



samen sterk voor werk

C# 2013

ADO.NET Entity Framework

Deze cursus is eigendom van de VDAB©

Inhoudsopgave

1	INLEIDING	1
1.1	Doelstelling	1
1.2	Vereiste voorkennis	1
1.3	Nodige software	1
2	DE OBJECT-RELATIONAL MISMATCH	2
2.1	Algemeen	2
2.2	Granularity	2
2.3	Inheritance	3
2.3.1	Table per concrete class	3
2.3.2	Table per class hierarchy	4
2.3.3	Table per subclass	5
2.4	Associaties	6
2.4.1	Één op veel Associaties	6
2.4.2	Veel op veel associaties	6
2.5	Navigeren door associaties	7
2.6	ORM (Object-relational mapping)	8
2.7	EDMX bestand versus Code First	8
3	DE DATABASE	9
4	HET ENTITY DATA MODEL (EDM)	10
4.1	Algemeen	10
4.2	Je eerste EDM	10
4.3	HET EDM als XML bestand	13
4.3.1	Algemeen	13
4.3.2	StorageModels	14
4.3.3	ConceptualModels	14
4.3.4	Mappings	15
4.4	De Model Browser	16
4.5	Extra technieken in de grafische designer	17
4.6	De DbContext class	17
4.7	Entity classes uitbreiden met properties en methods	18
5	QUERIES	19
5.1	Algemeen	19

5.1.1	Een foreach iteratie	19
5.1.2	Een LINQ query	19
5.1.3	Query methods	20
5.1.4	LINQ queries en queries met methods vergeleken	21
5.2	Een entity zoeken op zijn primary key waarde	22
5.3	Gedeeltelijke objecten ophalen	23
5.4	Groeperen in queries	23
5.5	Lazy Loading	24
5.6	Eager loading lost het performantieprobleem op	25
5.7	De method ToList van een query	26
6	ENTITIES TOEVOEGEN	28
6.1	Één entity toevoegen	28
6.2	Meerdere entities toevoegen	28
6.3	Entities met nieuwe geassocieerde entities toevoegen	28
6.4	Een entity toevoegen met een associatie naar een bestaande entity	29
6.4.1	Een entity toevoegen en de associatie definiëren vanuit de veel kant	29
6.4.2	Een entity toevoegen en de associatie definiëren vanuit de één kant	30
7	ENTITIES WIJZIGEN	31
7.1	Één entity wijzigen	31
7.2	Meerdere entities lezen en slechts enkele daarvan wijzigen	31
7.3	Entities wijzigen die je indirect gelezen hebt met associaties	32
7.4	Een associatie van een entity wijzigen	32
7.4.1	De associatie wijzigen vanuit de veel kant	32
7.4.2	De associatie wijzigen vanuit de één kant	33
8	ENTITIES VERWIJDEREN	34
9	TRANSACTIES	35
9.1	Algemeen	35
9.2	Isolation level	36
9.3	De method SaveChanges	36
9.4	Eigen transactiebeheer met TransactionScope	37
9.4.1	Algemeen	37
9.4.2	Voorbeeld	38
10	OPTIMISTIC RECORD LOCKING	41

10.1	Algemeen	41
10.2	Table zonder timestamp kolom	42
10.3	Table met timestamp kolom	43
11	ASSOCIATIES	45
11.1	Algemeen	45
11.2	Veel op veel associaties zonder extra associatieinformatie	45
11.2.1	De tables Boeken, Cursussen en BoekenCursussen toevoegen aan de database	45
11.2.2	De entities definiëren die bij de tables Boeken en Cursussen horen	45
11.2.3	De entities gebruiken vanuit je code	46
11.3	Veel op veel associaties met extra associatieinformatie	46
11.3.1	De tables Boeken2, Cursussen2 en BoekenCursussen2 toevoegen aan de database.	46
11.3.2	De entities definiëren die bij de tables Boeken2, Cursussen2 en BoekenCursussen2 horen	47
11.3.3	De entities gebruiken vanuit je code	48
11.4	Een associatie van een entity class naar zichzelf	49
11.4.1	Algemeen	49
11.4.2	De table Cursisten toevoegen aan de database.	49
11.4.3	De entity definiëren die bij de table Cursisten hoort	50
11.4.4	De entities gebruiken vanuit je code	50
12	INHERITANCE	52
12.1	Algemeen	52
12.2	Table per concrete class (TPC)	52
12.2.1	Algemeen	52
12.2.2	De tables KlassikaleCursussen en ZelfstudieCursussen toevoegen aan de database	52
12.2.3	De entities definiëren die horen bij de tables KlassikaleCursussen en ZelfstudieCursussen	53
12.2.4	De entity Cursus toevoegen en inheritance definiëren	53
12.2.5	De entities gebruiken vanuit je code	55
12.3	Table per hierarchy (TPH)	56
12.3.1	De entities definiëren die horen bij de Cursussen3	56
12.3.2	De entity KlassikaleCursus toevoegen en inheritance definiëren	56
12.3.3	De entity ZelfstudieCursus toevoegen en inheritance definiëren	57
12.3.4	De entities gebruiken vanuit je code	57
12.4	Table per type (TPT)	58
12.4.1	De tables Cursussen4, KlassikaleCursussen4 en ZelfstudieCursussen4 toevoegen aan de database	58
12.4.2	De entities definiëren die horen bij de tables Cursussen4, KlassikaleCursussen4 en ZelfstudieCursussen4	58
12.4.3	Inheritance definiëren	59
12.4.4	De entities gebruiken vanuit je code	59
13	COMPLEX TYPES	60
13.1	Algemeen	60
13.2	Een complex type maken op basis van een bestaande entity	61
13.3	Een complex type herbruiken in een andere entity	61
13.4	Complex type als partial class	62

13.5	Voorbeeldgebruik	62
14	ENUMS	63
14.1	Algemeen	63
14.2	Voordelen van een enum	63
14.2.1	De compiler controleert de inhoud van een enum variabele	63
14.2.2	Visual Studio helpt bij het invullen van een enum variabele	63
14.3	Enums en EF	63
15	VIEWS	65
15.1	Algemeen	65
15.2	Een view aanmaken	65
15.3	De data van een view lezen vanuit SQL	65
15.4	De voorbeeldview	65
15.5	De entities definiëren die horen bij de view	65
15.6	De entities aanspreken vanuit code	66
16	STORED PROCEDURES	67
16.1	Algemeen	67
16.2	Een stored procedure aanmaken	67
16.3	Een stored procedure oproepen vanuit SQL	67
16.4	Stored procedures oproepen met EF	67
16.4.1	Een stored procedure die data terug geeft in de vorm van entities	67
16.4.2	Een stored procedure die data terug geeft in een vorm die niet overeenstemt met de structuur van een entity	68
16.4.3	Een stored procedure die geen data terug geeft	69
16.4.4	Een stored procedure die data leest als een scalar value	70
17	CODE FIRST	72
17.1	Algemeen	72
17.2	De entity classes voor de nieuwe database	72
17.3	De DbContext class	73
17.4	De connectionstring	73
17.5	De DbContext gebruiken	73
17.6	De database opnieuw maken	74
17.7	De aangemaakte table structuur verfijnen	74
17.7.1	Expliciet de table naam instellen	74
17.7.2	Expliciet de kolomnaam instellen	74

17.7.3	Een kolom instellen als verplicht in te vullen	74
17.7.4	Een kolom instellen als niet verplicht in te vullen	75
17.7.5	Het maximum aantal tekens in een varchar kolom instellen	75
17.7.6	Het kolomtype instellen	75
17.7.7	De property instellen die bij de primary key hoort.	75
17.7.8	Een primary key die geen int met autonumber is	75
17.8	Complex type	76
17.9	Inheritance	77
17.9.1	Table per hierarchy (TPH)	77
17.9.2	Table per type (TPT)	78
17.9.3	Table per concret class (TPC)	78
17.10	Associaties tussen entities	79
17.10.1	Één op veel associaties	79
17.10.2	Veel-op-veel associaties	80
17.10.3	Een associatie naar dezelfde tabel	81
18	WPF	83
18.1	De entity classes voor de nieuwe database	83
18.2	De DbContext class	83
18.3	De connectionstring	84
18.4	Een instantie van de DbContext class	84
18.5	Een ListBox tonen met data uit de database	84
18.6	Een ListBox met gerelateerde data tonen	84
18.7	Een DataGrid tonen met data uit de database	85
18.8	Data wijzigen	85
19	COLOFON	86

1 INLEIDING

1.1 Doelstelling

Je leert in deze module het ADO.NET entity framework gebruiken. Dit helpt je om objecten (in het interne geheugen) in verband te brengen met records (in een relationele database).

We gebruiken in deze cursus EF als afkorting voor entity framework.

1.2 Vereiste voorkennis

- C# PF
- SQL
- WPF

1.3 Nodige software

- Visual Studio 2013 (met Update 4)
- SQL Server/ SQL Server Express.
EF werkt samen met veel merken databases (SQL Server, Oracle, DB2, MySQL, ...)
Je gebruikt in deze cursus SQL Server. Je kan een volwaardige SQL Server gebruiken of SQL Server Express (die met Visual Studio is meegeleverd).
- Als je SQL Server (Express) versie 2014, kies je in Visual Studio in het menu *TOOLS* de opdracht *Extensions and Updates*. Je kiest links *Updates*. Je kiest rechts *Microsoft SQL Server Update for database tooling*. Je kiest daarbij *Update*. Deze opdracht downloadt een update die je installeert.
- SQL Server Management Studio

2 DE OBJECT-RELATIONAL MISMATCH

2.1 Algemeen

Je stelt data in het interne geheugen voor als objecten. Dit zijn instanties van classes.

Je stelt dezelfde data in een database voor als records in een table.

De manier waarop je data voorstelt als objecten, stemt niet helemaal overeen met de manier waarop je die data voorstelt als records. Dit heet 'the object-relational mismatch'.

Dit verschil heeft meerdere aspecten, hieronder uitgelegd.

2.2 Granularity

Granularity ('korreligheid') geeft aan in welke mate je data kan opsplitsen in onderdelen.

Een database heeft slechts twee granularity niveaus: tabellen en kolommen.

- Tabellen bevatten kolommen
- Een kolom bevat een enkelvoudige waarde (getallen, datums, tekst) die je niet verder kan opsplitsen

C# heeft op het eerste zicht maar twee granularity niveaus: classes en properties.


- Classes bevatten properties
- Een property kan een enkelvoudige waarde bevatten (int, decimal, ...)
Een property kan echter ook een reference zijn naar een object dat ook properties bevat ...

Op die manier is de granularity van classes oneindig.

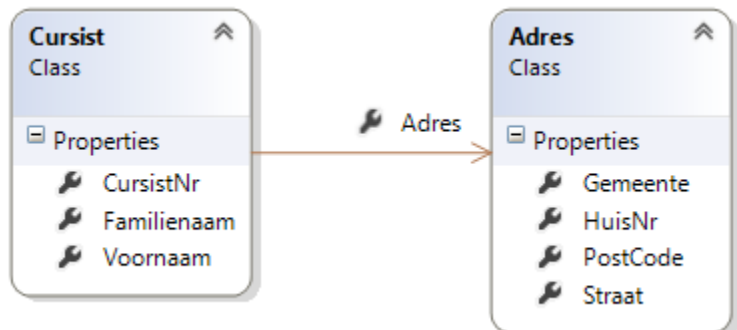
Voorbeeld: het gegeven Cursist.

- In de database bevat één table alle cursisten eigenschappen
- Deze eigenschappen zijn in C# verdeeld over twee classes.

De database

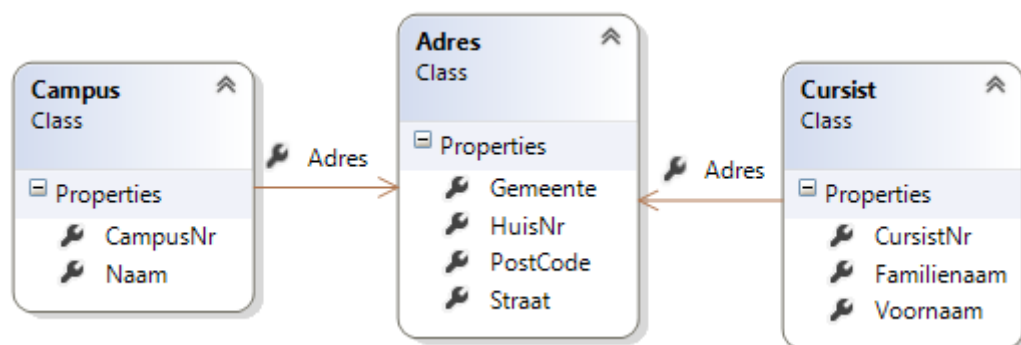
Cursisten	
	CusistNr
	Voornaam
	Familienaam
	Straat
	HuisNr
	PostCode
	Gemeente

De bijbehorende classes



- De class Cursist bevat een reference naar de class Adres.
- De class Adres bevat de properties Gemeente, HuisNr, Postcode en Straat

Een aparte class Adres is handig: je kan ze ook gebruiken vanuit andere classes:



Een class kan zelfs meerdere keer verwijzen naar één andere class.

