINGENDbContext())
    {
        Console.WriteLine("DocentNr.:");
        if (int.TryParse(Console.ReadLine(), out int docentNr))
        {
            var docent = context.Docenten.Find(docentNr); // (1)
            Console.WriteLine(docent == null ? "Niet gevonden" : docent.Familienaam); // (2)
        }
        else
            Console.WriteLine("U tikte geen getal");
    }
}
```

(1) Je geeft het docentnummer mee aan de Find-method.

(2) Afhankelijk van het resultaat geef je de tekst "Niet gevonden" weer of de docentnaam.

6.12 Gedeeltelijke objecten ophalen

De query's die je tot nu toe maakte, lezen uit de records alle kolommen en vullen hiermee per entity alle bijbehorende property's. Dit kan de performantie benadelen als je in een programma-onderdeel slechts enkele property's per entity nodig hebt. Je gebruikt als oplossing een LINQ-query, waarin je slechts enkele property's opvraagt. EF Core vertaalt zo'n LINQ-query naar een SQL-select-statement dat enkel de kolommen leest die bij die property's horen.

- Neem onderstaand voorbeeld over :

```
static void Main(string[] args)
{
    using (var context = new EFOPLEIDINGENDbContext())
    {
        var query = from campus in context.Campussen
                    orderby campus.Naam
                    select new { campus.CampusId, campus.Naam }; //(1)

        foreach (var campusDeel in query)
            Console.WriteLine("{0}: {1}", campusDeel.CampusId,
                               campusDeel.Naam);
    }
}
```

(1) Het resultaat van deze query is een verzameling objecten.

Het type van deze objecten is een anonieme tijdelijke class met twee property's: CampusId en Naam.

Je kan deze query ook uitschrijven met query-methods. De code wordt dan :

```
var query = context.Campussen.OrderBy(campus => campus.Naam).Select(campus => new {  
    campus.CampusId, campus.Naam });
```

Groeperen in query's Je kan in een query de objecten groeperen. Je gebruikt hiervoor de sleutelwoorden group, by en into.

In onderstaand voorbeeld groepeer je de docenten op voornaam.

- Neem onderstaande code over :

```
using (var context = new EF0pleidingenDbContext())  
{  
    var query = from docent in context.Docenten //(1)  
                group docent by docent.Voornaam  
                into voornaamGroep  
                select new  
                {  
                    Voornaam = voornaamGroep.Key,  
                    Aantal = voornaamGroep.Count()  
                };  
    foreach (var voornaamStatistiek in query)  
    {  
        Console.Write(voornaamStatistiek.Voornaam + ": ");  
        Console.WriteLine(voornaamStatistiek.Aantal + " keer.");  
    }  
}
```

(1) Het resultaat van deze query is een verzameling objecten. Deze hebben twee property's : een voornaam (de key waarop gegroepeerd werd) en het aantal records per groep.

6.13 Lazy en Eager loading

Wanneer je de gegevens van docenten ophaalt heb je soms ook de gegevens van de bijhorende campussen nodig. Je kan deze bijkomende gegevens meteen inlezen of ze uitgesteld lezen. Het ene heet Eager Loading het andere Lasy Loading

6.13.1 Eager Loading

Eager loading is net het tegengestelde van lazy loading : je leest in één keer meteen alle informatie in.

Je stuurt slechts één request naar de database maar de resultset is wel veel groter. Je past eager loading toe door in de LINQ-query niet enkel de Docent-entity's te lezen maar ook al de geassocieerde Campus-entity's.

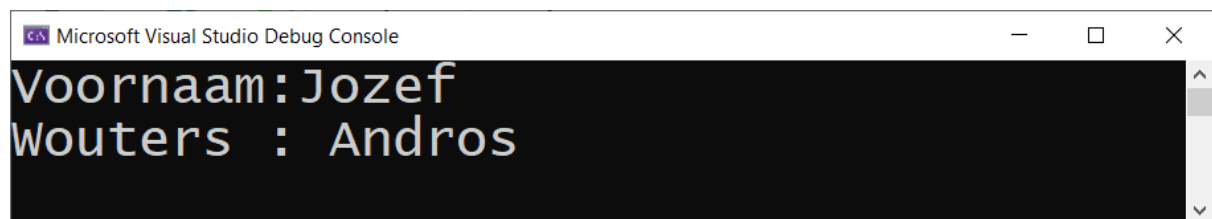
Je gebruikt hiervoor in de query de **Include()**-method. We proberen dit uit.

- Wijzig de code in het hoofdprogramma nu als volgt

using Microsoft.EntityFrameworkCore;

```
using (var context = new EF0pleidingenDbContext())
{
    Console.WriteLine("Voornaam:");
    var voornaam = Console.ReadLine();
    var query = from docent in context.Docenten.Include("Campus")//(1)
                where docent.Voornaam == voornaam
                select docent;
    foreach (var docent in query)
        Console.WriteLine("{0} : {1}", docent.Familienaam, docent.Campus.Naam);
}
```

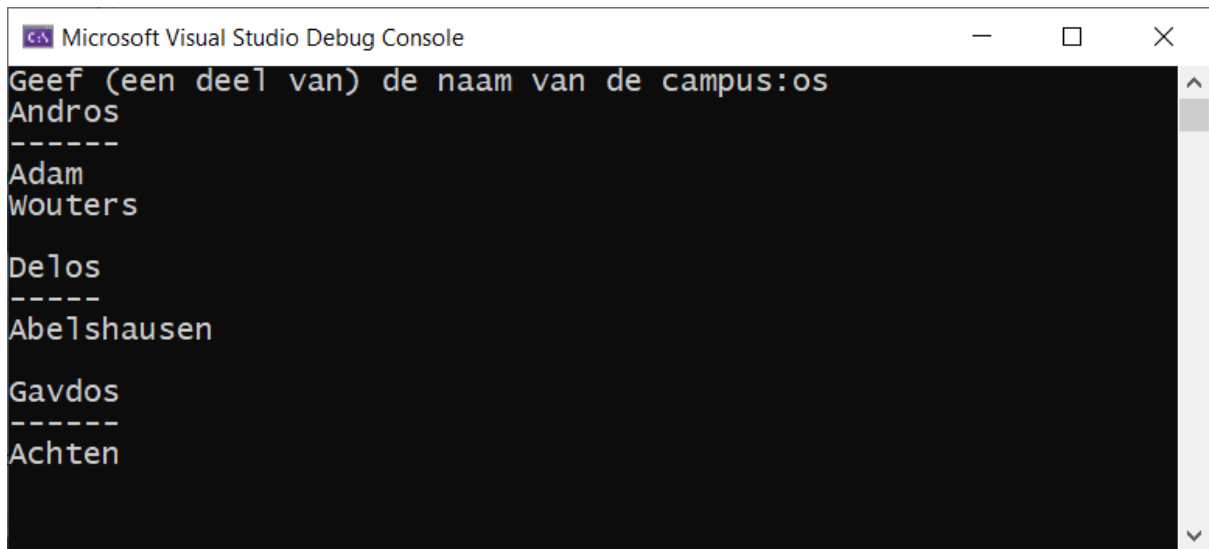
(1) Je gebruikt de method Include op de property Docenten. Als parameter geef je de naam van de associatie mee.



Een tweede voorbeeld : deze keer lezen we de campus-objecten waarvan de naam een zoekwoord bevat. We lezen meteen ook de bijhorende docenten.

- Neem onderstaande code over in het hoofdprogramma :

```
using (var context = new EF0pleidingenDbContext())
{
    Console.WriteLine("Geef (een deel van) de naam van de campus:");
    var deelNaam = Console.ReadLine();
    var query = from campus in context.Campussen.Include("Docenten")
                where campus.Naam.Contains(deelNaam)
                orderby campus.Naam
                select campus;
    foreach (var campus in query)
    {
        var campusNaam = campus.Naam;
        Console.WriteLine(campusNaam);
        Console.WriteLine(new string('-', campusNaam.Length));
        foreach (var docent in campus.Docenten)
            Console.WriteLine(docent.Familienaam);
        Console.WriteLine();
    }
}
```

A screenshot of the Microsoft Visual Studio Debug Console window. The window has a title bar with the Visual Studio icon and the text "Microsoft Visual Studio Debug Console". The console area has a black background with white text. The text shows a prompt "Geef (een deel van) de naam van de campus:os" followed by the input "Andros". Below this, a list of names is displayed: "Adam", "Wouters", "Delos", "Abelshausen", "Gavdos", and "Achten". Each name is preceded by a dashed line separator. The console window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
Microsoft Visual Studio Debug Console
Geef (een deel van) de naam van de campus:os
Andros
-----
Adam
Wouters

Delos
-----
Abelshausen

Gavdos
-----
Achten
```

Je kan deze query ook schrijven met query methods i.p.v. een LINQ query :

```
using (var context = new EF0pleidingenDbContext())
{
    Console.WriteLine("Geef (een deel van) de naam van de campus:");
    var deelNaam = Console.ReadLine();
    var query = context.Campussen.Include("Docenten")
        .Where(campus => campus.Naam.Contains(deelNaam))
        .OrderBy(campus => campus.Naam);
}
```

6.13.2 Lazy Loading.

Lazy Loading Bij Lazy Loading wordt het ophalen van de gegevens uit de databases dus uitgesteld tot het ogenblik waarop ze echt nodig zijn. Lazy Loading is niet standaard. Je moet expliciet aangeven dat je Lazy Loading wilt gebruiken. Lazy Loading heeft het voordeel dat je minder grote hoeveelheden gegevens in één keer inleest. Aan de andere kant betekent dit dan ook dat je meerdere requests naar de database stuurt.