Een class Klant verwijst bijvoorbeeld twee keer naar de class Adres

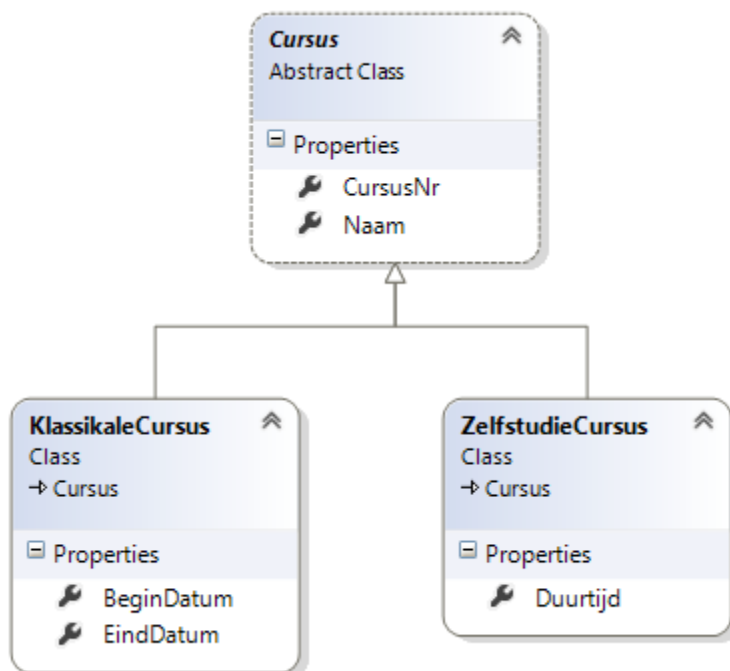
- Een eerste keer voor het facturatieadres
- Een tweede keer voor het leveringsadres



2.3 Inheritance

Inheritance is een essentieel onderdeel van C#.

Voorbeeld: de classes KlassikaleCursus en ZelfstudieCursus erven van de class Cursus:




Inheritance bestaat niet in een database. Je kan er inheritance enkel nabootsen, op drie manieren


- Table per concrete class
- Table per class hierarchy
- Table per subclass

2.3.1 Table per concrete class

De database bevat één table per niet-abstrakte class.

De table bevat kolommen voor alle properties van de class, inclusief de geërfde properties:

KlassikaleCursussen	
	CursusNr
	Naam
	BeginDatum
	EindDatum

Zelfstudie cursussen	
	CursusNr
	Naam
	DuurTijd

Voorbeelddata:

De table KlassikaleCursussen:

CursusNr	Naam	BeginDatum	EindDatum
1	Frans voor beginners	01-09-2007	11-09-2007
2	Frans voor gevorderden	12-09-2007	22-09-2007

De table ZelfstudieCursussen:


CursusNr	Naam	DuurTijd
1	Franse correspondentie	5
2	Engelse correspondentie	5

Nadeel van table per concrete class:

Als je een property toevoegt aan de base class, moet je in de database aan meerdere tables een kolom toevoegen. Als je bijvoorbeeld de property Prijs toevoegt aan de class Cursus, moet je een kolom Prijs toevoegen aan de table KlassikaleCursussen én aan de table ZelfstudieCursussen.

2.3.2 Table per class hierarchy

De database bevat één table voor de complete class inheritance hiërarchie. Deze table bevat kolommen voor alle properties van alle classes van de hiërarchie:

Cursussen	
	CursusNr
	Naam
	BeginDatum
	EindDatum
	DuurTijd
	Soort

Je neemt in de table ook een kolom op die aangeeft bij welke subclass een record hoort. In het voorbeeld is dit de kolom Soort. De kolom bevat K als het record een klassikale cursus voorstelt en bevat Z als het record een zelfstudiecursus voorstelt.

Voorbeelddata:

De table Cursussen:

CursusNr	Naam	BeginDatum	EindDatum	DuurTijd	Soort
1	Frans voor beginners	01/09/2007	11/09/2007		K
2	Frans voor gevorderden	12/09/2007	22/09/2007		K
3	Franse correspondentie			5	Z
4	Engels correspondentie			5	Z

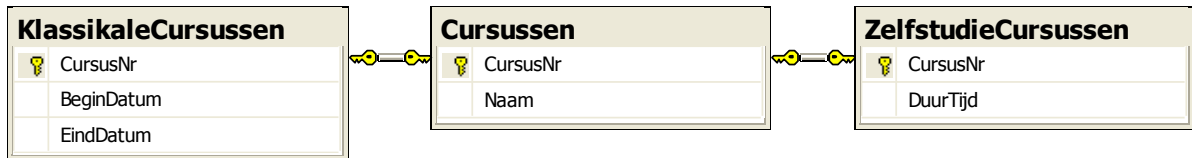
Nadeel van table per class hierarchy:

Je kan op de kolommen die horen bij properties van de subclasses geen constraints toepassen, want een constraint geldt voor alle records van de table. Je kan bijvoorbeeld geen not-null constraint op de kolom DuurTijd plaatsen, want deze kolom vul je enkel in bij een zelfstudiecursus.

2.3.3 Table per subclass

De database bevat één table per class uit de class inheritance.

- Iedere table bevat enkel kolommen voor de properties die de bijbehorende class niet erft.
- Een table die hoort bij een subclass bevat een primary key die ook een foreign key is naar de primary key van de table die hoort bij de base class:



De kolom CursusNr is in de table Cursussen een autonumber kolom, maar niet in de tables KlassikaleCursussen en ZelfstudieCursussen.

Voorbeelddata:

De table Cursussen:

CursusNr	Naam
1	Frans voor beginners
2	Frans voor gevorderden
3	Engels voor beginners
4	Engels voor gevorderden
5	Franse correspondentie
6	Engelse correspondentie

De table Klassikalecursussen:

CursusNr	BeginDatum	EindDatum
1	01-09-2007	11-09-2007
2	12-09-2007	22-09-2007
3	01-09-2007	11-09-2007
4	12-09-2007	22-09-2007

De table Zelfstudiecursussen:

CursusNr	DuurTijd
5	5
6	5

Nadeel van table per subclass:

Je moet twee tables joinen om de gegevens van klassikale cursussen of zelfstudiecursussen op te halen. Dit benadeelt de performantie.

Je ziet dat geen enkel van de drie inheritance nabootsingen perfect is.

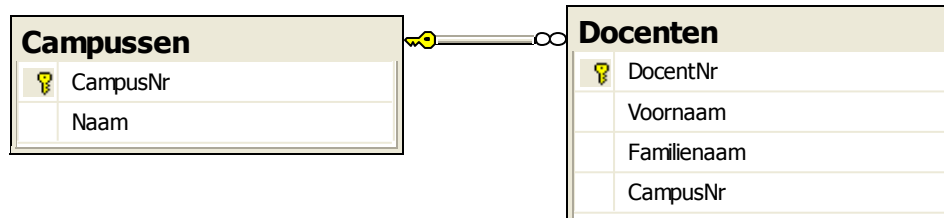
2.4 Associaties

2.4.1 Één op veel Associaties

Je stelt een één op veel associatie in de database voor met twee tables.

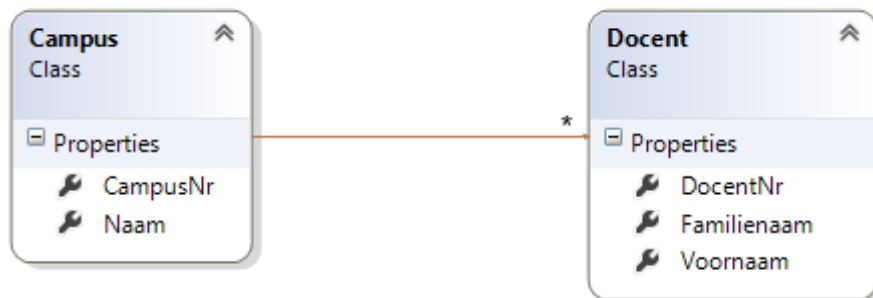
De table aan de veel zijde van de associatie bevat een foreign key kolom, die verwijst naar de primary key kolom van de table aan de één zijde van de associatie.

Voorbeeld:



Je stelt dezelfde één op veel associatie in C# voor met twee classes,

die je verbindt met een associatie. Associaties zijn geen getallen, maar references !



De class aan de veel zijde van de associatie (**Docent**) bevat één reference met als type de class aan de één zijde van de associatie (**Campus**). **Docent** bevat dus een reference variabele van het type **Campus**, waarmee je bijhoudt welke campus bij de docent hoort:

```
private Campus campusValue;
```

De class aan de één zijde van de associatie bevat een verzameling references met als type de class aan de veel zijde van de associatie. **Campus** bevat dus een verzameling reference variabelen van het type **Docent**, waarmee je bijhoudt welke docenten bij de campus horen:

```
private List<Docent> docentenValue;
```

2.4.2 Veel op veel associaties

Je hebt in een relationele database enkel één op veel relaties en één op één relaties.

Je hebt een tussentable nodig om een veel-op-veel relatie tussen twee tables voor te stellen.

De oorspronkelijke tables hebben dan een één-op-veel relatie met deze tussentable.

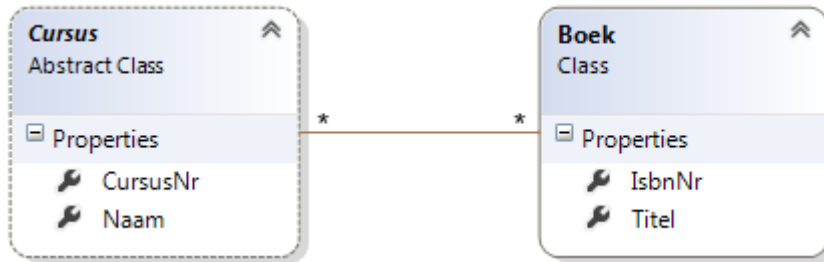
Voorbeeld: in een cursus worden meerdere boeken gebruikt.

Één boek wordt soms gebruikt in meerdere cursussen.

De database:



Je kan in C# die veel-op-veel associatie tussen Cursus en Boek uitdrukken zonder tussenclass:

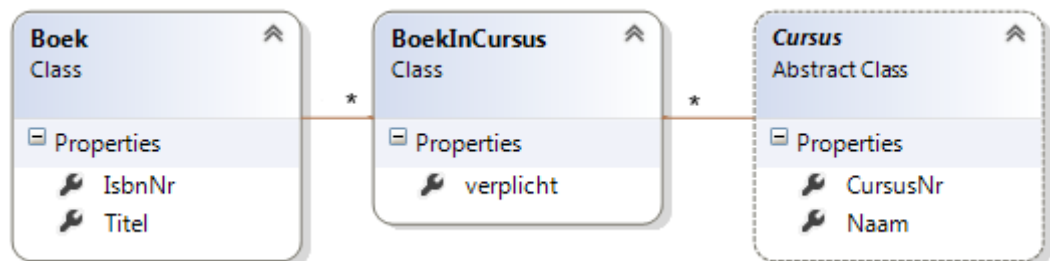


Beide classes bevatten een verzameling references met als type de class aan de andere zijde van de associatie.

- Cursus bevat dus een verzameling reference variabelen van het type Boek, waarmee je bijhoudt welke boeken bij de cursus horen: `private List<Boek> boeken;`
- Boek bevat dus een verzameling reference variabelen van het type Cursus, waarmee je bijhoudt welke cursussen van het boek gebruik maken: `private List<Cursus> cursussen;`



Opmerking: zodra je over de associatie zelf properties bijhoudt, heb je ook een tussenclass nodig (zoals je in de database een tussentable hebt). Voorbeeld: de relatie tussen Boek en Cursus. Een boek kan een verplicht te lezen boek of een optioneel te lezen boek zijn in een cursus. De table CursussenBoeken bevat dan een kolom *verplicht*. De aangepaste classes:



2.5 Navigeren door associaties

Je navigeert in C# van een object naar een geassocieerd object door de reference variabele te volgen die de associatie definieert. Je toont de titels van de boeken die bij een cursus horen als:

```

public void ToonCursusEnBijbehorendeBoeken(Cursus cursus)
{
    Console.WriteLine(cursus.Naam);
    foreach (var boek in cursus.Boeken)
    {
        Console.WriteLine(boek.IsbnNr);
        Console.WriteLine(boek.Titel);
    }
}
  
```

Je gebruikt in een database een SQL join om gegevens uit gerelateerde tables op te halen. Je gebruikt volgend SQL statement om de gegevens van één cursus (bvb. nr. 7), samen met de gebruikte boeken binnen die cursus op te halen:

```

select Naam, Boeken.IsbnNr, Titel
from Cursussen left join CursussenBoeken
on Cursussen.Cursusnr = CursussenBoeken.CursusNr
join Boeken on CursussenBoeken.IsbnNr = Boeken.IsbnNr
where Cursussen.CursusNr = 7
  
```



Opmerking: de left outer join tussen de tables Cursussen en CursussenBoeken geeft je ook informatie over cursussen die géén gerelateerde boeken hebben.

2.6 ORM (Object-relational mapping)

Je stelt gegevens in C# dus op een andere manier voor dan in de database.

Je zal ergens een vertaalslag moeten doen tussen deze verschillende visies.

Deze vertaalslag zelf uitschrijven is niet gemakkelijk en vergt veel tijd.

Object-relational mapping is juist het converteren van de object georiënteerde visie naar de database visie. Een ORM library helpt je deze conversie te doen.

EF is een ORM library van Microsoft.

Een ORM library biedt volgende meerwaarden:

- Productiviteit.
Zelf de code schrijven die de vertaalslag doet tussen de twee verschillende visies op gegevens vraagt veel code (en dus ook tijd).
- Onderhoudbaarheid
Het databaseschema wijzigt in de tijd. Het class diagram wijzigt in de tijd.
Deze visies continu op mekaar afstemmen gaat gemakkelijker met een ORM library dan zonder ORM library.
- Databasemerk onafhankelijkheid.
Een ORM library neemt de verschillen tussen databases voor zijn rekening.
Een voorbeeld is het automatisch nummeren van de primary key kolom bij nieuwe records.
Je doet dit bij sommige databases (bvb SQL Server) met autonumber kolommen, bij andere databases (bvb. Oracle) met sequences.

2.7 EDMX bestand versus Code First

EF heeft twee manieren de verbanden leggen tussen de database tables en de classes

- in een XML bestand met de extensie EDMX.
- Met attributen die je tikt in je classes. Dit heet code first.

Je leert eerst te werken met een EDMX bestand, daarna code first.

3 DE DATABASE



Je werkt in deze cursus met de database Opleidingen.

Het script `CreateOpleidingen.sql` uit het akenmateriaal maakt deze database.

Je voert dit script uit:

- Je start SQL Server Management Studio
- Je tikt in het venster *Connect to Server* `.\sqlexpress`, als je SQL Express gebruikt, of je kiest je volwaardige SQL Server uit de lijst.
- Je laat de keuze bij *Authentication* op *Windows Authentication*.
- Je kiest *Connect*.
- Je kiest in het menu *File* de opdracht *Open* en de vervolgoopdracht *File*.
- Je selecteert *CreateOpleidingen.sql* en je kiest *Open*.
- Je kiest in de knoppenbalk de opdracht *Execute*.
- Je klikt in het linkerdeel met de rechtermuisknop op de map *Databases* en je kiest *Refresh*.
- Je klapt de map *Databases* open en je ziet de database *Opleidingen*.

Deze database bevat de tables *Campussen* en *Docenten*:

Campussen				Docenten			
Column Name	Data Type	Allow Nulls		Column Name	Data Type	Allow Nulls	
 CampusNr	int	<input type="checkbox"/>		 DocentNr	int	<input type="checkbox"/>	
Naam	nvarchar(50)	<input type="checkbox"/>		Voornaam	nvarchar(50)	<input type="checkbox"/>	
Straat	nvarchar(50)	<input type="checkbox"/>		Familienaam	nvarchar(50)	<input type="checkbox"/>	
HuisNr	nvarchar(10)	<input type="checkbox"/>		Wedde	decimal(10, 2)	<input type="checkbox"/>	
PostCode	nvarchar(10)	<input type="checkbox"/>		CampusNr	int	<input type="checkbox"/>	
Gemeente	nvarchar(50)	<input type="checkbox"/>					



Opmerking: de primary key kolommen van deze tables zijn autonumber kolommen.

Je zorgt er voor dat je de database waarmee je applicatie zal samenwerken ook ziet vanuit Visual Studio. Dit is belangrijk voor de volgende stappen:

- Je kiest in *Visual Studio* in het menu *VIEW* de opdracht *Server Explorer*.
- Je klikt in de *Server Explorer* met de rechtermuisknop op *Data Connections* en je kiest *Add Connection*
- Als het titel van het venster *Change Data Source* is, kies je *Microsoft SQL Server* bij *Data Source* en je kiest *Continue*
- Je tikt bij *Server name:* `.\sqlexpress`, als je SQL Express gebruikt, of je kiest je volwaardige SQL Server.
- Je kiest bij *Select or enter a database name:* *Opleidingen* en je kiest *OK*.

4 HET ENTITY DATA MODEL (EDM)

4.1 Algemeen

Een entity class is een class die een gegeven uit de werkelijkheid voorstelt.

De classes Docent en Campus zijn bijvoorbeeld entity classes.

Objecte van entity classes heten entities.

Het entity data model beschrijft:

- De entity classes van je applicatie en de verbanden (inheritance, associatie) tussen deze entity classes.
- De structuur van de database die bij je applicatie hoort.
- Het verband tussen de entity classes en de database:
 - Welke entity class hoort bij welke table ?
 - Welke property van een entity class hoort bij welke kolom ?

Het entity data model is een XML bestand met de extensie *edmx*.

Je bewerkt dit XML bestand met een grafische designer van Visual Studio.

Visual Studio maakt daarna, gebaseerd op dit entity data model:

- De entity classes als C# classes die je vanuit je applicatie aanspreekt.
- Een "DbContext class" waarmee je van je applicatie:
 - Records leest uit de database als entity.
 - Records toevoegt gebaseerd op entity.
 - Records wijzigt gebaseerd op gewijzigde entities.
 - Records verwijdert die horen bij entities.

Je leert in dit hoofdstuk het entity data model en de bijbehorende designer kennen.

4.2 Je eerste EDM

Je maakt in Visual Studio een Console Application project met de naam *EFCursus*

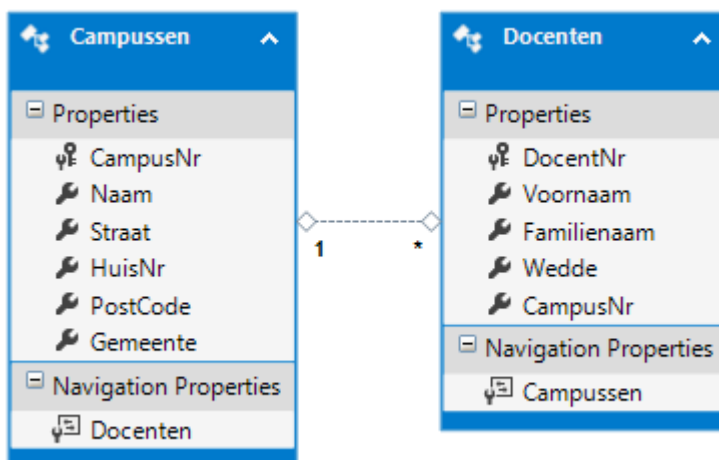
Je voegt aan het project een entity data model toe:


- Je klikt in de *Solution Explorer* met de rechtermuisknop op je project, je kiest *Add* en daarna *New Item*.
 - Je kiest in het middendeel *ADO.NET Entity Data Model*.
 - Je tikt bij *Name Opleidingen* en je kiest *Add*.
 - Je krijgt een vraag: *What should the model contain?*
 - *EF Designer from database*
Visual Studio maakt hierbij entity classes die lijken op de tabelstructuren van de bijbehorende database. Zo win je tijd.
Je kan deze entity classes daarna uitbreiden (properties, methods, ...)
 - *Empty EF Designer model*
Je definieert bij deze keuze de entity classes vanaf nul. Dit vraagt tijd.
 - *Empty Code First Model*
Code First wordt later in de cursus uitgelegd
 - *Code First from database*
Code First wordt later in de cursus uitgelegd
- Je kiest *EF Designer from database* en je kiest *Next*.
- Je kiest bij *Which data connection ...* één van de databaseconnecties die gedefinieerd zijn in de *Server Explorer*.
Je kiest in deze cursus de connectie naar de database *Opleidingen*.
 - Als de vraag *Do you want to include this sensitive data in the connection string?* beschikbaar is, antwoord je *Yes*.

- Je laat het vinkje staan bij *Save entity connection settings in App.Config* staan. Visual Studio maakt in dit configuratiebestand van je applicatie een onderdeel *OpleidingenEntities* waarin de databaseconnectie gedefinieerd is. Je past dit bestand aan als de databaseconnectie moet wijzigen.
- Je kiest *Next*.
- Je kiest *Entity Framework 6.x* en je kiest *Next*
- Je klappt bij *Which database objects do you want to include in your model* het onderdeel *Tables* open. Je klappt daarbinnen *dbo* open. Je plaatst een vinkje bij *Campussen* en *Docenten*.
- Je plaatst geen vinkje bij *Pluralize or singularize generated object names*. Dit is enkel nuttig als de tables in de database Engelse namen hebben. Je kan in dat geval wel een vinkje plaatsen. Visual Studio maakt dan bij een table met een naam in meervoudsvorm (bvb. Clients) een bijbehorende class met een naam in enkelvoudsvorm (bvb. Client).
- Je laat het vinkje staan bij *Include foreign key columns in the model*. Visual Studio stelt dan een foreign key kolom op twee manieren voor in de bijbehorende class:
 - één keer als een property met als type de class die hoort bij de table waar de foreign key naar verwijst. (Visual Studio stelt de foreign key kolom *CampusNr* in de table *Docenten* voor in de class *Docenten* als een property met als type de class *Campussen*). Je gebruikt deze property om van een docent campusinformatie op te halen (zoals de naam van de campus).
 - één keer als een property met als type het .net type dat hoort bij het type van de foreign key kolom. (Visual Studio stelt de foreign key kolom *CampusNr* in de table *Docenten* in de class *Docenten* ook voor als een property met als type *int*). Je gebruikt in je code deze property als je van de campus die bij een docent hoort enkel het campusnummer wil ophalen. Dit ophalen gaat zeer snel, omdat dit campusnummer zich op dat moment al in het interne geheugen bevindt, en niet uit het gerelateerde record van de table *Campussen* moet gelezen worden.

Als je het vinkje wegdoet, maakt Visual Studio enkel de eerste voorstelling van de foreign key (de reference naar de class *Campussen*).
- Je kiest *Finish*. Je kiest *OK* bij de Security warnings.

Je ziet in de designer van *Opleidingen.edmx* het ontwerp van de entity classes *Campussen* en *Docenten*. Ze lijken op de structuur van de tables *Campussen* en *Docenten* in de database:



Eri is een 1 op veel associatie  tussen *Campussen* en *Docenten*. Visual Studio maakte die omdat de database een één op veel relatie bevat tussen de tables *Campussen* en *Docenten*.

Je ziet per entity class twee soorten properties:

- Navigation properties. Dit zijn references naar geassocieerde entity classes.
 - De navigation property *Docenten* in de entity class *Campussen* verwijst naar de bijbehorende *Docenten* objecten.
 - De navigation property *Campussen* in de entity class *Docenten* verwijst naar de bijbehorende *Campus* entity.
- Gewone properties. Deze bevatten data die geen references zijn naar andere entity classes

Als je een property aanklikt,
zie je in het Properties venster
detailinformatie over die property.

Voorbeeld: de property *CampusNr* van *Campussen*:


Code Generation	
Getter	Public
Setter	Public
General	
Concurrency Mode	None
Default Value	(None)
Documentation	
Entity Key	True
Name	CampusNr
Nullable	False
StoreGeneratedPattern	Identity
Type	Int32

De namen van de entity classes (*Campussen*, *Docenten*) zijn overgenomen uit de database. De entity class *Campussen* stelt echter één campus voor, geen verzameling campussen en de entity class *Docenten* stelt één docent voor.

Je wijzigt daarom de namen van deze entity classes:

- Je dubbelklikt in de hoofding van de entity class *Campussen* het woord *Campussen*, je corrigeert naar *Campus* en je drukt *Enter*.
- Je dubbelklikt in de hoofding van de entity class *Docenten* het woord *Docenten*, je corrigeert naar *Docent* en je drukt *Enter*.

Als je de hoofding van een entity class aanklikt,
zie je in het Properties venster detailinformatie
over die entity class. Voorbeeld: de entity class
Campus:

Code Generation	
Abstract	False
Access	Public
Diagram	
Fill Color	 0; 122; 204
General	
Base Type	(None)
Documentation	
Entity Set Name	Campussen
Name	Campus

Één van deze properties heet *Entity Set Name*.

Deze naam (*Campussen*) staat voor de verzameling met alle entities uit de database.

Het is ook beter de namen van enkele *Navigation Properties* te corrigeren.

- De naam van de Navigation Property *Docenten* in de entity class *Campus* is OK: één class heeft meerdere docenten.
- De naam van de Navigation Property *Campussen* in de entity class *Docent* is niet OK: een Docent heeft maar één Campus.

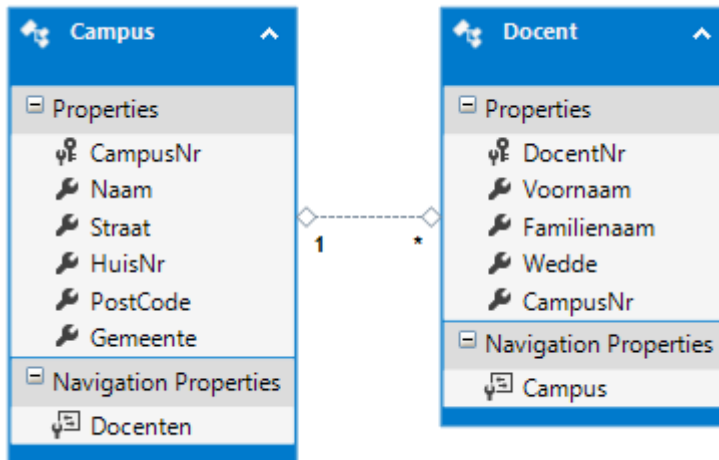
Je wijzigt daarom de naam van deze Navigation Property naar *Campus*:

- Je klikt op de Navigation Property *Campussen* in de entity class *Docent*.
- Je klikt nog eens op dezelfde Navigation Property
- Je corrigeert naar *Campus* en je drukt *Enter*.



Opmerking: Het is belangrijk deze naamcorrecties te doen vooraleer je het EDM gebruikt in je code. Als je achteraf de namen in het EDM wijzigt, moet je ook de verwijzingen in je code wijzigen !

Resultaat:



Je slaat het EDM op met de knop in de toolbar. Je krijgt hierbij een security warning, die niet belangrijk is. Je krijgt deze warning iedere keer je het EDM opslaat. Je plaatst daarom een vinkje bij *Do not show this message again* en je kiest *OK*

4.3 HET EDM als XML bestand

4.3.1 Algemeen

Je bekijkt het EDM tot nu met de grafische designer. Het EDM is in feite een XML bestand. Je krijgt hier wat inzicht in de opbouw van het XML bestand.