6.13.2.1 Lazy Loading met proxies

We installeren eerst de NuGet package Microsoft.EntityFrameworkCore.Proxies.

- Start de NuGet package manager en installeer voor het project Model de package **Microsoft.EntityFrameworkCore.Proxies**. Kies versie 3.1.11

We activeren de proxies in de method OnConfiguring() in de class EFopleidingenDbContext.

- Open het bestand en voeg onderstaande vetgedrukte regel toe in de method **OnConfiguring()** :

```
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
    configuration = new ConfigurationBuilder()
        .SetBasePath(Directory.GetParent(AppContext.BaseDirectory).FullName)
        .AddJsonFile("appsettings.json", false)
        .Build();
    var connectionString =
        configuration.GetConnectionString("efopleidingen");
    if (connectionString != null)
    {
        optionsBuilder.UseSqlServer(connectionString,
            options => options.MaxBatchSize(150)).
            UseLazyLoadingProxies();
    }
}
```

Daarnaast moeten we alle navigation properties virtual maken.

- Controleer of in onderstaande classes de navigation properties virtual zijn.

In de class Docent :

```
public partial class Docent
{
    ...
    public int CampusId { get; set; }
    public virtual Campus Campus { get; set; }
}
```

In de class Campus :

```
public class Campus
{
    ...
    public virtual ICollection<Docent> Docenten { get; set; }
}
```

We proberen lazy loading nu uit.

- Neem onderstaande code over in Main() :

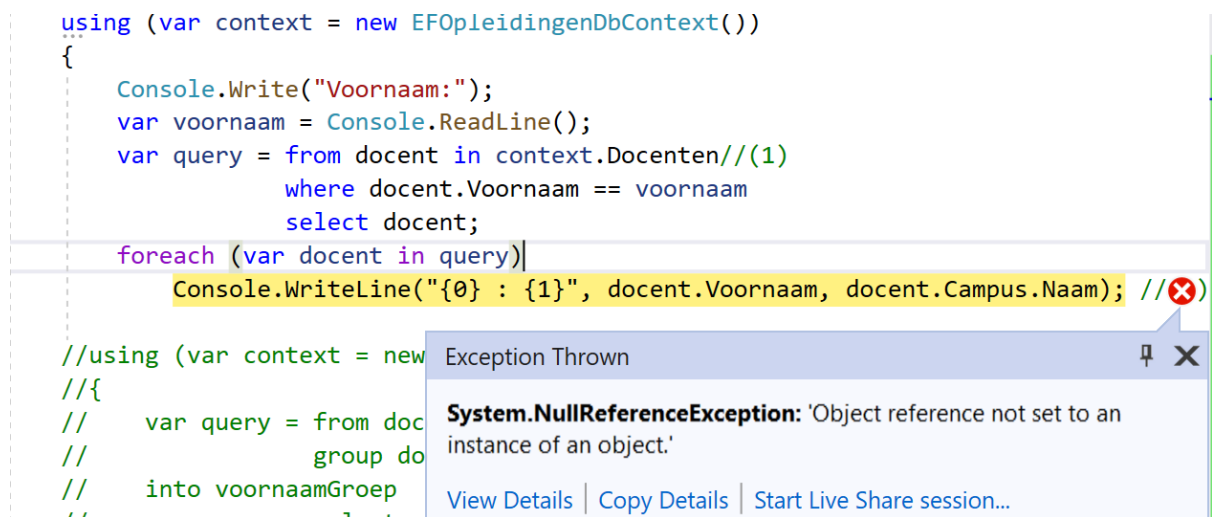
Plaats in de Main de volgende code en probeer de app te runnen

```
using (var context = new EFopleidingenDbContext())
{
    Console.WriteLine("Voornaam:");
    var voornaam = Console.ReadLine();
    var query = from docent in context.Docenten //(1)
                where docent.Voornaam == voornaam
                select docent;
    foreach (var docent in query)
        Console.WriteLine("{0} : {1}", docent.Voornaam, docent.Campus.Naam); //(2)
}
```

(1) Je leest enkel de docent-entity's.

(2) Je toont voor elke docent zijn naam en de campusnaam. Deze is nog niet beschikbaar dus stuurt EF Core een SQL-statement naar de database om de campus in te lezen.

Je krijgt je een foutmelding.



Oorzaak van de fout is dat je meerdere zogenaamde resultsets tegelijk hebt open staan. Eén voor de docenten en per docent nog eens een extra resultset voor de bijhorende campus. Als we dit willen doen moeten we een extra attribuut **multipleactiveresultsets** toevoegen aan de connectionstring en deze de waarde **true** geven.

- Wijzig de connectionstring in appSettings.json als volgt :

```
"Server=(localdb)\\mssqllocaldb;Database=EFopleidingen;
Trusted_Connection=true;MultipleActiveResultSets=true"
```

- Probeer opnieuw uit. Nu werkt de code zonder foutmeldingen

De docenten worden eerst ingelezen. Daarna wordt elke campus waarvoor info nodig is en die nog niet werd gelezen één keer ingelezen.

7 Entity's toevoegen

Behalve entiteiten uitlijsten kan je via EF Core ook entity's toevoegen. Je kan één of meerdere entity's toevoegen en ook bij het toevoegen meteen de associatie leggen met een gerelateerde entiteit.

7.1 Eén entiteit toevoegen

Om een entity toe te voegen in de database moet je volgende stappen zetten :

1. Je maakt een entity aan in het interne geheugen en je vult de property's van die entity in.
2. Je voegt deze entity toe aan de verzameling gelijkaardige entity's in de DbContext. Je doet dit met de method **Add()** van deze DbSet.

In ons voorbeeld beschikken de property's Docenten en Campussen uit EF0pleidingenDbContext over een method Add.

3. Je roept op de DbContext de method **SaveChanges()** op. EF Core stuurt op dat moment een insert-statement naar de database om de entity als een record toe te voegen.

We proberen dit uit.

- Voeg onderstaande code toe in het hoofdprogramma :

```
var campus = new Campus
{
    Naam = "Campus01",
    Straat = "Straat01",
    Huisnummer = "1",
    Postcode = "1111",
    Gemeente = "Gemeente01"
};
using (var context = new EF0pleidingenDbContext())
{
    context.Campussen.Add(campus); //(1)
    context.SaveChanges(); //(2)
    Console.WriteLine(campus.CampusId); //(3)
}
```

(1) Je voegt de entiteit toe aan de DbSet Campussen in de DbContext-class. De entiteit is nu nog niet opgeslagen in de database.

(2) Je slaat de entiteit op door de method SaveChanges() toe te passen op de contextclass.

(3) De campusid van de nieuwe campus wordt automatisch ingevuld en kan meteen opgevraagd worden.

De method SaveChanges() doet echter meer dan we vermoeden. Het controleert namelijk alle objecten van alle DbSets in de contextclass. Deze objecten hebben een bepaalde state. In bovenstaand voorbeeld heeft de nieuwe campus de state toegevoegd. Alle objecten met een state toegevoegd zullen via een insert-statement aan de database toegevoegd worden. Dit gebeurt in één transactie. Dit betekent dat als er één insert niet lukt er geen enkele insert doorgaat.

Je kan nu controleren in SSMS of in de Visual Studio Server Explorer of de campus goed is toegevoegd.

7.1.1 Meerdere entiteiten toevoegen

Je kan aan een DbSet meerdere entiteiten toevoegen door meerdere keren de Add()-method te gebruiken of je kan ze ook in één keer toevoegen met de AddRange()-method.

- Neem onderstaande code over in het hoofdprogramma :

```
var campus2 = new Campus
{
    Naam = "Campus02",
    Straat = "Straat02",
    Huisnummer = "2",
    Postcode = "2222",
    Gemeente = "Gemeente02"
};
var campus3 = new Campus
{
    Naam = "Campus03",
    Straat = "Straat03",
    Huisnummer = "3",
    Postcode = "3333",
    Gemeente = "Gemeente03"
};
var campus4 = new Campus
{
    Naam = "Campus04",
    Straat = "Straat04",
    Huisnummer = "4",
    Postcode = "4444",
    Gemeente = "Gemeente04"
};
var campus5 = new Campus
{
    Naam = "Campus05",
    Straat = "Straat05",
    Huisnummer = "5",
    Postcode = "5555",
    Gemeente = "Gemeente05"
};
using (var context = new EF0pleidingenDbContext())
{
    context.Campussen.AddRange(campus2, campus3); //(1)
    context.Campussen.AddRange(new List<Campus> { campus4, campus5 }); //(2)
    context.SaveChanges();
}
```

(1) Je voegt meerdere campussen toe door ze als parameters mee te geven aan de AddRange-method.

(2) Je voegt meerdere campussen toe door ze als een List van campussen mee te geven aan de AddRange-method.

Run de app

Je kan nu controleren in SSMS of in de Visual Studio Server Explorer of de campussen goed is toegevoegd.

7.2 Meerdere entiteiten van een verschillend type toevoegen

Je kan ook meerdere entiteiten in één keer toevoegen, ook al zijn ze van een verschillend type. Je doet dit eveneens met de **AddRange**-method.

- Neem onderstaande code over in het hoofdprogramma :

```
var campus6 = new Campus
{
    Naam = "Campus06",
    Straat = "Straat06",
    Huisnummer = "6",
    Postcode = "6666",
    Gemeente = "Gemeente06"
};
var docent1 = new Docent
{
    Familiennaam = "Docent01",
    Voornaam = "Voornaam01",
    Wedde = 1111,
    CampusId = 1
};
using (var context = new EFopleidingenDbContext())
{
    context.AddRange(campus6, docent1); //(1)
    context.SaveChanges();
}
```

(1) De campus en de docent worden samen toegevoegd.

Run de app

Je kan nu controleren in SSMS of in de Visual Studio Server Explorer of de campussen goed is toegevoegd.

7.3 Entiteiten én bijhorende nieuwe geassocieerde entiteiten toevoegen

Om een nieuwe entity met een nieuwe, bijhorende geassocieerde entity op te slaan volstaat het één van beide entity's toe te voegen aan de DbContext via de Add-method. Wanneer je daarna de SaveChanges-method uitvoert, voegt EF Core beide entiteiten (records) toe aan de database. In een eerste voorbeeld maken we een nieuwe campus en een nieuwe docent. We associëren de docent met de campus vanuit het standpunt van de campus. Daarna bewaren we beide entiteiten in één beweging.

- Neem onderstaande code op in het hoofdprogramma :

```
var campus7 = new Campus
{
    Naam = "Campus07",
    Straat = "Straat07",
    Huisnummer = "7",
    Postcode = "7777",
    Gemeente = "Gemeente07"
};
var docent2 = new Docent
{
    Voornaam = "Voornaam02",
    Familiennaam = "Docent02",
    Wedde = 2222
};
campus7.Docenten = new List<Docent>();
```

```

        campus7.Docenten.Add(docent2); //(1)
        using (var context = new EFOPLEIDINGENDbContext())
        {
            context.Campussen.Add(campus7);
            context.SaveChanges();
        }

```

- (1) De docent wordt geassocieerd met de campus door deze toe te voegen aan de verzameling docenten van die campus.

Run de app

Je kan nu controleren in SSMS of in de Visual Studio Server Explorer of de campussen goed is toegevoegd.

In een tweede voorbeeld voegen we opnieuw een campus en een docent toe. We associëren de docent met de campus vanuit het standpunt van de docent.

- Neem onderstaande code op in het hoofdprogramma :

```

var campus8 = new Campus
{
    Naam = "Campus08",
    Straat = "Straat08",
    Huisnummer = "8",
    Postcode = "8888",
    Gemeente = "Gemeente08"
};
var docent3 = new Docent
{
    Voornaam = "Voornaam03",
    Familienaam = "Docent03",
    Wedde = 3333
};
docent3.Campus = campus8; //(1)
using (var context = new EFOPLEIDINGENDbContext())
{
    context.Docenten.Add(docent3);
    context.SaveChanges();
}

```

- (1) We associëren de docent met de campus door de property Campus van de docent in te vullen.

- Probeer deze code uit.

7.4 Entiteiten én bijhorende bestaande geassocieerde entiteiten toevoegen

We kunnen ook nieuwe entiteiten toevoegen met een associatie naar een bestaande entiteit. Als voorbeeld voegen we een nieuwe docent toe die tot een bestaande campus moet behoren. Dit kan op twee manieren :

1. Je leest de geassocieerde entity (de bestaande campus) en associeert die aan de nieuwe entity via de Campus-property.

2. Je associeert de bestaande campus aan de docent via de foreign-key property CampusId. In onderstaande code proberen we beide uit.

- Neem onderstaande code op toe in het hoofdprogramma :

```
var docent4 = new Docent
{
    Voornaam = "Voornaam04",
    Familiennaam = "Docent04",
    Wedde = 4444
};
using (var context = new EF0pleidingenDbContext())
{
    var campus1 = context.Campussen.Find(1); //(1)
    if (campus1 != null)
    {
        context.Docenten.Add(docent4); //(2)
        docent4.Campus = campus1; //(3)
        context.SaveChanges(); //(4)
    }
    else
        Console.WriteLine("Campus 1 niet gevonden");
}
var docent5 = new Docent
{
    Voornaam = "Voornaam05",
    Familiennaam = "Docent05",
    Wedde = 5555,
    CampusId = 1 //(5)
};
using (var context = new EF0pleidingenDbContext())
{
    context.Docenten.Add(docent5); //(6)
    context.SaveChanges(); //(7)
}
```

(1) Je leest de bestaande campus die aan de nieuwe docent moet worden gekoppeld.

(2) Als die campus bestaat voeg je de docent toe.

(3) Je zet de Campus-property van de docent op de gevonden campus.

(4) Je bewaart de wijzigingen.

(5) Je zet de CampusId-property van de docent op de id van de gevonden campus.

(6) Je voegt de docent toe.

(7) Je bewaart de wijzigingen.

- Probeer uit.

In de bovenstaande voorbeelden hebben we de associatie telkens ingesteld vanuit de veel-kant van de relatie : de docenten. We kunnen de associatie ook instellen vanuit de één-kant van de relatie.

- Neem onderstaande code over in het hoofdprogramma :

```
var docent = new Docent
{
    Voornaam = "Voornaam06",
    Familiennaam = "Docent06",
    Wedde = 6666
};
using (var context = new EF0pleidingenDbContext())
{
    var campus = context.Campussen.Find(1); //(1)
    if (campus != null)
    {
        campus.Docenten.Add(docent); //(2)
        context.SaveChanges();
    }
    else
        Console.WriteLine("Campus 1 niet gevonden");
}
```

(1) We zoeken de campus met campusid = 1

(2) Als we die vinden dan voegen we die toe aan de docentenlijst van deze campus

8 Entity's wijzigen

Je kan de gegevens, opgeslagen in één of meerdere entity's wijzigen. Ook de entity's die je indirect inleest omdat ze gerelateerd zijn aan een andere entity kan je aanpassen. In dit hoofdstuk bekijken we tenslotte ook hoe je een associatie verandert.

8.1 Eén entity wijzigen

Om een entity te wijzigen volstaat het een property van de ingelezen entity te veranderen en vervolgens de method `SaveChanges` op te roepen op de context. EF Core stuurt dan een update SQLstatement naar de database. Een voorbeeld : we geven één docent opslag.

- Neem onderstaande code over in het hoofdprogramma :

```
Console.Write("DocentNr.:");
if (int.TryParse(Console.ReadLine(), out int docentNr))
{
    using (var context = new EFOpleidingenContext())
    {
        var docent = context.Docenten.Find(docentNr);
        if (docent != null)
        {
            Console.WriteLine("Wedde:{0}", docent.Wedde);
            Console.Write("Bedrag:");
            if (decimal.TryParse(Console.ReadLine(), out decimal bedrag))
            {
                docent.Opslag(bedrag);
                context.SaveChanges();
            }
            else
                Console.WriteLine("Tik een getal");
        }
        else
            Console.WriteLine("Docent niet gevonden");
    }
}
else
    Console.WriteLine("Tik een getal");
```

- In de class `Docent.cs` voeg je een method `Opslag()` toe :

```
public partial class Docent
{
    ...
    public void Opslag(decimal bedrag)
    {
        Wedde += bedrag;
    }
}
```

- Je kan het programma uitproberen. Controleer daarna de waarde van de wedde van de betrokken docent.

8.2 Meerdere entity's lezen en slechts enkele daarvan wijzigen

Meestal zal je in jouw programma meerdere entiteiten inlezen en er slechts enkele daarvan wijzigen. Een oproep van de method `SaveChanges` op de context zorgt ervoor dat voor alle aangepaste entiteiten een update-statement naar de database wordt gestuurd.

Een voorbeeld : je geeft alle docenten met een wedde tot een bepaalde bovengrens een opslag van €100.