Je bekijkt het EDM als XML:

- Je klikt in de Solution Explorer met de rechtermuisknop op *Opleidingen*.
- Je kiest *Open With*.
- Je kiest *XML (Text) Editor*.
- Je kiest *OK*.
- Je kiest *Yes* op de vraag *The document ... is already open. Do you want to close it?*

Je klapt met de onderdelen `<edmx:Runtime>` en `<Designer ...>` dicht.

Dan zie je de grove opbouw van het edmx bestand:

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>...</edmx:Runtime>
  <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <Designer xmlns="http://schemas.">...</Designer>
</edmx:Edmx>
```

- Het onderdeel `<edmx:Runtime>` bevat het eigenlijke ontwerp van het EDM. Je leert dit verder in detail kennen.
- Visual Studio houdt in het onderdeel `<Designer ...>` de positie van de entity classes bij in de grafische designer en is niet zo interessant.

Je klikt op het onderdeel `<edm:Runtime>` open. Je klikt daarin de onderdelen `<edm:StorageModels>`, `<edm:ConceptualModels>` en `<edm:Mappings>` dicht. Je ziet dan de grove opbouw van het onderdeel `<edm:Runtime>`:

```
<edm:Runtime>
  <!-- SSDL content -->
  <edm:StorageModels>...</edm:StorageModels>
  <!-- CSDL content -->
  <edm:ConceptualModels>...</edm:ConceptualModels>
  <!-- C-S mapping content -->
  <edm:Mappings>...</edm:Mappings>
</edm:Runtime>
```

- **StorageModels** Bevat de databasestructuur.
- **ConceptualModels** Bevat de structuur van de entity classes.
- **Mappings** Bevat de verbanden tussen de entity classes en de database.

4.3.2 StorageModels

Het onderdeel `StorageModels` bevat de structuur van iedere table. De structuur van de table *Campussen*:

```
<EntityType Name="Campussen">
  <Key>
    <PropertyRef Name="CampusNr" />
  </Key>
  <Property Name="CampusNr" Type="int" StoreGeneratedPattern="Identity"
    Nullable="false" />
  <Property Name="Naam" Type="nvarchar" MaxLength="50" Nullable="false" />
  <Property Name="Straat" Type="nvarchar" MaxLength="50" Nullable="false" />
  <Property Name="HuisNr" Type="nvarchar" MaxLength="10" Nullable="false" />
  <Property Name="PostCode" Type="nvarchar" MaxLength="10" Nullable="false" />
  <Property Name="Gemeente" Type="nvarchar" MaxLength="50" Nullable="false" />
</EntityType>
```

Het onderdeel `StorageModels` bevat ook informatie over de relaties tussen de tables. De relatie tussen de tables *Campussen* en *Docenten*

```
<Association Name="FK__Docenten__Campus__1273C1CD">
  <End Role="Campussen" Type="Self.Campussen" Multiplicity="1" />
  <End Role="Docenten" Type="Self.Docenten" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Campussen">
      <PropertyRef Name="CampusNr" />
    </Principal>
    <Dependent Role="Docenten">
      <PropertyRef Name="CampusNr" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

4.3.3 ConceptualModels

Het onderdeel `ConceptualModels` bevat informatie over de opbouw van de entity classes. De entity class *Campus*:

```
<EntityType Name="Campus">
  <Key>
    <PropertyRef Name="CampusNr" />
  </Key>
  <Property Name="CampusNr" Type="Int32" Nullable="false"
    annotation:StoreGeneratedPattern="Identity" />
  <Property Name="Naam" Type="String" MaxLength="50" FixedLength="false"
    Unicode="true" Nullable="false" />
</EntityType>
```

```
<Property Name="Straat" Type="String" MaxLength="50" FixedLength="false"
  Unicode="true" Nullable="false" />
<Property Name="HuisNr" Type="String" MaxLength="10" FixedLength="false"
  Unicode="true" Nullable="false" />
<Property Name="PostCode" Type="String" MaxLength="10" FixedLength="false"
  Unicode="true" Nullable="false" />
<Property Name="Gemeente" Type="String" MaxLength="50" FixedLength="false"
  Unicode="true" Nullable="false" />
<NavigationProperty Name="Docenten"
  Relationship="Self.FK__Docenten__Campus__1273C1CD" FromRole="Campussen"
  ToRole="Docenten" />
</EntityType>
```

Het onderdeel ConceptualModels bevat ook informatie over de associaties tussen entity classes. De associatie tussen de entities *Campus* en *Docent*:

```
<Association Name="FK__Docenten__Campus__1273C1CD">
  <End Role="Campussen" Type="OpleidingenModel.Campus" Multiplicity="1" />
  <End Role="Docenten" Type="OpleidingenModel.Docent" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Campussen">
      <PropertyRef Name="CampusNr" />
    </Principal>
    <Dependent Role="Docenten">
      <PropertyRef Name="CampusNr" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

4.3.4 Mappings

Het onderdeel Mappings bevat informatie over welke table bij welke entity class hoort. Het verband tussen de table *Campussen* en de entity class *Campus*:

```
<EntitySetMapping Name="Campussen">
  <EntityTypeMapping TypeName="OpleidingenModel.Campus">
    <MappingFragment StoreEntitySet="Campussen">
      <ScalarProperty Name="CampusNr" ColumnName="CampusNr" />
      <ScalarProperty Name="Naam" ColumnName="Naam" />
      <ScalarProperty Name="Straat" ColumnName="Straat" />
      <ScalarProperty Name="HuisNr" ColumnName="HuisNr" />
      <ScalarProperty Name="PostCode" ColumnName="PostCode" />
      <ScalarProperty Name="Gemeente" ColumnName="Gemeente" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

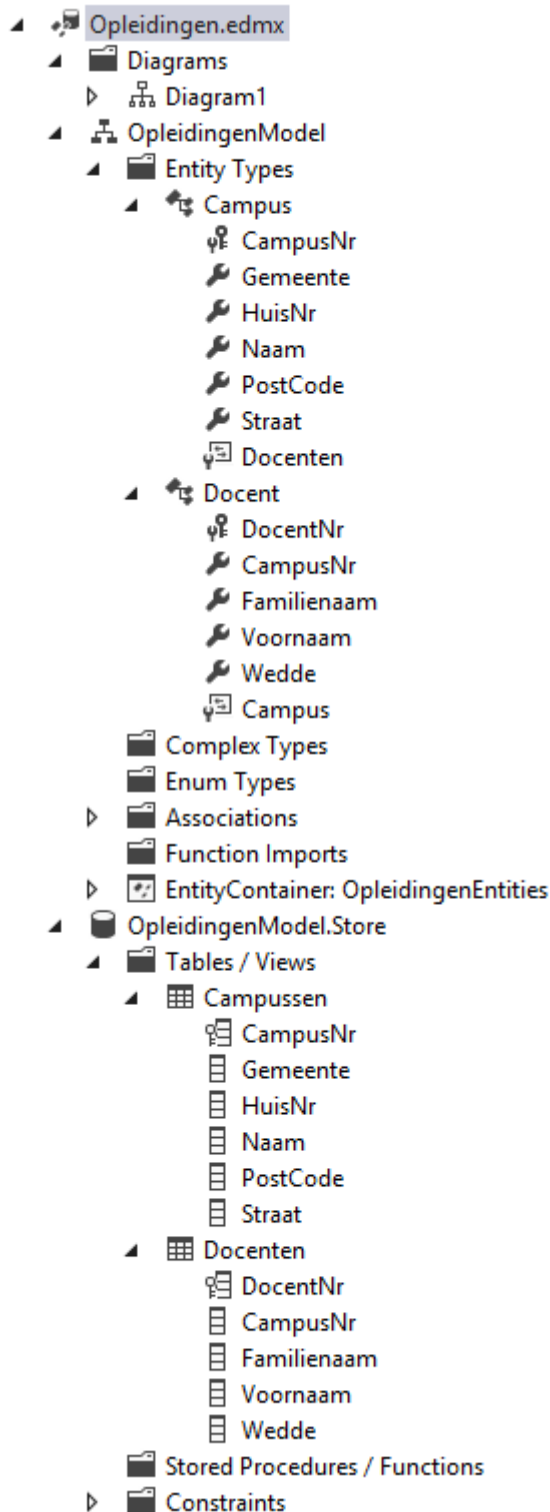
Je mag het venster met deze XML sluiten en *Opleidingen.edmx* in de *Solution Explorer* dubbel klikken. Visual Studio opent dit bestand dan terug in de designer.

4.4 De Model Browser

De model browser toont de opbouw van het EDM op nog een andere manier.

Je ziet de Model Browser door in de achtergrond van het EDM

te klikken met de rechtermuisknop en *Model Browser* te kiezen:



Je ziet bij OpleidingenModel de classes die het EDM aanmaakt:

- Campus (entity class)
- Docent (entity class)

Je ziet bij OpleidingModel.Store de database tables die bij het EDM horen:

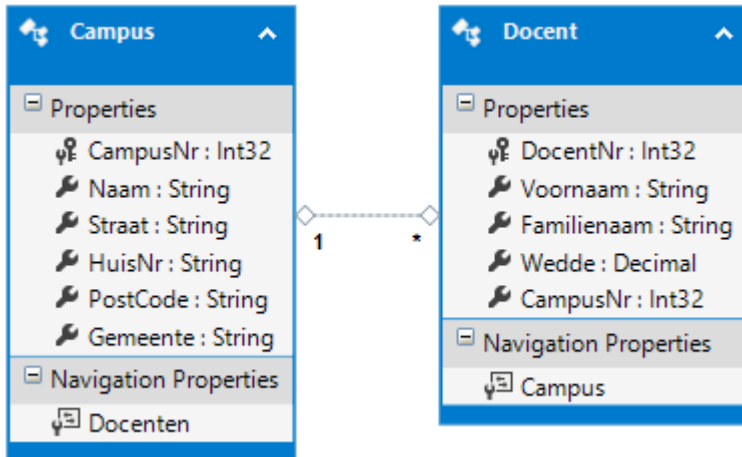
- Campussen
- Docenten

4.5 Extra technieken in de grafische designer

Je kan in die grafische designer ook de types van de entity properties zien:

- Je klikt in het achtergrond van de designer met de rechtermuisknop.
- Je kiest *Scalar Property Format* en daarna *Display Name and Type*.

Het resultaat:



Je kan ook de mapping details zien: welke table hoort bij welke entity class en welke kolom binnen die table hoort bij één property van de entity class:

- Je klikt met de rechtermuisknop ergens in de entity class *Docent*.
- Je kiest *Table Mapping*.

Resultaat:

Mapping Details - Docent			
	Column	Operator	Value / Property
Tables			
Maps to Docenten			
<Add a Condition>			
Column Mappings			
	DocentNr : int	↔	DocentNr : Int32
	Voornaam : nvarchar	↔	Voornaam : String
	Familienaam : nvarchar	↔	Familienaam : String
	Wedde : decimal	↔	Wedde : Decimal
	CampusNr : int	↔	CampusNr : Int32

4.6 De DbContext class

Visual Studio maakt per entity data model (in ons geval *Opleidingen.edmx*) één class, die erft van *DbContext*. Je gebruikt deze class om vanuit je applicatie de database aan te spreken.

De property *Entity Container Name* van het EDM bevat de naam van deze class. Je ziet de properties van het EDM als je in de het achtergrond van het EDM klikt en daarna in het properties venster kijkt. Bij ons is de naam *OpleidingenEntities*.

Een *DbContext* class implementeert de interface *IDisposable*.

Je moet een *DbContext* object dus 'opkuisen' na gebruik.

Als je de *DbContext* object aanmaakt met het sleutelwoord *using*, gebeurt deze opkuis automatisch:

```
using (var entities = new OpleidingenEntities())
{
    // Je doet hier op de variabele entities één of meerdere bewerkingen
}
```

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.

4.7 Entity classes uitbreiden met properties en methods

Je kan de entity classes uitbreiden met extra properties en methods.

De entity classes die het EDM heeft aangemaakt zijn partial classes. Jij kan een source file toevoegen aan het project. Je beschrijft in die source file dezelfde class ook als partial class. Je voegt in deze source file properties en of methods toe aan de entity class.

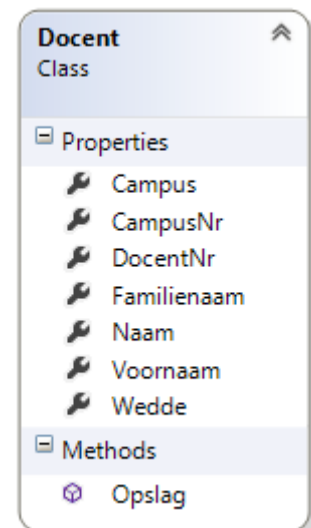
Voorbeeld: je voegt een readonly property Naam en een method Opslag toe aan de class Docent:

- Je kiest in het menu *PROJECT* de opdracht *Add Class*
- Je tikt *DocentUitbreiding* bij *Name* en je kiest *Add*
- Je wijzigt deze source file:

```
namespace EFCursus
{
    public partial class Docent
    {
        public string Naam
        {
            get
            {
                return Voornaam + ' ' + Familiennaam;
            }
        }
        public void Opslag(decimal bedrag)
        {
            Wedde += bedrag;
        }
    }
}
```

Je kan zien dat de class Docent de extra property en de extra method bevat:

- Je klikt in de Solution Explorer met de rechtermuisknop op *DocentUitbreiding.cs*.
- Je kiest *View Class Diagram*.