- Neem onderstaande code over in het hoofdprogramma :

```
Console.Write("Bovengrens : ");
if (int.TryParse(Console.ReadLine(), out int grens))
{
    using (var context = new EF0pleidingenDbContext())
    {
        foreach (var docent in context.Docenten) //(1)
        if (docent.Wedde <= grens) docent.Opslag(100m); //(2)
        context.SaveChanges(); //(3)
    }
}
else
    Console.WriteLine("Tik een getal");
```

(1) We lezen alle docenten in.

(2) De docenten met een wedde tot aan de bovengrens krijgen een opslag.

(3) Een oproep van de `SaveChanges` method stuurt de nodige update-statements naar de database.

- Probeer uit en controleer de nieuwe weddes.

Opmerking : het bovenstaande programma kan uiteraard ook efficiënter geschreven worden door bij de selectie van de docenten reeds de voorwaarde op te nemen via een `where`-clause. We deden dit niet zodat we over een set entiteiten zouden beschikken waarvan er slechts enkele gewijzigd waren.

8.3 Entity's wijzigen die indirect zijn ingelezen via associaties

Soms lees je een entity in en lees je via de navigation property één of een verzameling geassocieerde entity's in.

Voorbeeld : je leest een Campus-entity. Via de `Docenten`-property kunnen één of meerdere geassocieerde docenten-entity's worden gelezen. Wanneer we de gegevens van deze docenten wijzigen zullen er na de oproep van de `SaveChanges`-method ook voor deze entity's update statements naar de database worden gestuurd.

- Neem onderstaande code over in het hoofdprogramma :

```
using (var context = new EF0pleidingenDbContext())
{
    var campus1 = context.Campussen.Include("Docenten")
        .FirstOrDefault(c => c.CampusId == 1);
    if (campus1 != null)
    {
        foreach (var docent in campus1.Docenten)
        {
            docent.Opslag(10M);
            context.SaveChanges();
        }
    }
}
```

- Probeer uit en controleer de nieuwe weddes.

8.4 Een associatie van een entity wijzigen

Ook een associatie van een entiteit kan wijzigen. Een docent kan bijvoorbeeld veranderen van campus. We kunnen dit doen vanuit de veel-kant van de relatie of vanuit de één-kant.

8.4.1 De associatie wijzigen vanuit de veel-kant

Ja kan een associatie op twee manieren wijzigen vanuit de veel-kant :

1. De te associëren entity inlezen en associëren aan de te wijzigen entity. In het voorbeeld betekent dit dat de nieuwe campus wordt gelezen en dat de docent aan deze campus wordt geassocieerd.
2. De te associëren entity associëren via de foreign key property. In het voorbeeld betekent dit dat de campusid-property in de docent entity wordt gewijzigd.

We proberen beide methodes nu uit.

Als eerste voorbeeld verhuizen we docent 1 naar campus 6.

- Neem onderstaande code over in het hoofdprogramma :

```
using (var context = new EF0pleidingenDbContext())
{
    var docent1 = context.Docenten.Find(1); //(1)
    if (docent1 != null)
    {
        var campus6 = context.Campussen.Find(6); //(2)
        if (campus6 != null)
        {
            docent1.Campus = campus6; //(3)
            context.SaveChanges();
        }
        else
            Console.WriteLine("Campus 6 niet gevonden");
    }
    else
        Console.WriteLine("Docent 1 niet gevonden");
}
```

(1) De te wijzigen docent wordt opgehaald.

(2) Deze docent moet gekoppeld worden aan campus 6. We halen deze campus op.

(3) We associëren de docent met de nieuwe campus via de property Campus.

- Probeer de code uit. We kunnen de associatie ook via de foreign key wijzigen. Op deze manier moet de te associëren entity niet ingelezen worden wat performantiewinst betekent. In onderstaand voorbeeld verhuizen we docent 1 naar campus 2.

- Neem onderstaande code over in het hoofdprogramma.

```
using (var context = new EF0pleidingenDbContext())
{
    var docent1 = context.Docenten.Find(1);
    if (docent1 != null)
    {
        docent1.CampusId = 2; //(1)
        context.SaveChanges();
    }
    else
        Console.WriteLine("Docent 1 niet gevonden");
}
```

(1) We veranderen de associatie door de campusid een andere waarde te geven.

- Probeer de code uit

8.4.2 De associatie wijzigen vanuit de één-kant

Je kan een associatie ook wijzigen vanuit de één-kant van de relatie. In ons voorbeeld waarbij we een docent van campus laten verhuizen betekent dit dat we dit zullen doen vanuit de kant van de campus. We doen dit door de docent toe te voegen aan de docentenlijst van de nieuwe campus. Je hoeft de docent niet te verwijderen uit de docentenlijst van de oude campus.

We proberen dit uit met een voorbeeld : we verhuizen docent 1 naar campus 3.

- Neem onderstaande code over in het hoofdprogramma en probeer daarna uit.

```
using (var context = new EF0pleidingenDbContext())
{
    var docent1 = context.Docenten.Find(1); //(1)
    if (docent1 != null)
    {
        var campus3 = context.Campussen.Find(3); //(2)
        if (campus3 != null)
        {
            campus3.Docenten.Add(docent1); //(3)
            context.SaveChanges();
        }
        else
            Console.WriteLine("Campus 3 niet gevonden");
    }
    else
        Console.WriteLine("Docent 1 niet gevonden");
}
```

(1) De docent wordt ingelezen.

(2) De campus wordt ingelezen.

(3) De docent wordt toegevoegd aan de docentenlijst van de campus.

9 Entity's verwijderen

Entiteiten moeten uiteraard ook verwijderd kunnen worden. We doen dit door eerst de entiteit in te lezen. Vervolgens geven we deze entiteit mee als parameter aan de method `Remove` van de bijhorende dbset uit de contextclass. Vervolgens wordt op diezelfde contextclass de `SaveChanges`method opgeroepen.

9.1 Voorbeeld

Een voorbeeld : we laten de gebruiker een docentid intikken en verwijderen vervolgens de bijhorende docent.

- Neem onderstaande code over in het hoofdprogramma en probeer daarna uit :

```
Console.Write("Nummer docent:");
if (int.TryParse(Console.ReadLine(), out int docentNr))
{
    using (var context = new EF0pleidingenDbContext())
    {
        var docent = context.Docenten.Find(docentNr); //(1)
        if (docent != null)
        {
            context.Docenten.Remove(docent); //(2)
            context.SaveChanges();
        }
        else
            Console.WriteLine("Docent niet gevonden");
    }
}
else
    Console.WriteLine("Tik een getal");
```

(1) Je leest de docent.

(2) Je verwijdert de docent uit de dbset `Docenten` uit de contextclass.

10 Transacties

Bij een database transactie worden meerdere SQL-statements als één geheel aanzien. Het is de verantwoordelijkheid van de database ervoor te zorgen dat...

- ...ofwel de volledige transactie lukt, wat wil zeggen dat alle SQL-statements binnen de transactie uitgevoerd zijn. Dit noemt men een commit van de transactie.
- ...ofwel de volledige transactie mislukt. Bijvoorbeeld bij een fout in de database, een fout in jouw applicatie, stroomuitval,... Dit wil zeggen dat alle SQL-statements binnen de transactie ongedaan gemaakt worden. Dit noemt men een rollback van de transactie.

Een voorbeeld van een transactie is het overschrijven van geld van een spaarrekening naar een zichtrekening bij dezelfde bank.

Hiervoor zijn twee update-statements nodig:

1. Een statement dat het te transfereren geld aftrekt van het saldo van de spaarrekening
2. Een statement dat het te transfereren geld bijtelt bij het saldo van de zichtrekening

10.1 Kenmerken van transacties

Transacties hebben 4 kenmerken ook wel gekend als de ACID kenmerken :

Atomicity

De SQL-statements die tot de transactie behoren vormen één geheel. Een database voert een transactie helemaal of niet uit. Als er halverwege de transactie een fout gebeurt, dan brengt de database de bijgewerkte records terug in hun oorspronkelijke toestand.

Consistency

De transactie breekt geen databaseregels. Als een kolom bijvoorbeeld geen duplicaten mag bevatten, dan zal de database de transactie afbreken (rollback) op het moment dat je toch probeert het duplicaat toe te voegen.

Isolation

Gedurende een transactie zijn de bewerkingen van de transactie niet zichtbaar voor andere lopende transacties. Om dit te bereiken vergrendelt de database de bijgewerkte records tot het einde van de transactie.

Durability

Een voltooide transactie is definitief vastgelegd in de database, zelfs al valt de computer uit juist na het voltooien van de transactie.

10.2 Isolation level

Het isolation level van een transactie definieert hoe de transactie beïnvloed wordt door handelingen van andere gelijktijdige transacties. Als meerdere transacties op eenzelfde moment in uitvoering zijn, kunnen volgende problemen optreden:

- Dirty read

Dit gebeurt als een transactie data leest die een andere transactie geschreven heeft, maar nog niet gecommit heeft. Als die andere transactie een rollback doet, is de data gelezen door de eerste transactie verkeerd.