Bank maken: zie takenbundel

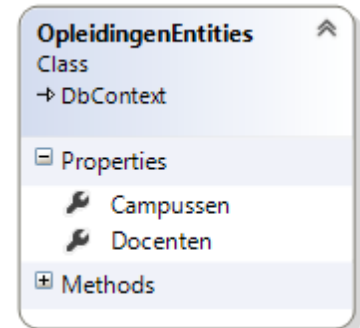
5 QUERIES

5.1 Algemeen

De DbContext class bevat per entity class een property waarvan de naam gelijk is aan de *Entity Set Name* property van die entity class.

Onze DbContext class *OpleidingenEntities* bevat de properties

- *Campussen*
Je spreekt met die property de records uit de table *campussen* aan.
- *Docenten*
Je spreekt met die property de records uit de table *docenten* aan.



5.1.1 Een foreach iteratie

Als je op de property *Docenten* van *OpleidingenEntities* een foreach iteratie uitvoert, doet EF volgende stappen:

- Het stuurt een SQL select statement naar de database om álle records uit de bijbehorende table (*Docenten*) op te vragen.
- Het maakt van ieder record een *Docent* entity.
- Het verzamelt deze entities in de property *Docenten* van het *OpleidingenEntities* object.

Je itereert dus met deze foreach over alle docenten uit de database.

Je kan dit uitproberen met volgende code in de method *Main* van *Program.cs*:

```
using (var entities = new OpleidingenEntities())
{
    foreach (var docent in entities.Docenten)
    {
        Console.WriteLine(docent.Naam);
    }
}
```

Deze manier van records lezen is beperkt:

je kan niet sorteren of filteren (bvb. enkel de docenten met een wedde vanaf 2000).

5.1.2 Een LINQ query

Als je een LINQ query uitvoert op de property *Docenten* van *OpleidingenEntities*, doet EF volgende stappen:

- Het vertaalt de LINQ query naar een SQL select statement.
- Het stuurt dit SQL statement naar de database om records uit de bijbehorende table (*Docenten*) op te vragen.
- Het maakt van ieder gevonden record een *Docent* entity.
- Het verzamelt deze *Docent* entities in één verzameling. Deze verzameling, van het type *IQueryable<Docent>*, is het resultaat van de LINQ query.
- Je kan met een foreach itereren over deze verzameling.

Je kan dit uitproberen in de method *Main* van *Program.cs*:

```
Console.WriteLine("Minimum wedde:");
decimal minWedde;
if (decimal.TryParse(Console.ReadLine(), out minWedde))
{
    using (var entities = new OpleidingenEntities())
    {
        var query = from docent in entities.Docenten
                     where docent.Wedde >= minWedde
                     orderby docent.Voornaam, docent.Familienaam
                     select docent;
        foreach (var docent in query)
        {
            Console.WriteLine("{0}: {1}", docent.Naam, docent.Wedde);
        }
    }
}
else
{
    Console.WriteLine("Tik een getal");
}
```

Je ziet verder in de cursus meer gespecialiseerde LINQ queries.

5.1.3 Query methods

In plaats van een query te definiëren als een LINQ query, kan je dezelfde query ook definiëren met query methods (bvb. de method *Where* en de method *OrderBy*).

Je probeert dit uit door de opdracht *var query* (over de volledige drie regels) te vervangen door:

```
var query = entities.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.
De lambda expressie krijgt één entity als parameter binnen. Je noemt deze parameter *docent*. Je geeft *true* terug als de entity in het resultaat mag voorkomen.
Je geeft *false* terug als de entity niet in het resultaat mag voorkomen.
- (2) Je gebruikt de method *OrderBy* om records te sorteren.
Je geeft een lambda expressie mee, waarin je sortering definieert.
De lambda expressie krijgt één entity als parameter binnen. Je noemt deze parameter *docent*. Je geeft in deze lambda expressie een property van deze entity terug.
EF sorteert records op de kolom die hoort bij die property.
- (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. Je geeft terug een lambda expressie. Deze lambda expressie werkt op dezelfde manier als de lambda expressie van de *OrderBy* method.



Opmerking: de methods *OrderBy* en *ThenBy* sorteren oplopend.
De methods *OrderByDescending* en *ThenByDescending* sorteren aflopend.

5.1.4 LINQ queries en queries met methods vergeleken

- LINQ queries zijn meestal leesbaarder dan queries gedefinieerd met query methods.
- Sommige programmaonderdelen zijn meer onderhoudbaar met query methods dan met LINQ queries. In het volgende voorbeeld (in de method Main) ziet de gebruiker de docenten met een wedde vanaf een in te tikken grens. De gebruiker kiest daarna hoe hij diedocenten sorteert. De versie met een LINQ query bevat drie sterk gelijkaardige queries:

```
Console.WriteLine("Minimum wedde:");
decimal minWedde;
if (decimal.TryParse(Console.ReadLine(), out minWedde))
{
    Console.WriteLine("Sorteren:1=op wedde, 2=op familienaam, 3=op voornaam:");
    var sorterenOp = Console.ReadLine();
    using (var entities = new OpleidingenEntities())
    {
        IQueryable<Docent> query; // het type van de variabele query is een
                                // LINQ query die Docent entities teruggeeft

        switch (sorterenOp)
        {
            case "1":
                query = from docent in entities.Docenten
                        where docent.Wedde >= minWedde
                        orderby docent.Wedde
                        select docent;

                break;
            case "2":
                query = from docent in entities.Docenten
                        where docent.Wedde >= minWedde
                        orderby docent.Familienaam
                        select docent; // deze query lijkt sterk op de vorige

                break;
            case "3":
                query = from docent in entities.Docenten
                        where docent.Wedde >= minWedde
                        orderby docent.Voornaam
                        select docent; // deze query lijkt sterk op de vorige

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

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

```
Console.WriteLine("Minimum wedde:");
decimal minWedde;
if (decimal.TryParse(Console.ReadLine(), out minWedde))
{
    Console.WriteLine("Sorteren:1=op wedde, 2=op familienaam, 3=op voornaam:");
    var sorterenOp = Console.ReadLine();
    Func<Docent, Object> sorteerLambda;
```

```

switch (sorterenOp)
{
    case "1":
        sorteerLambda = (docent) => docent.Wedde;
        break;
    case "2":
        sorteerLambda = (docent) => docent.Familienaam;
        break;
    case "3":
        sorteerLambda = (docent) => docent.Voornaam;
        break;
    default:
        Console.WriteLine("Verkeerde keuze");
        sorteerLambda = null;
        break;
}
if (sorteerLambda != null)
{
    using (var entities = new OpleidingenEntities())
    {
        var query = entities.Docenten
            .Where(docent => docent.Wedde >= minWedde)
            .OrderBy(sorteerLambda);
        foreach (var docent in query)
        {
            Console.WriteLine("{0}: {1}", docent.Naam, docent.Wedde);
        }
    }
}
else
{
    Console.WriteLine("U tikte geen getal");
}
}

```

5.2 Een entity zoeken op zijn primary key waarde

Je hoeft geen LINQ query te schrijven om een entity te zoeken op zijn primary key waarde. Je kan voor zo'n zoekoperatie de Find method gebruiken van de property die de verzameling entities voorstelt in de DbContext.

Voorbeeld in de method *Main* : je gebruikt de Find method op de property Docenten van OpleidingenEntities, om een docent te zoeken op zijn docentnummer.

- Je geeft als parameter aan de Find method de primary key waarde mee van de zoeken entity.
- Deze Find method geeft de entity terug, als deze entity voorkomt in de database.
- Deze Find method geeft null terug, als deze entity niet voorkomt in de database.

Je kan dit uitproberen in de method *Main* van *Program.cs*:

```

using (var entities = new OpleidingenEntities())
{
    Console.Write("DocentNr.:");
    int docentNr;
    if (int.TryParse(Console.ReadLine(), out docentNr))
    {
        var docent = entities.Docenten.Find(docentNr);
        Console.WriteLine(docent == null ? "Niet gevonden" : docent.Naam);
    }
    else
    {
        Console.WriteLine("U tikte geen getal");
    }
}

```

5.3 Gedeeltelijke objecten ophalen

De queries die je tot nu maakte, lezen uit de records álle kolommen en vullen hiermee per entity alle bijbehorende properties. Dit kan de performantie benadelen als je in een programma onderdeel slechts enkele properties per entity nodig hebt.

Je gebruikt als oplossing een LINQ query, waarin je slechts enkele properties opvraagt. EF vertaalt zo'n LINQ query naar een SQL select statement dat enkel de kolommen leest die bij die properties horen.

Een voorbeeld in de method *Main*

```
using (var entities = new OpleidingenEntities())
{
    var query = from campus in entities.Campussen
                orderby campus.Naam
                select new { campus.CampusNr, campus.Naam };
    foreach (var campusDeel in query)
    {
        Console.WriteLine("{0}: {1}", campusDeel.CampusNr, campusDeel.Naam);
    }
}
```

- (1) Je vraagt enkel de properties CampusNr en Naam, niet de overige properties. EF vertaalt dit naar een SQL statement dat ook enkel leest uit de kolommen CampusNr en Naam. Het resultaat van deze query is een verzameling objecten. Het type van deze objecten is een anonieme tijdelijke class, aangemaakt door EF. Deze class heeft twee properties: CampusNr en Naam.

Je kan ook deze query schrijven met query methods in plaats van een LINQ query.

Je probeert dit uit door de opdracht `var query` (over de volledige twee regels) te vervangen door:

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

5.4 Groeperen in queries

Je kan in een query objecten groeperen met de combinatie van de sleutelwoorden *group by* en *into*. Je vermeldt na *by* de entity property waarop je wil groeperen.

Een voorbeeld in de method *Main*. Je groepeerde de docenten op voornaam:

```
using (var entities = new OpleidingenEntities())
{
    var query = from docent in entities.Docenten
                group docent by docent.Voornaam into VoornaamGroep
                select new { VoornaamGroep, Voornaam = VoornaamGroep.Key };
    foreach (var voornaamStatistiek in query)
    {
        Console.WriteLine(voornaamStatistiek.Voornaam);
        Console.WriteLine(new string('-', voornaamStatistiek.Voornaam.Length));
        foreach (var docent in voornaamStatistiek.VoornaamGroep)
        {
            Console.WriteLine(docent.Naam);
        }
        Console.WriteLine();
    }
}
```

- (1) Het resultaat van deze query is een verzameling objecten. Deze hebben als type een anonieme tijdelijke class. Deze class heeft twee properties: Voornaam en VoornaamGroep. Voornaam is een voornaam die bij één of meerdere docenten voorkomt. VoornaamGroep is de verzameling Docent entities met deze voornaam.

Een voorbeeld van twee objecten uit het resultaat van de query:

Voornaam (String)	VoornaamGroup (een verzameling Docent entities)		
Armand	260	Armand	Van Bruaene
	24	Armand	Bayens
Arsène	32	Arsène	Bauwens

Je kan ook deze query schrijven met query methods in plaats van een LINQ query.

Je kan dit uitproberen door de opdracht `var` query (over de volledige drie regels) te vervangen door:

```
var query = entities.Docenten
    .GroupBy((docent) => docent.Voornaam,
        (Voornaam, docenten) => new { Voornaam, VoornaamGroep = docenten });
```

5.5 Lazy Loading

Je kan ná het uitvoeren van een query op een entity, die je van de query krijgt, een geassocieerde entity lezen uit de database. Dit heet lazy loading. Daarbij stuurt EF een nieuw SQL select statement naar de database, om de geassocieerde entitie(s) te lezen.

Een voorbeeld in de method *Main*. Je leest in de query zelf Docent entities.

Je leest pas ná de query de geassocieerde Campus entities:

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

- (1) Je leest in de query Docent entities.
- (2) Je spreekt het geassocieerde Campus object aan. EF stuurt op dat moment een SQL select statement naar de database om het juiste record uit de table Campussen te lezen. Je moet dit doen terwijl de DbContext nog niet gesloten is. Anders krijg je een exception.

Lazy loading kan een performantieprobleem veroorzaken.

Als je het programma uitvoert en Roger intikt, krijg je volgende output:

```
Roger Baens:Delos
Roger Baguet:Andros
Roger Blockx:Ikaria
Roger Decock:Andros
Roger De Vlaeminck:Delos
Roger Gyselinck:Delos
Roger Lambrecht:Gavdos
Roger Swerts:Delos
```

EF heeft 9 SQL select statements naar de database gestuurd:

- Één SQL statement dat de records met als voornaam *Roger* leest uit de table Docenten. Dit gebeurde bij het uitvoeren van de LINQ query.
- Acht SQL statements die elk één record lezen uit de table Campussen. Dit gebeurde in de foreach loop bij het lezen van docent.Campus.Naam.

Je kan het aantal SQL select statement terug brengen tot één met eager loading (zie hier onder)

5.6 Eager loading lost het performantieprobleem op

Je kan het aantal SQL select statement terug brengen tot één, door in de LINQ query niet enkel de Docent entities te lezen, maar ook al de geassocieerde Campus entities. Dit heet eager loading.

Je gebruikt daarvoor in de query de Include method:

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

(1)

- (1) De DbContext heeft properties waarop je LINQ queries uitvoert. In ons voorbeeld zijn dit de properties *Docenten* en *Campussen*. Deze properties bevatten een method Include. Je geeft aan deze method een String mee met de naam van een associatie die voorkomt in de bijbehorende entity class:
- Bij de DbContext property *Docenten* hoort de class *Docent*. Deze class bevat een associatie *Campus* (die verwijst naar de bijbehorende *Campus* entity).
 - Bij de DbContext property *Campussen* hoort de class *Campus*. Deze class bevat een associatie *Docenten* (die verwijst naar de bijbehorende *Docent* entities).