- Nonrepeatable read

Dit gebeurt als een transactie meerdere keren dezelfde data leest en per leesopdracht deze data wijzigt. De oorzaak zijn andere transacties die tussen de leesoperaties van de eerste transactie dezelfde data wijzigen. De eerste transactie krijgt geen stabiel beeld van de gelezen data.

- Phantom read

Dit gebeurt als een transactie meerdere keren dezelfde data leest en per leesoperatie meer records leest. De oorzaak zijn andere transacties die records toevoegen tussen de leesoperaties van de eerste transactie. De eerste transactie krijgt geen stabiel beeld van de gelezen data. Je verhindert één of meerdere van deze problemen door het isolation level van de transactie in te stellen.

Er zijn 4 isolation levels : read uncommitted, read committed, repeatable read en serializable. Deze isolation levels hebben volgende kenmerken :

↓ Isolation level ↓	Dirty read kan optreden	Nonrepeatable read kan optreden	Phantom read kan optreden
Read uncommitted	Ja	Ja	Ja
Read committed	Nee	Ja	Ja
Repeatable read	Nee	Nee	Ja
Serializable	Nee	Nee	Nee

Het lijkt aanlokkelijk altijd het isolation level Serializable te gebruiken, want deze keuze lost alle problemen op. Het is echter zo dat Serializable het traagste isolation level is. De isolation levels van snel naar traag:

(SNEL) Read uncommitted -> Read committed -> Repeatable read -> Serializable (TRAAG)

10.3 De method SaveChanges

Je leerde al de method SaveChanges() van de DbContext kennen. We herhalen nog eens wat deze method doet:

- Het stuurt voor iedere entity die je aan de DbContext hebt toegevoegd een insert-SQLstatement naar de database.
- Het stuurt voor iedere entity die je gelezen én gewijzigd hebt een update-SQL-statement naar de database.
- Het stuurt voor iedere entity die je verwijderd hebt ten opzichte van de object context een delete-SQL-statement naar de database.

De method SaveChanges() verzamelt al deze bewerkingen zelf in één transactie. Jij hoeft dus in veel gevallen geen transactiebeheer te doen. De method SaveChanges() gebruikt read committed als transaction isolation level

10.4 Eigen transactiebeheer met TransactionScope

In sommige gevallen kan je de ingebouwde transacties van de method SaveChanges() niet gebruiken:

- Je wil een ander isolation level dan read committed gebruiken.
- Je wil een distributed transaction doen. Bij een dergelijke transaction bevinden de records die tot de transaction behoren zich niet in één maar in meerdere databases.

Je kan je eigen transactiebeheer doen met de class TransactionScope.

Je maakt een TransactionScope-object binnen een using-structuur. Alle databasebewerkingen die je binnen deze using-structuur uitvoert, behoren automatisch tot één en dezelfde transactie.

```
using (var transactionScope = new TransactionScope())
{
    ...
}
```

Wanneer alle bewerkingen goed aflopen voer je op het transactionscope object de method Complete uit. Bij het uitvoeren van deze method gebeurt een commit van alle bewerkingen van alle databaseconnecties die je uitgevoerd hebt binnen de using-structuur.

Als je de using-structuur verlaat zonder de method Complete uit te voeren (bijvoorbeeld omdat er een exception optreedt), gebeurt er automatisch een rollback van alle bewerkingen van alle databaseconnecties die je opende binnen de using-structuur.

De method SaveChanges detecteert automatisch een lopende transactie die je met TransactionScope gestart hebt. De method SaveChanges start dan geen eigen transactie, maar doet zijn bewerkingen binnen jouw transactie.

Je beëindigt de transactie met een oproep van de TransactionScope-method Complete().

Je kan transactionscoptes nesten. Je kan dus binnen een transaction een soort sub-transaction opstarten. Dit ziet er in code als volgt uit :

```

using (var transactionScope = new TransactionScope())
{
    using (var transactionScope2 = new TransactionScope())
    {
        ...
    }
}

```

Hoe de binnenste transaction zich gedraagt ten opzichte van de buitenste transaction kan je instellen door de TransactionScope via een constructor een scopeoption mee te geven. Deze scopeoption is een enum die volgende waarden kan hebben :

- TransactionScopeOption.**Required** Het TransactionScope object start een transactie als het object niet in een ander TransactionScope object genest is. Als er wel genest is, gebruikt het object de transactie die al door het omringende TransactionScope-object gestart werd.
- TransactionScopeOption.**RequiresNew** Het TransactionScope object start een nieuwe transactie, zelfs als het object genest is in een ander TransactionScope-object.
- TransactionScopeOption.**Suppress**

Alle databasebewerkingen binnen dit TransactionScope-object behoren niet tot een transactie, zelfs als het TransactionScope-object genest is in een ander TransactionScopeobject.

Ook het isolationlevel kan je aangeven via de TransactionScope constructor. Je geeft daartoe een parameter mee van het type TransactionOption. Dit object heeft een property IsolationLevel.

10.5 Voorbeeld

We proberen transactions uit in een voorbeeld. In een tabel `CursusVoorraden` worden de voorraden van cursussen in magazijnen bijgehouden. De tabel heeft volgende structuur :

DESKTOP-BQCVL2L\....CursusVoorraden			
	Column Name	Data Type	Allow Nulls
🔑	MagazijnNr	int	<input type="checkbox"/>
🔑	CursusNr	int	<input type="checkbox"/>
	AantalStuks	int	<input type="checkbox"/>
	RekNr	int	<input type="checkbox"/>

In een magazijn kan er een bepaald aantal stuks van een cursus aanwezig zijn. De kolom `RekNr` geeft aan in welk rek deze cursussen liggen. We zullen cursussen van het éne magazijn transfereren naar een ander magazijn. We maken deze tabel nu aan.

- Voeg in het project Model in de map Entities een class `Voorraad` toe :

```
[Table("CursusVoorraden")]
public class Voorraad
{
    public int MagazijnNr { get; set; }
    public int CursusNr { get; set; }
    [Required]
    public int AantalStuks { get; set; }
    [Required]
    public int RekNr { get; set; }
}
```

De tabel `Voorraad` heeft een samengestelde sleutel (`MagazijnNr+CursusNr`). We stellen die in via de method `OnModelCreating()`. In dezelfde method voegen we ook via seeding wat data toe. Bovenaan de class `EFOpleidingenDbContext` voeg je tenslotte een extra `DbSet` toe voor de tabel `CursusVoorraden`.

- Voeg in `EFOpleidingenContext.cs` bovenaan een extra `DbSet` toe :

```
public class EFOpleidingenContext : DbContext
{
    public DbSet<Campus> Campussen { get; set; }
    public DbSet<Docent> Docenten { get; set; }
    public DbSet<Land> Landen { get; set; }
    public DbSet<Voorraad> Voorraden { get; set; }
    ...
}
```

- In de method `OnModelCreating()` definieer je de samengestelde sleutel en voeg je seeding data toe:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Voorraad>().HasKey(table => new {
        table.MagazijnNr,
        table.CursusNr
    });
    if (!testMode)
    {
        modelBuilder.Entity<Voorraad>().HasData
        (
            new Voorraad { MagazijnNr = 1, CursusNr = 10,
                AantalStuks = 100, RekNr = 3 },
            new Voorraad { MagazijnNr = 2, CursusNr = 10,
                AantalStuks = 1000, RekNr = 17 },
            new Voorraad { MagazijnNr = 1, CursusNr = 20,
                AantalStuks = 200, RekNr = 12 },
            new Voorraad { MagazijnNr = 2, CursusNr = 20,
                AantalStuks = 2000, RekNr = 23 },
            new Voorraad { MagazijnNr = 1, CursusNr = 30,
                AantalStuks = 300, RekNr = 4 },
            new Voorraad { MagazijnNr = 2, CursusNr = 30,
                AantalStuks = 3000, RekNr = 9 }
        );
    }
    ...
}
```

We maken nu een nieuwe migration aan.

- Start de package manager console en ga met de instructie `cd model` naar de map `Model`.
- Tik het commando `dotnet ef migrations add metvoorraden` om de migration aan te maken.
- Je voert de migration door via het commando `dotnet ef database update`

We proberen nu op twee verschillende manieren een voorraadtransfer te doen : één keer met het ingebouwde transactionbeheer via `SaveChanges()` en één keer met eigen transactionbeheer.

10.5.1 Ingebouwd transactiebeheer

We voegen eerst in `Program.cs` een method `VoorraadTransfer()` toe die een voorraad cursussen transfereert van het éne magazijn naar het andere.

- Voeg onderstaande method toe in `Program.cs` :

```
static void VoorraadTransfer(int cursusNr, int vanMagazijnNr,
    int naarMagazijnNr, int aantalStuks)
{
    using (var context = new EF0pleidingenDbContext())
    {
        var vanVoorraad = context.Voorraden.Find(vanMagazijnNr, cursusNr); //
(1)
        if (vanVoorraad != null) //(2)
        {
            if (vanVoorraad.AantalStuks >= aantalStuks) //(3)
            {
                vanVoorraad.AantalStuks -= aantalStuks; //(4)
            }
        }
    }
}
```



```

        var naarVoorraad =
context.Voorraden.Find(naarMagazijnNr, cursusNr); //(5)
        if (naarVoorraad != null) //(6)
            naarVoorraad.AantalStuks += aantalStuks; //(7)
        else //(8)
        {
            naarVoorraad = new Voorraad
            {
                CursusNr = cursusNr,
                MagazijnNr = naarMagazijnNr,
                AantalStuks = aantalStuks
            };
            context.Voorraden.Add(naarVoorraad); //(9)
        }
        context.SaveChanges(); //(10)
    }
    else
        Console.WriteLine("Te weinig voorraad voor transfer");
    }
    else
        Console.WriteLine("Artikel niet gevonden in magazijn {0}",
            vanMagazijnNr);
    }
}

```

- (1) We zoeken het voorraad-record op voor een bepaalde cursus in het magazijn van waaruit de transfer vertrekt.
- (2) Als we een dergelijk record vinden...
- (3) ...en de aanwezige voorraad is minstens zo groot als het aantal te transfereren cursussen...
- (4) ...dan verminderen we de voorraad in het vertrekmagazijn.
- (5) We zoeken het voorraad-record op voor een bepaalde cursus in het magazijn waar de transfer naartoe moet.
- (6) Als we dat record vinden...
- (7) ...dan passen we de voorraad aan.
- (8) Vinden we geen dergelijke record dan zijn er nog geen dergelijke cursussen in het bestemmingsmagazijn en moeten we een nieuw voorraadrecord aanmaken...
- (9) ...en toevoegen aan de database.
- (10) We eindigen de transactie met een SaveChanges-oproep. We gebruiken deze method nu in het hoofdprogramma.

- Voeg onderstaande code toe in het hoofdprogramma :

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("Cursusnr.");
        var cursusNr = int.Parse(Console.ReadLine());
        Console.WriteLine("Van magazijn nr.");
        var vanMagazijnNr = int.Parse(Console.ReadLine());
        Console.WriteLine("Naar magazijn nr.");
        var naarMagazijnNr = int.Parse(Console.ReadLine());
        Console.WriteLine("Aantal stuks.");
        var aantalStuks = int.Parse(Console.ReadLine());
        VoorraadTransfer(cursusNr, vanMagazijnNr,
            naarMagazijnNr, aantalStuks);
    }
    catch (FormatException)
    {
        Console.WriteLine("Tik een getal");
    }
}
```

- Controleer de voorraad van een cursus en probeer daarna een transfer uit, bijvoorbeeld 5 exemplaren van cursus 10 van magazijn 1 naar magazijn 2.

10.6 Eigen transactiebeheer

Via de logging-informatie kon je in het bovenstaande voorbeeld zien dat EF Core in totaal 4 SQLstatements uitvoerde. Het programma doet de voorraadaanpassing correct op voorwaarde dat geen andere gebruikers tegelijk aanpassingen doen. Gedurende jouw transactie worden er immers geen records gelockt. In een tweede versie van dit programma beheer je de transactie zelf zodat deze problemen zich niet meer kunnen voordoen.

Je plaatst het isolation level van de transactie op **repeatable read**. Dan **lockt** de database de records die je leest tot het einde van de transactie. Zo kunnen andere gebruikers tussendoor de zelfde records niet wijzigen.

- Pas de method VoorraadTransfer() als volgt aan :

```
static void VoorraadTransfer(int cursusNr, int vanMagazijnNr,
    int naarMagazijnNr, int aantalStuks)
{
    var transactionOptions = new TransactionOptions//(1)
    {
        IsolationLevel = IsolationLevel.RepeatableRead
    };
    using (var transactionScope = new TransactionScope(
        TransactionScopeOption.Required, transactionOptions)) //(2)
    {
        using (var context = new EFopleidingenDbContext())
        {
            var vanVoorraad = context.Voorraden.Find(vanMagazijnNr, cursusNr);// (1)
            if (vanVoorraad != null) //(2)
            {
                if (vanVoorraad.AantalStuks >= aantalStuks) //(3)
                {
                    vanVoorraad.AantalStuks -= aantalStuks; //(4)
                    var naarVoorraad =
                        context.Voorraden.Find(naarMagazijnNr, cursusNr); //(5)
                    if (naarVoorraad != null) //(6)
                        naarVoorraad.AantalStuks += aantalStuks; //(7)
                }
            }
        }
    }
}
```

```

else //(8)
{
    naarVoorraad = new Voorraad
    {
        CursusNr = cursusNr,
        MagazijnNr = naarMagazijnNr,
        AantalStuks = aantalStuks
    };
    context.Voorraden.Add(naarVoorraad); //(9)
}
context.SaveChanges();
transactionScope.Complete(); //(3)
}
}
}
}
}
}
}

```

(1) We definiëren een TransactionOptions-object waarvan we het **isolationlevel** op **RepeatableRead** zetten.

(2) We starten een transactionscope met de gedefinieerde options.

(3) Aan het einde van de transaction roepen we de method **Complete()** op. Zonder deze oproep zou de transaction gerollbackt worden.

- Probeer uit.

11 Optimistic record locking

In het voorbije hoofdstuk heb je gezien dat transactions een oplossing kunnen zijn voor allerlei problemen die zich kunnen voordoen wanneer meerdere gebruikers dezelfde records lezen en aanpassen.

Transactions zorgen echter voor een zekere overhead en vertragen de uitvoering van het programma omdat telkens records gelockt moeten worden. In dit hoofdstuk stellen we met optimistic record locking een alternatief voor.

Bij optimistic record locking wordt het gelezen record niet echt gelockt. Bij het aanpassen van het record controleert EF Core of de waarden in het record nog dezelfde zijn als op het moment dat het record gelezen werd. Als dit niet het geval is (omdat ondertussen een andere applicatie dit record wijzigde), werpt EF Core een exception.

De manier waarop je optimistic record locking activeert verschilt een beetje, naargelang de table al of niet een timestamp-kolom heeft. Een timestamp-kolom is een kolom in de database waarin de database zelf bij iedere recordwijziging een andere waarde invult. In dit hoofdstuk proberen we optimistic record locking uit mét en zonder timestamp-veld.

11.1 Probleemstelling

In onderstaand voorbeeld vullen we de cursusvoorraad aan. Tussen het inlezen van de huidige voorraad en het wegschrijven van de nieuwe voorraad kan een andere gebruiker de voorraad ook veranderen. Dit leidt tot een probleem.

We proberen dit uit.

- Voeg onderstaande method toe in Program.cs :

```
static void VoorraadBijvulling(int cursusNr, int magazijnNr, int aantalStuks)
{
    using (var context = new EF0pleidingenDbContext())
    {
        var voorraad = context.Voorraden.Find(magazijnNr, cursusNr); //(1)
        if (voorraad != null)
        {
            voorraad.AantalStuks += aantalStuks; //(2)
            Console.WriteLine("Pas nu de voorraad aan met de Server Explorer,"
                + " druk daarna op Enter");
            Console.ReadLine(); //(3)
            context.SaveChanges(); //(4)
        }
        else
            Console.WriteLine("Voorraad niet gevonden");
    }
}
```

(1) Je zoekt de voorraad van een cursus in een magazijn.

(2) Je verhoogt de voorraad met een aantal stuks.

(3) We laten hier even een pauze zodat de voorraad elders kan gewijzigd worden.

(4) We slaan alle wijzigingen op.