Bij het uitvoeren van de LINQ query, zal EF ook de records lezen die bij deze associatie horen. In onze code leest EF niet enkel records uit de table Docenten, maar ook de gerelateerde records uit de table Campussen. EF doet dit met één SQL select statement, met daarin het sleutelwoord JOIN.

Je maakt een tweede voorbeeld in de method *Main*.

Je leest in de query de Campus objecten waarvan in de naam een zoekwoord voorkomt.

Je leest in de query ook onmiddellijk de gerelateerde Docenten.

EF zal dit vertalen naar één SQL select statement.

```
using (var entities = new OpleidingenEntities())
{
    Console.WriteLine("Deel naam campus:");
    var deelNaam = Console.ReadLine();
    var query = from campus in entities.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.Naam);
        }
        Console.WriteLine();
    }
}
```

Je kan ook deze query schrijven met query methods in plaats van een LINQ query.

Je probeert dit uit door de opdracht `var query` (over de volledige drie regels) te vervangen door:

```
var query = entities.Campussen.Include("Docenten")
    .Where(campus => campus.Naam.Contains(deelNaam))
    .OrderBy(campus => campus.Naam);
```

Per programma onderdeel kan lazy loading of eager loading de beste oplossing zijn.

5.7 De method ToList van een query

Wanneer dat je met `foreach` itereert over een query, doet EF volgende stappen:

- De query omzetten naar een SQL select statement.
- Dit SQL select statement naar de database sturen.
- Bij iedere iteratie van jouw `foreach` een volgend record lezen uit het resultaat van dit SQL select statement.
- Dit record omzetten naar een entity.
- Jij kan deze entity binnen je `foreach` iteratie aanspreken.
- Op het einde van iedere `foreach` iteratie wordt deze entity vergeten.

Deze werkwijze heeft gevolgen:

- Als je twee keer itereert over een query, worden de voorgaande stappen twee keer allemaal uitgevoerd. Dit houdt in dat het SQL select statement ook twee keer uitgevoerd wordt. Je wil soms twee keer itereren over het resultaat van een query, zonder de query een tweede keer als SQL select statement uit te voeren (wegens bvb. performantieredenen).
- Je kan enkel itereren over een query binnen de `using` van de `DbContext` waarmee je query opbouwde, niet daarbuiten. Volgende code veroorzaakt een exception:

```
IQueryable<Campus> query;
using (var entities = new OpleidingenEntities())
{
    var query = from campus in entities.Campussen
                orderby campus.Naam
                select campus;
}
// Itereren na het sluiten van de DbContext (entities) kan niet
foreach (var campus in query)
{
}
```

Dit houdt in dat het niet mogelijk is de opbouw van de query in één method te schrijven en het itereren over de query in een andere method te schrijven.

De oplossing voor deze problemen is de method `ToList` van een query.

Als je deze method uitvoert, doet EF volgende stappen:

- De query omzetten naar een SQL select statement.
- Dit SQL select statement naar de database sturen.
- Itereren over de records uit het resultaat van het SQL select statement. Van ieder record een entity maken. Deze entity toevoegen aan een List.
- Op het einde van de method geeft de method `ToList` deze List terug.

Als je itereert over de List die je van de method `ToList` krijgt, itereer je over de entities in deze List (in het RAM geheugen).

Als je twee keer over deze List itereert, itereer je twee keer over entities in het RAM geheugen, en stuur je geen SQL select statement naar de database:

```
List<Campus> campussen;
using (var entities = new OpleidingenEntities())
{
    var query = from campus in entities.Campussen
                orderby campus.Naam
                select campus;
    campussen = query.ToList();
}
foreach (var campus in campussen)
{
    Console.WriteLine(campus.Naam);
}
Console.WriteLine();
foreach (var campus in campussen)
{
    Console.WriteLine(campus.Naam);
}
```

Je kan de query én de List opbouwen in één method en de List doorgeven aan een andere method, die itereert over de List:

```
static void Main(string[] args)
{
    Program program = new Program();
    foreach (var campus in program.FindAllCampussen())
    {
        Console.WriteLine(campus.Naam);
    }
}

List<Campus> FindAllCampussen()
{
    using (var entities = new OpleidingenEntities())
    {
        return (from campus in entities.Campussen
                orderby campus.Naam
                select campus).ToList();
    }
}
```



Opmerking: ook op queries die je definieert met query methods kan je de method ToList uitvoeren.



Klanten en hun rekeningen: zie takenbundel: zie takenbundel

6 ENTITIES TOEVOEGEN

6.1 Één entity toevoegen

Je doet volgende stappen om een entity toe te voegen aan de database:

- Je maakt de entity aan in het interne geheugen en je vult de properties van die entity.
- Je voegt deze entity toe aan verzameling gelijkaardige entiteiten in de DbContext.
Je doet dit met de method `Add` van deze verzameling. In ons voorbeeld bevatten de properties `Docenten` en `Campussen` van `OpleidingenEntities` een method `Add`.
- Je roept op de DbContext de method `SaveChanges` op. EF stuurt op dat moment een insert SQL statement naar de database om de entity als een record toe te voegen.

Voorbeeld in de method `Main`

```
var campus = new Campus{Naam = "Naam1", Straat = "Straat1", HuisNr = "1",
    PostCode = "1111", Gemeente = "Gemeente"};
using (var entities = new OpleidingenEntities())
{
    entities.Campussen.Add(campus);
    entities.SaveChanges();
    Console.WriteLine(campus.CampusNr);
}
```

- (1) Je hebt de entity toegevoegd aan de DbContext.
De entity is dan nog niet opgeslagen in de database.
- (2) Je slaat de entity op in de database.
- (3) EF vult na het toevoegen van een record het autonumber van dit nieuwe record automatisch in bij de property die hoort bij de autonumber kolom: `CampusNr`.

Je voert het programma uit.

Je ziet daarna in de Server Explorer een nieuw record in de table `campussen`:

CampusNr	Naam	Straat	HuisNr	PostCode	Gemeente
7	Naam1	Straat1	1	1111	Gemeente

6.2 Meerdere entities toevoegen

Als je meerdere entities met de `Add...` methods verbindt met de DbContext, moet je maar één keer de method `SaveChanges` uitvoeren om de bijbehorende SQL insert statements naar de database te sturen.

Voorbeeld in de method `Main`

```
var campus2 = new Campus {Naam = "Naam2", Straat = "Straat2", HuisNr = "2",
    PostCode = "2222", Gemeente = "Gemeente2"};
var campus3 = new Campus {Naam = "Naam3", Straat = "Straat3", HuisNr = "3",
    PostCode = "3333", Gemeente = "Gemeente3"};
using (var entities = new OpleidingenEntities())
{
    entities.Campussen.Add(campus2);
    entities.Campussen.Add(campus3);
    entities.SaveChanges();
}
```

Je voert het programma uit.

Je ziet daarna in de Server Explorer twee nieuwe records in de table `campussen`.

6.3 Entities met nieuwe geassocieerde entities toevoegen

Je kan in het interne geheugen een nieuwe entity én een nieuwe geassocieerde entity maken.

Het volstaat één van beide entities toe te voegen aan de DbContext met een `Add` method.

Als je daarna op de DbContext de method `SaveChanges` uitvoert, voegt EF twee records toe aan de database.

Voorbeeld 1 in de method *Main*: je maakt een nieuwe campus. Je maakt een nieuwe docent. Je associeert de docent met die campus vanuit het standpunt van de campus:

```
var campus4 = new Campus {Naam = "Naam4", Straat = "Straat4", HuisNr = "4",
    PostCode = "4444", Gemeente = "Gemeente4"};
var docent1 = new Docent {Voornaam = "Voornaam1", Familienaam = "Familienaam1",
    Wedde = 1};
// docent associëren met campus
//door hem toe te voegen aan de verzameling docenten van die campus
campus4.Docenten.Add(docent1);
using (var entities = new OpleidingenEntities())
{
    entities.Campussen.Add(campus4);
    entities.SaveChanges();
}
```

Je ziet na het uitvoeren in de Server Explorer een nieuw record in de table campussen én een nieuw geassocieerd record in de table docenten.

Voorbeeld 2 in de method *Main*: je maakt een nieuwe campus. Je maakt een nieuwe docent. Je associeert de docent met die campus vanuit het standpunt van de docent:

```
var campus5 = new Campus {Naam = "Naam5", Straat = "Straat5", HuisNr = "5",
    PostCode = "5555", Gemeente = "Gemeente5"};
var docent2 = new Docent {Voornaam = "Voornaam2", Familienaam = "Familienaam2",
    Wedde = 2};
// docent associëren met campus
// door de property Campus van de docent in te vullen:
docent2.Campus = campus5;
using (var entities = new OpleidingenEntities())
{
    entities.Docenten.Add(docent2);
    entities.SaveChanges();
}
```

Je ziet na het uitvoeren in de Server Explorer een nieuw record in de table campussen én een nieuw geassocieerd record in de table docenten.

6.4 Een entity toevoegen met een associatie naar een bestaande entity

6.4.1 Een entity toevoegen en de associatie definiëren vanuit de veel kant

Je zal als voorbeeld een nieuwe docent toevoegen en vanuit die docent (de veel kant van de associatie) een associatie leggen naar een bestaande campus.

Er bestaan hiertoe twee methodes.

- De geassocieerde entity lezen en associëren aan de nieuwe entity
- De geassocieerde entity associëren met de foreign key property

6.4.1.1 De geassocieerde entity lezen en associëren aan de nieuwe entity

Je doet bij deze methode stappen.

- Je maakt de nieuwe entity.
- Je leest de bestaande entity waarmee je de nieuwe entity wil associëren.
- Je associeert de bestaande entity met de nieuwe entity.
- Je voert op de object context de method *SaveChanges* uit.

Je moet deze stappen op dezelfde object context uitvoeren.

Voorbeeld in de method *Main*: je maakt een docent en associeert deze docent met de campus 1.

```
var docent3 = new Docent {Voornaam = "Voornaam3", Familiennaam = "Familiennaam3",
    Wedde = 3};
using (var entities = new OpleidingenEntities())
{
    var campus1 = entities.Campussen.Find(1);
    if (campus1 !=null)
    {
        entities.Docenten.Add(docent3);
        docent3.Campus = campus1;
        entities.SaveChanges();
    }
    else
    {
        Console.WriteLine("Campus 1 niet gevonden");
    }
}
```

6.4.1.2 De geassocieerde entity koppelen met de foreign key property

Je leest bij deze methode de geassocieerde entity niet, wat performantiewinst oplevert.

- Je vult de primary key van de geassocieerde entity in bij de property in de nieuwe entity die de foreign key naar de geassocieerde entity voorstelt:
 eenNieuweDocent.CampusNr = 1; // Campus 1 wordt de campus van de nieuwe docent

Voorbeeld in de method *Main*: je maakt een docent en associeert die met de bestaande campus 1:

```
var docent4 = new Docent {Voornaam = "Voornaam4", Familiennaam = "Familiennaam4",
    Wedde = 4, CampusNr = 1};
using (var entities = new OpleidingenEntities())
{
    entities.Docenten.Add(docent4);
    entities.SaveChanges();
}
```

6.4.2 Een entity toevoegen en de associatie definiëren vanuit de één kant

Je doet bij deze methode volgende stappen.

- Je maakt de nieuwe entity.
- Je leest de bestaande entity waarmee je de nieuwe entity wil associëren.
- Je voegt de nieuwe entity toe aan de associatie in de bestaande entity met de Add method.
- Je voert op de object context de method SaveChanges uit.

Voorbeeld in de method *Main*: je maakt een docent en associeert die met de bestaande campus 1:

```
var docent5 = new Docent {Voornaam="Voornaam5",Familiennaam="Familiennaam5",Wedde=5};
using (var entities = new OpleidingenEntities())
{
    var campus1 = entities.Campussen.Find(1);
    if (campus1 !=null)
    {
        campus1.Docenten.Add(docent5);
        entities.SaveChanges();
    }
    else
    {
        Console.WriteLine("Campus 1 niet gevonden");
    }
}
```



Zichtrekening toevoegen: zie takenbundel

7 ENTITIES WIJZIGEN

7.1 Één entity wijzigen

Je doet volgende stappen om een entity te wijzigen in de database:

- Je leest de entity vanuit de database.
- Je wijzigt deze entity in het interne geheugen.
- Je roept op de object context de method *SaveChanges* op. EF stuurt op dat moment een update SQL statement naar de database om het record dat bij de entity hoort te wijzigen.

Voorbeeld in de method *Main*: één docent opslag geven:

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

Je kan in de Server Explorer nazien of het juiste record in de table docenten aangepast is.

7.2 Meerdere entities lezen en slechts enkele daarvan wijzigen

Het kan gebeuren dat je met de DbContext meerdere entities leest en slechts enkele wijzigt.

Wanneer je op de DbContext de method *SaveChanges* uitvoert, stuurt EF enkel voor de aangepast entities SQL update statements naar de database.

Voorbeeld in de method *Main*: je leest de docenten met de nummers 1 en 2 uit de database.

Je wijzigt enkel de docent met het nummer 2. De method *SaveChanges* stuurt één SQL update statement naar de database om het record met DocentNr 2 te wijzigen:

```
using (var entities = new OpleidingenEntities())
{
    var docent1 = entities.Docenten.Find(1);
    var docent2 = entities.Docenten.Find(2);
    docent2.Opslag(10m);
    entities.SaveChanges();
}
```

7.3 Entities wijzigen die je indirect gelezen hebt met associaties

Soms lees je een entity en lees je met een Navigation Property van die entity een geassocieerde entity of een verzameling geassocieerde entities.

Je kan bijvoorbeeld een Campus entity lezen met een query. Als je de property Docenten van deze Campus entity aanspreekt, leest EF automatisch de Docent entities die bij de Campus entity horen.