- Neem nu onderstaande code over in het hoofdprogramma :

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("Cursusnr.");
        var cursusNr = int.Parse(Console.ReadLine());
        Console.WriteLine("Magazijn nr.");
        var magazijnNr = int.Parse(Console.ReadLine());
        Console.WriteLine("Aantal stuks toevoegen.");
        var aantalStuks = int.Parse(Console.ReadLine());
        VoorraadBijvulling(cursusNr, magazijnNr, aantalStuks);
    }
    catch (FormatException)
    {
        Console.WriteLine("Tik een getal");
    }
}
```

- Bekijk de voorraad van cursus 10 in magazijn 1.
- Start het programma.
- Verhoog de voorraad van cursus 10 in magazijn 1 met 100.
- Op het moment dat je de boodschap 'Pas nu de voorraad...' krijgt schakel je over naar SSMS.
- Wijzig de voorraad van cursus 10 in magazijn 1 naar 666.
- Schakel terug over naar het programma en druk op Enter.
- Controleer de voorraad in SSMS. De wijziging via SSMS is terug ongedaan gemaakt.

11.2 Oplossing zonder timestampveld

Op dit moment heeft de table CursusVoorraden geen timestamp-kolom. In dit geval activeren we optimistic record locking op de volgende manier. Je wijzigt de class Voorraad door vóór alle properties die meegenomen moeten worden voor de locking de annotation [ConcurrencyCheck] te plaatsen.

- Wijzig de class Voorraad als volgt :

```
[Table("CursusVoorraden")]
public class Voorraad
{
    [ConcurrencyCheck]
    public int MagazijnNr { get; set; }
    [ConcurrencyCheck]
    public int CursusNr { get; set; }
    [Required]
    [ConcurrencyCheck]
    public int AantalStuks { get; set; }
    [Required]
    [ConcurrencyCheck]
    public int RekNr { get; set; }
}
```

Opmerking : als alternatief voor de annotations kan je ook volgende code toevoegen in de **OnModelCreating()** methode van de context class:

```
modelBuilder.Entity<Voorraad>().Property(a => a.MagazijnNr).IsConcurrencyToken();
modelBuilder.Entity<Voorraad>().Property(a => a.CursusNr).IsConcurrencyToken();
modelBuilder.Entity<Voorraad>().Property(a => a.AantalStuks).IsConcurrencyToken();
modelBuilder.Entity<Voorraad>().Property(a => a.RekNr).IsConcurrencyToken();
```

Bij het lezen van een record onthoudt EF Core nu de waarden van de kolommen waarbij de annotation **[ConcurrencyCheck]** staat vermeld.

Bij het wijzigen van het record stuurt EF Core volgend SQL-statement naar de database :

```
update Voorraden set AantalStuks = @0 where MagazijnNr=@1
and CursusNr=@2 and AantalStuks = @3 and RekNr = @4
```

In dit statement krijgt parameter @0 de waarde van het nieuwe aantal stuks. De parameters @1, @2, @3 en @4 krijgen de originele waarden van de kolommen.

Er zal dus pas een wijziging doorgevoerd worden als de waarden van de kolommen bij het updaten nog steeds dezelfde zijn als bij het inlezen van het record.

Als dat niet het geval is – iemand anders heeft de waarden intussentijd veranderd – dan wordt er een **DbUpdateConcurrencyException** geworpen.

We passen de code nu aan.

- Wijzig de method **VoorraadBijvulling** als volgt :

```
static void VoorraadBijvulling(int cursusNr, int magazijnNr, int aantalStuks)
{
    using (var context = new EFopleidingenDbContext())
    {
        var voorraad = context.Voorraden.Find(magazijnNr, cursusNr);
        if (voorraad != null)
        {
            voorraad.AantalStuks += aantalStuks;
            Console.WriteLine("Pas nu de voorraad aan met de Server Explorer,"
                + " druk daarna op Enter");
            Console.ReadLine();
            try
            {
                context.SaveChanges();
            }
            catch (DbUpdateConcurrencyException)
            {
                Console.WriteLine("Voorraad werd intussen door een andere"
                    + " applicatie aangepast.");
            }
        }
        else
            Console.WriteLine("Voorraad niet gevonden");
    }
}
```

We proberen opnieuw uit.

- Bekijk de voorraad van cursus 10 in magazijn 1.
- Start het programma.
- Verhoog de voorraad van cursus 10 in magazijn 1 met 100
- Op het moment dat je de boodschap 'Pas nu de voorraad...' krijgt schakel je over naar SSMS.
- Wijzig de voorraad van cursus 10 in magazijn 1 naar 666.
- Schakel terug over naar het programma en druk op Enter.

Deze keer krijg je een foutmelding. De voorraad wordt niet gewijzigd maar blijft op de waarde 666 staan die je manueel hebt ingegeven.

11.3 Oplossing met timestampveld

Als een table geen timestamp-kolom heeft wordt het where-deel van het update-SQL-statement bij optimistic record locking langer naargelang de table meer kolommen bevat. Het uitvoeren van een update-SQL-statement met een lang where-deel is nadelig voor de performantie.

Een oplossing is het toevoegen van een timestamp-kolom. Bij iedere wijziging van een record plaatst de database zelf een andere waarde in deze timestamp-kolom.

Je moet dan in het where-deel van het update-statement niet meer op iedere kolomwaarde controleren of het door een andere gebruiker werd bijgewerkt, maar enkel op deze **timestampkolom**.

We proberen dit uit.

- Verwijder in de class Voorraad alle **[ConcurrencyCheck]** annotations.
- Je voegt nu aan de class Voorraad een timestamp-kolom toe :

```
[Table("CursusVorraden")]
public class Voorraad
{
    public int MagazijnNr { get; set; }
    public int CursusNr { get; set; }
    [Required]
    public int AantalStuks { get; set; }
    [Required]
    public int RekNr { get; set; }
    [Timestamp]
    public byte[] Aangepast { get; set; }
}
```

(1)

(1) Je voegt een extra veld Aangepast toe van het type byte[]. Dit veld krijgt de annotation **[Timestamp]**.

Opmerking : als alternatief voor de annotation kan je ook volgende code toevoegen in de **OnModelCreating()** methode van de dbcontext class:

```
modelBuilder.Entity<Voorraad>().Property(a => a.Aangepast).IsRowVersion();
```

We maken nu een nieuwe migration aan om de extra kolom toe te voegen.

- Start de package manager console en ga met de instructie `cd model` naar de map Model.
- Tik het commando `dotnet ef migrations add mettimestamp` om de migration aan te maken.
- Je voert de migration door via het commando `dotnet ef database update`
- Controleer de tabelstructuur.

De tabelstructuur ziet er nu zo uit :

	Column Name	Data Type	Allow Nulls
🔑	MagazijnNr	int	<input type="checkbox"/>
🔑	CursusNr	int	<input type="checkbox"/>
	AantalStuks	int	<input type="checkbox"/>
	RekNr	int	<input type="checkbox"/>
	Aangepast	timestamp	<input checked="" type="checkbox"/>

We proberen opnieuw uit.

- Bekijk de voorraad van cursus 10 in magazijn 1.
- Start het programma.
- Verhoog de voorraad van cursus 10 in magazijn 1 met 100.
- Op het moment dat je de boodschap 'Pas nu de voorraad...' krijgt schakel je over naar SSMS.
- Wijzig de voorraad van cursus 10 in magazijn 1 naar 999.
- Schakel terug over naar het programma en druk op Enter. Je krijgt opnieuw een foutmelding. De voorraad wordt niet gewijzigd maar blijft op de waarde 999 staan die je manueel hebt ingegeven.

12 De change tracker

Bij het oproepen van de `SaveChanges()` method moet EF Core weten welke objecten werden gewijzigd om zo de gepaste SQL Statements (Update, Insert, Delete) te kunnen genereren. Dit gebeurt door middel van de **ChangeTracker** property van de `DbContext` die de gemaakte wijzigingen in de entities opvolgt.

De `ChangeTracker` property bevat een collectie met de naam `Entries` die alle op te volgen entities bevat.

Het opvolgen van de entities gebeurt vanaf het ogenblik dat die opgeladen wordt in de context. Bij het oproepen van de `SaveChanges()` method zal EF Core de correcte SQL statements uitvoeren op de database. De `SaveChanges()` method roept intern de `DetectChanges()` method op die de gewijzigde gegevens identificeert.

12.1 Statussen van de ChangeTracker

De `ChangeTracker` werkt met statussen die de soort van wijziging identificeert, het kan volgende statussen (uit de enum `Microsoft.EntityFrameworkCore.EntityState`) bevatten :

Status	Gevolg van de actie...	SQL statement
Unchanged	Geen. De entity is ongewijzigd.	-
Added	<code>Add()</code> . De entity is toegevoegd aan de context.	Insert
Modified	Wijziging van de waarde van één of meerdere property's van de entity.	Update
Deleted	<code>Remove()</code> . De entity werd verwijderd uit de context.	Delete
Detached	Geen. De entity wordt niet opgevolgd.	-

12.2 Een voorbeeld

We bekijken in een voorbeeld de changetracker-statussen van enkele entities.

- Open de solution EFCore.
- Neem onderstaande code over in het hoofdprogramma en voer daarna uit :

```
static void Main(string[] args)
{
    using (var context = new EFOPLEIDINGENDbContext())
    {
        Console.WriteLine("-----\nWijzigingen\n-----");
        // UnChanged
        var land0 = context.Landen.First();
        Console.WriteLine("\n" + land0.LandCode + " - " + land0.Naam +
            " - " + context.Entry(land0).State + "\n"); //(1)
        // Added
        var land1 = new Land { LandCode = "AB", Naam = "abcdef" };
        context.Landen.Add(land1);
        Console.WriteLine("\n" + land1.LandCode + " - " + land1.Naam +
            " - " + context.Entry(land1).State + "\n"); //(2)
        // Modified
        var land2 = context.Landen.Where(c => c.LandCode == "FR")
            .FirstOrDefault();
        land2.Naam = "France";
        Console.WriteLine("\n" + land2.LandCode + " - " + land2.Naam +
            " - " + context.Entry(land2).State + "\n"); //(3)
        // Deleted
        var land3 = context.Landen.Where(c => c.LandCode == "LU")
            .FirstOrDefault();
        context.Landen.Remove(land3);
        Console.WriteLine("\n" + land3.LandCode + " - " + land3.Naam +
            " - " + context.Entry(land3).State + "\n"); //(4)
        // Detached - Disconnected data
        var land4 = new Land { LandCode = "XY", Naam = "xyz" };
        Console.WriteLine("\n" + land4.LandCode + " - " + land4.Naam +
            " - " + context.Entry(land4).State + "\n"); //(5)
        Console.WriteLine("-----\nNa wijzigingen\n-----");
        context.ChangeTracker.DetectChanges();
        Console.WriteLine("\nHasChanges: {0}\n",
            context.ChangeTracker.HasChanges()); //(6)
        foreach (var entry in context.ChangeTracker.Entries()) //(7)
        {
            Console.WriteLine("Entity: {0}, Status: {1}",
                entry.Entity.GetType().Name, entry.State); //(8)
            foreach (var x in entry.Properties) //(9)
            {
                Console.WriteLine(
                    $"Property '{x.Metadata.Name}' +
                    $" is {(x.IsModified ? "modified" : "not modified")} " +
                    $" - Current value: '{x.CurrentValue}' " +
                    $" - Original value: '{x.OriginalValue}'");
            }
            Console.WriteLine("\n");
        }
    }
}
```

In een eerste stuk van bovenstaand programma zorgen we ervoor dat vijf entiteiten een bepaalde status krijgen.

(1) land0 is een ingelezen entiteit dat we verder niet gewijzigd hebben (=> unchanged)

(2) land1 is een nieuw en toegevoegd land (=> added)

(3) land2 is een land met een nieuwe naam (=> modified)

(4) land3 is een land dat verwijderd werd (=> deleted)

(5) land4 is een land dat niet tot de context behoort (=> detached)

Na de wijzigingen gebruiken we enkele ChangeTracker-methods.

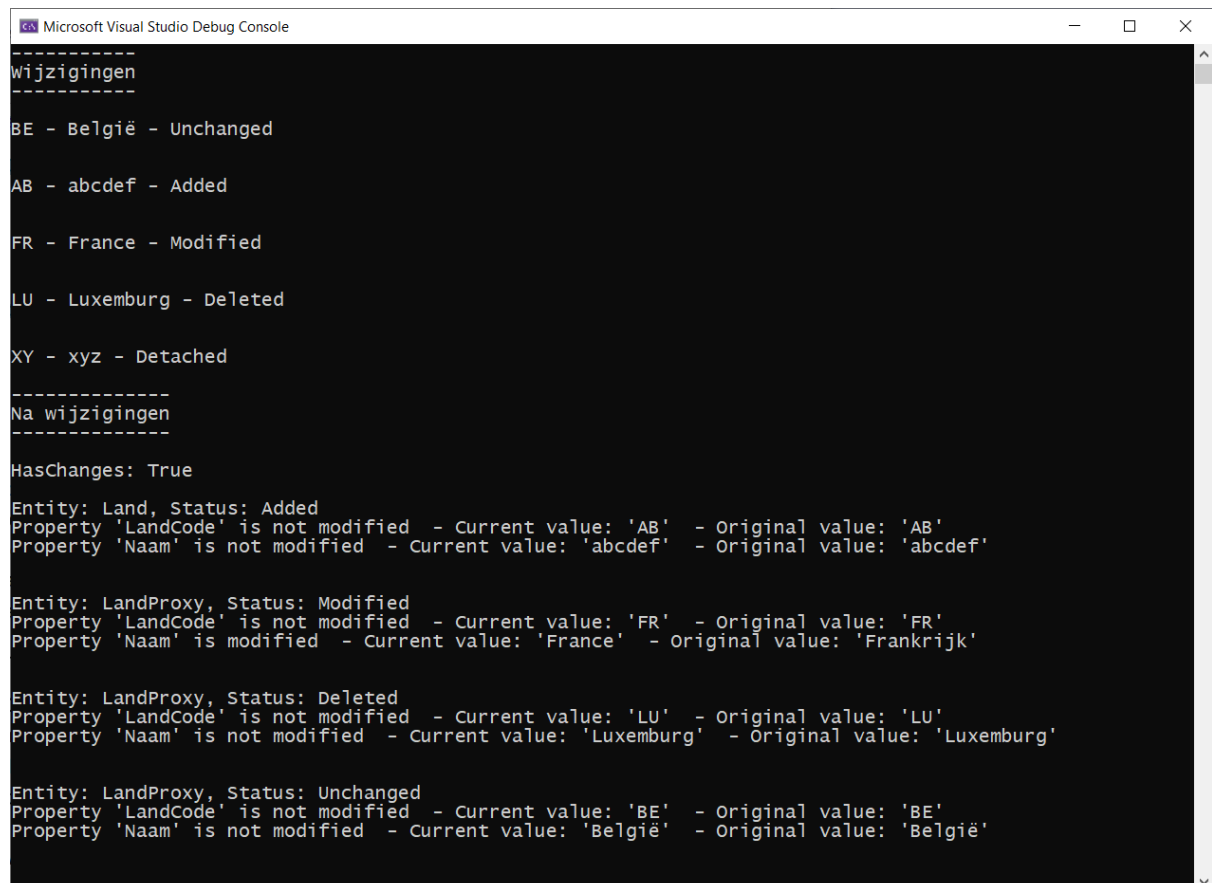
(6) De method DetectChanges controleert de entiteiten op eventuele veranderingen in status. De method HasChanges vertelt ons of er al dan niet veranderingen zijn.

(7) We overlopen een lijst entity-entry's, verkregen door een oproep van de method Entries(). Dit zijn allemaal entiteiten die door de ChangeTracker worden gevolgd. De entiteit land4 is daar dus niet bij.

(8) Per entiteit tonen we de status...

(9) ...en voor elke property van de entiteit of deze al dan niet is gewijzigd, de oude en de nieuwe waarde.

Merk op dat we de method SaveChanges() niet oproepen in het programma. De wijzigingen worden dus niet opgeslagen.



```
-----
Wijzigingen
-----
BE - België - Unchanged

AB - abcdef - Added

FR - France - Modified

LU - Luxemburg - Deleted

XY - xyz - Detached

-----
Na wijzigingen
-----
HasChanges: True

Entity: Land, Status: Added
Property 'LandCode' is not modified - Current value: 'AB' - Original value: 'AB'
Property 'Naam' is not modified - Current value: 'abcdef' - Original value: 'abcdef'

Entity: LandProxy, Status: Modified
Property 'LandCode' is not modified - Current value: 'FR' - Original value: 'FR'
Property 'Naam' is modified - Current value: 'France' - Original value: 'Frankrijk'

Entity: LandProxy, Status: Deleted
Property 'LandCode' is not modified - Current value: 'LU' - Original value: 'LU'
Property 'Naam' is not modified - Current value: 'Luxemburg' - Original value: 'Luxemburg'

Entity: LandProxy, Status: Unchanged
Property 'LandCode' is not modified - Current value: 'BE' - Original value: 'BE'
Property 'Naam' is not modified - Current value: 'België' - Original value: 'België'
```

12.3 Het StateChanged en Tracked event

De ChangeTracker beschikt over een aantal interessante events : StateChanged en Tracked. De event StateChanged wordt geraised wanneer de status van een entiteit verandert. Tracked treedt op wanneer een entiteit door de ChangeTracker gevolgd wordt.

In onderstaand programma koppelen we aan beide events een eventhandler zodat we goed kunnen zien wanneer welke event zich voordoet.

- Voeg aan EF0pleidingenContext.cs onderstaande vetgedrukte code toe :

```
public class EF0pleidingenContext : DbContext
{
    ...
    public EF0pleidingenContext()
    {
        ChangeTracker.StateChanged += StateChanged;           (1)
        ChangeTracker.Tracked += Tracked;                     (2)
    }

    public EF0pleidingenContext(DbContextOptions<EF0pleidingenContext> options) :
        base(options)
    {
        ChangeTracker.StateChanged += StateChanged;           (1)
        ChangeTracker.Tracked += Tracked;                     (2)
    }

    private void StateChanged(object sender, EntityStateChangedEventArgs e)
    {
        Console.WriteLine("Status is gewijzigd" +
            $" van {e.OldState} naar {e.NewState}");           (3)
    }

    private void Tracked(object sender, EntityTrackedEventArgs e)
    {
        Console.WriteLine("Nieuw te volgen entity");           (4)
    }
}
```

(1) In beide constructors koppelen we een eventhandler StateChanged aan de gelijknamige event.

(2) Ook aan de event Tracked koppelen we een eventhandler.

(3) In StateChanged() tonen we de oude en de nieuwe status.

(4) In Tracked() geven we aan dat een entiteit wordt gevolgd.

- Probeer uit.