Ook als je zo'n geassocieerde entities (in dit voorbeeld Docent entities) wijzigt en op de DbContext de method SaveChanges uitvoert, wijzigt EF de records die horen bij deze geassocieerde entities:

Voorbeeld in de method *Main*: je geeft 10 € opslag aan de docenten uit campus 1

```
using (var entities = new OpleidingenEntities())
{
    var campus1 = entities.Campussen.Find(1);
    if (campus1 != null)
    {
        foreach (var docent in campus1.Docenten)
        {
            docent.Opslag(10M);
        }
        entities.SaveChanges();
    }
}
```

Je kan in de Server Explorer nazien of de juiste records in de table docenten aangepast zijn.



Opmerking: EF stuurt in dit voorbeeld evenveel update SQL statements naar de database als er docenten behoren tot campus 1. Een snellere oplossing is vanuit EF een stored procedure op te roepen die alle docenten van campus 1 opslag geeft met één update SQL statement. Je ziet verder in de cursus hoe je een stored procedure oproept.

7.4 Een associatie van een entity wijzigen

7.4.1 De associatie wijzigen vanuit de veel kant

Je zal als voorbeeld een docent verhuizen naar een andere campus.

Er bestaan hiertoe twee methodes.

- De te associëren entity lezen en associëren aan de te wijzigen entity
- De te associëren entity associëren met de foreign key property

7.4.1.1 De te associëren entity lezen en associëren aan de te wijzigen entity

Je doet bij deze methode volgende stappen.

- Je leest de te wijzigen entity.
- Je leest de entity waarmee je de te wijzigen entity wil associëren.
- Je legt de associatie.
- Je voert op de object context de method SaveChanges uit.

Voorbeeld in de method *Main*: je verhuist docent 1 naar campus 6:

```
using (var entities = new OpleidingenEntities())
{
    var docent1 = entities.Docenten.Find(1);
    if (docent1 != null)
    {
        var campus6 = entities.Campussen.Find(6);
        if (campus6 != null)
        {
            docent1.Campus = campus6;
            entities.SaveChanges();
        }
        else
        {
            Console.WriteLine("Campus 6 niet gevonden");
        }
    }
}
```

```

    }
  }
  else
  {
    Console.WriteLine("Docent 1 niet gevonden");
  }
}

```

7.4.1.2 De te associëren entity associëren met de foreign key property

Je leest bij deze methode de te associëren entity niet, wat performantiewinst oplevert.

- Je vult de primary key van de te associëren entity in bij de property in de te wijzigen entity die de foreign key naar de te associëren entity voorstelt:
 teWijzigenDocent.CampusNr = 1; // Campus 1 wordt de campus gewijzigde docent

Voorbeeld in de method *Main*: je verhuist docent 1 naar campus 2:

```

using (var entities = new OpleidingenEntities())
{
    var docent1 = entities.Docenten.Find(1);
    if (docent1 != null)
    {
        docent1.CampusNr = 2;
        entities.SaveChanges();
    }
    else
    {
        Console.WriteLine("Docent 1 niet gevonden");
    }
}

```

7.4.2 De associatie wijzigen vanuit de één kant

Je zal als voorbeeld een docent verhuizen naar een andere campus.

Je doet hierbij volgende stappen.

- Je leest de te wijzigen entity.
- Je leest de entity waarmee je de eerste entity wil associëren.
- Je voegt de eerste entity toe aan de associatie in de tweede entity met de Add method.
- Je voert op de object context de method *SaveChanges* uit.

Voorbeeld in de method *Main*: je verhuist docent 1 naar campus 3:

```

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

```



Storten: zie takenbundel

8 ENTITIES VERWIJDEREN

Je doet volgende stappen om een entity te verwijderen uit de database:

- Je leest de entity vanuit de database.
- Je voert de Remove method uit van verzameling in de de DbContext die soortgelijke entities bevat. Je geeft de gelezen entity mee als parameter.
- Je roept op de DbContext de method *SaveChanges* op. EF stuurt dan een delete SQL statement naar de database om het record dat bij de entity hoort te verwijderen.
- Belangrijk:
Je moet deze stappen op dezelfde DbContext uitvoeren.

Voorbeeld inde method *Main*

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



Klant verwijderen: zie takenbundel

9 TRANSACTIES

9.1 Algemeen

Het doel van een transactie is meerdere SQL statements als één geheel te aanzien.

Het is de verantwoordelijkheid van de database er voor 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 (bvb. bij fout in de database, fout in jouw applicatie, stroomuitval), wat wil zeggen dat de aanpassingen van 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:

- Een statement die het te transfereren geld aftrekt van het saldo van de spaarrekening.
- Een statement die het te transfereren geld bijtelt bij het saldo van de zichtrekening.

Voorbeeld: om 10 € over te schrijven van spaarrekening 111-1111111-70 naar zichtrekening 222-2222222-43 heb je twee update statements nodig:

Eerste statement:

```
update Rekeningen
set saldo=saldo - 10
where RekeningNr = '111-1111111-70'
```

Tweede statement:

```
update Rekeningen
set saldo=saldo + 10
where RekeningNr = '222-2222222-43'
```

Fouten die kunnen optreden:

- Een fout in de databasesoftware
- Een fout in jouw applicatie
- Stroomuitval
- Het rekeningnummer van één van de rekeningen bestaat niet.
- Één van de records is te lang gelockt door een andere applicatie.

Als één van deze twee statements mislukt, mag het andere statement ook niet uitgevoerd zijn.

Door de SQL statements te verzamelen in een transactie, zorgt de database er voor dat ofwel beide SQL statements uitgevoerd worden, ofwel geen van beide.

Transacties hebben vier kenmerken (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 halverwege de transactie een fout gebeurt, brengt de database de bijgewerkte records terug in hun toestand juist voor de transactie begon.
- **Consistency**
De transactie breekt geen databaseregels. Als een kolom bijvoorbeeld geen duplicaten kan bevatten, zal de database de transactie afbreken (rollback) op het moment dat je toch probeert duplication 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 (locking).
- **Durability**
Een voltooide transactie is definitief vastgelegd in de database, zelfs al valt de computer uit juist na het voltooien van de transactie.

9.2 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:

↓ 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:



Read uncommitted → Read committed → Repeatable read → Serializable



Je moet dus per programma onderdeel analyseren welke problemen (dirty read, ...) de goede werking van dat programma onderdeel benadelen. Je kiest daarna een isolation level dat deze problemen oplost. Het isolation level read uncommitted wordt zelden gebruikt, omdat het geen enkel probleem oplost.

9.3 De method SaveChanges

Je leerde al de method SaveChanges van de DbContext kennen.

We herhalen nog eens wat deze method doet:

- Voor iedere entity die je aan de DbContext toegevoegd hebt (met de Add... methods) een insert SQL statement naar de database sturen.
- Voor iedere entity die je gelezen én gewijzigd hebt een update SQL statement naar de database sturen.
- Voor iedere entity die je verwijderd hebt ten opzichte van de object context een delete SQL statement naar de database sturen.

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`.

9.4 Eigen transactiebeheer met `TransactionScope`

9.4.1 Algemeen

Je kan soms de ingebouwde transacties van de method `SaveChanges` niet gebruiken.
Je moet dan zelf transactiebeheer doen. Voorbeelden waarom je zelf de transactie beheert:

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

Je doet eigen transactiebeheer met de class `TransactionScope`.

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

Je maakt een `TransactionScope` object binnen een `using` structuur.

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

Alle databasebewerkingen die je binnen deze `using` structuur uitvoert, behoren automatisch tot één en dezelfde transactie.

Nadat alle bewerkingen goed aflopen, voer je op je `TransactionScope` object de method `Complete` uit: `transactionScope.Complete();`

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 (bvb. een `exception` treedt op), gebeurt automatisch een `rollback` van alle bewerkingen van alle databaseconnecties die je opende binnen de `using` structuur.

`TransactionScope` objecten kunnen in elkaar genest zijn:

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

Er bestaan `TransactionScope` constructors waarbij je een parameter meegeeft van het type `TransactionScopeOption`. Je beslist met deze parameter hoe het `TransactionScope` object zich gedraagt ten opzichte van andere `TransactionScope` objecten waarbinnen het genest is.

De parameter kan volgende waarden bevatten:

- **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.
- **RequiresNew**
Het `TransactionScope` object start nieuwe transactie, zelfs als het object genest is in een ander `TransactionScope` object.
- **Suppress**
Alle databasebewerkingen binnen dit `TransactionScope` object behoren niet tot een transactie, zelfs als het `TransactionScope` object genest is in een ander `TransactionScope` object.

Om het isolation level van de transactie in te stellen bestaan er TransactionScope constructors die een parameter van het type TransactionOption aanvaarden. Een TransactionOption object heeft een IsolationLevel property om de isolation level in te stellen.

De class TransactionScope bevindt zich in de DLL System.Transactions.

Je legt in je project een referentie naar *System.Transactions.dll*:

- Je klikt in de Solution Explorer met de rechtermuisknop op je project.
- Je kiest *Add Reference*.
- Je plaatst in de lijst een vinkje bij *System.Transactions*
- Je klikt op *OK*.

9.4.2 Voorbeeld

Het script *VoorradenToevoegen.sql* voegt aan de database *Opleidingen* de table *Voorraden* toevoegt. Je voert dit script uit in de SQL Server Management Studio.

De nieuwe table heeft volgende structuur:

Voorraden			
	Column Name	Data Type	Allow Nulls
🔑	MagazijnNr	int	<input type="checkbox"/>
🔑	ArtikelNr	int	<input type="checkbox"/>
	AantalStuks	int	<input type="checkbox"/>
	RekNr	int	<input type="checkbox"/>

De table houdt bij hoeveel voorraad er per artikel per magazijn is.

De kolom RekNr geeft aan in welk rek van een magazijn een artikel te vinden is.

Je zal een hoeveelheid voorraad van één artikel overbrengen van één magazijn naar een ander magazijn. Je moet hierbij twee records aanpassen.

Je voegt aan *Opleidingen.edmx* de entity toe die hoort bij de table Voorraden:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
Je breidt met deze opdracht het entity data model uit met informatie over nieuwe tables, relaties, kolommen, views en stored procedures uit de database.
- Je klapt in het tabblad *Add* het onderdeel *Tables* en open.
- Je klapt daarbinnen *dbo* open.
- Je plaatst een vinkje bij *Voorraden*.
- Je kiest *Finish*.

Je corrigeert de naam van de entity *Voorraden* naar *Voorraad*

Voorraad	
Properties	
🔑	MagazijnNr : Int32
🔑	ArtikelNr : Int32
🔧	AantalStuks : Int32
🔧	RekNr : Int32
Navigation Properties	

Je doet in de eerste versie van het programma geen eigen transactiebeheer, maar je gebruikt het ingebouwde transactiebeheer van de method `SaveChanges`:

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("Artikel nr.");
        var artikelNr = 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());
        new Program().VoorraadTransfer(artikelNr, vanMagazijnNr, naarMagazijnNr,
aantalStuks);
    }
    catch (FormatException)
    {
        Console.WriteLine("Tik een getal");
    }
}

void VoorraadTransfer(int artikelNr, int vanMagazijnNr, int naarMagazijnNr,
int aantalStuks)
{
    using (var entities = new OpleidingenEntities())
    {
        var vanVoorraad = entities.Voorraden.Find(vanMagazijnNr, artikelNr);
        if (vanVoorraad != null)
        {
            if (vanVoorraad.AantalStuks >= aantalStuks)           Ⓣ1
            {
                vanVoorraad.AantalStuks -= aantalStuks;          Ⓣ2
                var naarVoorraad = entities.Voorraden.Find(naarMagazijnNr, artikelNr);
                if (naarVoorraad != null) // voorraad aanpassen
                {
                    naarVoorraad.AantalStuks += aantalStuks;
                }
                else // nieuwe voorraad aanmaken
                {
                    naarVoorraad = new Voorraad { ArtikelNr = artikelNr,
                        MagazijnNr = naarMagazijnNr, AantalStuks = aantalStuks };
                    entities.Voorraden.Add(naarVoorraad);
                }
                entities.SaveChanges();                             Ⓣ3
            }
            else
            {
                Console.WriteLine("Te weinig voorraad voor transfer");
            }
        }
        else
        {
            Console.WriteLine("Artikel niet gevonden in magazijn {0}", vanMagazijnNr);
        }
    }
}
```

Het programma doet de voorraadaanpassing correct, op voorwaarde dat geen andere gebruikers tegelijk aanpassingen doen. Als dit wel het geval is, kan volgende fout optreden:

- Ⓣ1 Je controleert hier of er genoeg voorraad ligt in het van magazijn.
- Ⓣ2 Je past hier de bijbehorende entity aan in het interne geheugen.
- Ⓣ3 EF past het bijbehorende record pas aan op het moment dat je `SaveChanges` uitvoert.

Tussendoor is het record niet gelockt. Een andere gebruiker kan dus tussendoor het aantal stuks van hetzelfde record verlagen. Zo kan uiteindelijk het aantal stuks negatief worden !

Je beheert in de 2° versie van het programma de transactie zelf, om dit probleem op te lossen.

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 dezelfde records niet wijzigen.

Je voegt boven in de source volgende opdracht toe:

```
using System.Transactions;
```

Enkel de method VoorraadTransfer is uitgebreid (onderstreepte regels):

```
void VoorraadTransfer(int artikelNr, int vanMagazijnNr, int naarMagazijnNr,
    int aantalStuks)
{
    var transactionOptions = new TransactionOptions
    {
        IsolationLevel = System.Transactions.IsolationLevel.RepeatableRead
    };
    using (var transactionScope = new TransactionScope(
        TransactionScopeOption.Required, transactionOptions))
    {
        using (var entities = new OpleidingenEntities())
        {
            var vanVoorraad = entities.Voorraden.Find(vanMagazijnNr, artikelNr);
            if (vanVoorraad != null)
            {
                if (vanVoorraad.AantalStuks >= aantalStuks)
                {
                    vanVoorraad.AantalStuks -= aantalStuks;
                    var naarVoorraad = entities.Voorraden.Find( naarMagazijnNr, artikelNr);
                    if (naarVoorraad != null) // voorraad aanpassen
                    {
                        naarVoorraad.AantalStuks += aantalStuks;
                    }
                    else // nieuwe voorraad aanmaken
                    {
                        naarVoorraad = new Voorraad { ArtikelNr = artikelNr,
                            MagazijnNr = naarMagazijnNr, AantalStuks = aantalStuks };
                        entities.Voorraden.Add(naarVoorraad);
                    }
                    entities.SaveChanges();
                    transactionScope.Complete();
                }
                else
                {
                    Console.WriteLine("Te weinig voorraad voor transfer");
                }
            }
            else
            {
                Console.WriteLine("Artikel niet gevonden in magazijn {0}", vanMagazijnNr);
            }
        }
    }
}
```



Overschrijven: zie takenbundel

10 OPTIMISTIC RECORD LOCKING

10.1 Algemeen

Je doet volgende stappen om een record te wijzigen met EF:

- 1) Je leest het te wijzigen record als een entity in het interne geheugen.
- 2) Je wijzigt de entity in het interne geheugen.
- 3) Je voert op de object context de method `SaveChanges` uit. Deze method stuurt een update SQL opdracht naar de database om het record te wijzigen dat bij de entity hoort.

Standaard doet EF geen controle of het record tussendoor niet door een andere gebruiker werd bijgewerkt. Dit houdt in dat je bij stap 3 de wijzigingen van die andere gebruiker overschrijft !

Je kan dit zien met volgend voorbeeld, waarin de voorraad bijgevoerd wordt:

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("Artikel nr.");
        var artikelNr = 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());
        new Program().VoorraadBijvulling(artikelNr, magazijnNr, aantalStuks);
    }
    catch (FormatException)
    {
        Console.WriteLine("Tik een getal");
    }
}

void VoorraadBijvulling(int artikelNr, int magazijnNr, int aantalStuks)
{
    using (var entities = new OpleidingenEntities())
    {
        var voorraad = entities.Voorraden.Find(magazijnNr, artikelNr);
        if (voorraad != null)
        {
            voorraad.AantalStuks += aantalStuks;
            Console.WriteLine("Pas nu de voorraad aan met de Server Explorer," +
                " druk daarna op Enter");
            Console.ReadLine();
            entities.SaveChanges();
        }
        else
        {
            Console.WriteLine("Voorraad niet gevonden");
        }
    }
}
```

Je voert het programma uit. Op het moment dat de boodschap *Pas nu de voorraad aan ...* verschijnt, schakel je over naar Visual Studio.

Je wijzigt in de Server Explorer de voorraad van hetzelfde record.

Je schakelt daarna terug naar de console applicatie en je drukt op Enter.

Na het uitvoeren van de console applicatie keer je terug naar de Server Explorer. Als het venster met de table *Voorraden* nog open staat geef je een rechtermuisklik in de table en je kiest *Refresh*. Je ziet dat de wijziging die je deed in de Server Explorer overschreven is door de console applicatie.

EF stuurt hier volgend update SQL statement naar de database:

```
update Voorraden
set AantalStuks = @0
where MagazijnNr = @1 and ArtikelNr = @2
```

EF vult volgende waarden in bij de parameters:

- @0 Het aantal stuks zoals aangepast in de bijbehorende entity
- @1 De waarde die de kolom *MagazijnNr* had bij het lezen van het record
- @2 De waarde die de kolom *ArtikelNr* had bij het lezen van het record

Een eerste oplossing voor dit probleem is zelf transactiebeheer te doen, en het isolation level *Repeatable read* te gebruiken. De console applicatie vergrendelt dan het record tot het einde van de transactie. Tussendoor kan niemand anders het record wijzigen.

Je ziet een tweede oplossing in dit hoofdstuk: optimistic record locking.

Bij optimistic record locking wordt het gelezen record niet echt gelockt. Bij het aanpassen van het record controleert EF 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 een exception.

Optimistic record locking is performant ten opzichte van een transactie met isolation level *Repeatable read* omdat er geen records gelockt worden.

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.

10.2 Table zonder timestamp kolom

Op dit moment heeft de table *Voorraden* geen timestamp kolom.

Je activeert optimistic record locking dan op de volgende manier:

- Je opent het ontwerp van de entiteiten (*OpleidingenEntities.edmx*).
- Je klikt alle properties in de entity *Voorraad* en je wijzigt in het properties venster de property *Concurrency Mode* naar *Fixed*.

Bij het lezen van het record, onthoudt EF de kolomwaarden van het gelezen record. Vb.:
MagazijnNr: 1, ArtikelNr: 10, AantalStuks: 100, RekNr: 3

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

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

EF vult volgende waarden in bij de parameters:

- @0 Het aantal stuks zoals aangepast in de bijbehorende entity
- @1 De waarde die de kolom *MagazijnNr* had bij het lezen van het record
- @2 De waarde die de kolom *ArtikelNr* had bij het lezen van het record
- @3 De waarde die de kolom *AantalStuks* had bij het lezen van het record
- @4 De waarde die de kolom *RekNr* had bij het lezen van het record

Als een andere applicatie ondertussen één van deze kolommen wijzigde bij hetzelfde record, vindt het update SQL statement geen record meer om aan te passen, omdat één of meerdere where voorwaarden false terug geven.

Zo detecteert EF dat een andere applicatie hetzelfde record wijzigde.

Dan werpt EF een *DbUpdateConcurrencyException*.

Je voegt boven in de source volgende opdracht toe:

```
using System.Data.Entity.Infrastructure;
```


Enkel de method VoorraadBijvulling is gewijzigd:

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

Je voert het programma uit. Op het moment dat de boodschap *Pas nu de voorraad aan ...* verschijnt, schakel je over naar Visual Studio. Je wijzigt in de Server Explorer de voorraad van hetzelfde record. Daarna schakel je terug naar de console applicatie en druk je op Enter.

10.3 Table met timestamp kolom

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.

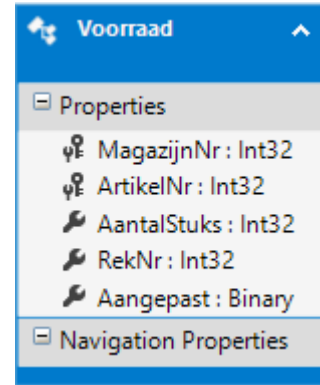
Je moet in het where deel van het update statement niet meer op iedere kolomwaarde controleren of het door een andere gebruiker werd bijgewerkt, maar enkel op deze timestamp kolom.

Het script
TimestampInVoorradenToevoegen.sql
voegt aan de table *Voorraden*
een *timestamp* kolom *Aangepast* toe.
Je voert dit script uit in de SQL Server
Management Studio.
De table heeft nu volgende structuur:

Voorraden			
	Column Name	Data Type	Allow Nulls
?	MagazijnNr	int	<input type="checkbox"/>
?	ArtikelNr	int	<input type="checkbox"/>
	AantalStuks	int	<input type="checkbox"/>
	RekNr	int	<input type="checkbox"/>
	Aangepast	timestamp	<input type="checkbox"/>

Je zorgt er voor dat in *Opleidingen.edmx* de entity *Voorraad* ook een extra property *Aangepast* heeft:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je kiest *Finish*.



De optimistic record locking moet nu enkel nog dit veld controleren:

- Je selecteert in de entity *Voorraad* de properties *MagazijnNr*, *ArtikelNr*, *AantalStuks* en *RekNr*, en je wijzigt in het properties venster de property *Concurrency Mode* terug naar *None*
- Je selecteert de property *Aangepast*, en je wijzigt in het properties venster de property *ConcurrencyMode* naar *Fixed*.

Als EF nu een record wijzigt, stuurt het volgend update SQL statement naar de database:

```
update Voorraden
set AantalStuks = @@
where MagazijnNr=@1 and ArtikelNr=@2 and Aangepast=@3
```

EF vult volgende waarden in bij de parameters:

- @0 Het aantal stuks zoals aangepast in de bijbehorende entity
- @1 De waarde die de kolom *MagazijnNr* had bij het lezen van het record
- @2 De waarde die de kolom *ArtikelNr* had bij het lezen van het record
- @3 De waarde die de kolom *Aangepast* had bij het lezen van het record

Als er nog kolommen bijkomen in de table *Voorraden*, blijft het where deel van het update SQL statement zoals nu: kort en daarom ook performant.



Klant wijzigen: zie takenbundel

11 ASSOCIATIES

11.1 Algemeen

Je leerde reeds werken met een één op veel associatie (tussen campus en docenten).
Je leert in dit hoofdstuk andere types associaties kennen.

11.2 Veel op veel associaties zonder extra associatieinformatie

11.2.1 De tables Boeken, Cursussen en BoekenCursussen toevoegen aan de database

Het script *BoekenEnCursussenToevoegen.sql* voegt aan de database *Opleidingen* de tables *Boeken*, *Cursussen* en *BoekenCursussen* toe. Je voert dit script uit in de SQL Server Management Studio.

De nieuwe tables hebben volgende structuur:

Boeken			BoekenCursussen			Cursussen		
Column Name	Data Type	Allow Nulls	Column Name	Data Type	Allow Nulls	Column Name	Data Type	Allow Nulls
BoekNr	int	<input type="checkbox"/>	BoekNr	int	<input type="checkbox"/>	CursusNr	int	<input type="checkbox"/>
ISBNNr	nvarchar(17)	<input type="checkbox"/>	CursusNr	int	<input type="checkbox"/>	Naam	nvarchar(50)	<input type="checkbox"/>
Titel	nvarchar(50)	<input type="checkbox"/>						

Er is een veel op veel relatie tussen *Boeken* en *Cursussen*:

- Één boek kan in meerdere cursussen gebruikt worden
- In één cursus kunnen meerdere boeken gebruikt worden.

Twee tables in een database kunnen geen directe veel op veel relatie hebben.

De oplossing is een tussentable (*BoekenCursussen*) die een veel op één relatie heeft naar *Boeken* én een veel op één relatie naar *Cursussen*.

De tussentable bevat enkel de kolommen die de associatie definiëren:

BoekNr en *CursusNr*. Er zijn geen kolommen met extra associatieinformatie.

11.2.2 De entities definiëren die bij de tables Boeken en Cursussen horen

Je voegt aan *Opleidingen.edmx* entities toe die horen bij de tables *Boeken* en *Cursussen*:

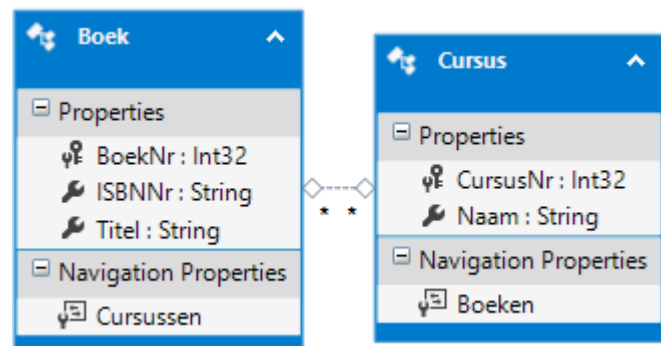
- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Tables* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *Boeken*, *BoekenCursussen* en *Cursussen* en je kiest *Finish*.

Je ziet de entities *Boeken* en *Cursussen* met een veel op veel associatie (* *).

Er is geen tussenentity nodig, omdat er geen associatieinformatie is.

Je corrigeert enkele namen:

- Je wijzigt de naam van de entity *Boeken* naar *Boek*
- Je wijzigt de naam van de entity *Cursussen* naar *Cursus*



- De navigation property *Cursussen* in de entity *Boek* verwijst vanuit een boek naar de cursussen waarin dit boek gebruikt wordt.
- De navigation property *Boeken* in de entity *Cursus* verwijst vanuit een cursus naar de boeken die in die cursus gebruikt worden.

11.2.3 De entities gebruiken vanuit je code

Je kan deze nieuwe entities nu gebruiken vanuit je code (Program.cs)

Voorbeeld 1 in de method *Main*: je toont elk boek en de cursussen waarin dit boek gebruikt wordt

```
using (var entities = new OpleidingenEntities())
{
    var query = from boek in entities.Boeken.Include("Cursussen")
                orderby boek.Titel
                select boek;
    foreach (var boek in query)
    {
        Console.WriteLine(boek.Titel);
        foreach (var cursus in boek.Cursussen)
        {
            Console.WriteLine("\t{0}", cursus.Naam);
        }
    }
}
```

Voorbeeld 2 in de method *Main*: je toont elke cursus en de daarin gebruikte boeken

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen.Include("Boeken")
                orderby cursus.Naam
                select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine(cursus.Naam);
        foreach (var boek in cursus.Boeken)
        {
            Console.WriteLine("\t{0}", boek.Titel);
        }
    }
}
```

Voorbeeld 3 in de method *Main*: je maakt een nieuw boek en associeert dit met de cursus Oracle:

```
using (var entities = new OpleidingenEntities())
{
    var nieuwBoek = new Boek {ISBNNr = "0-0788210-6-1",
                              Titel = "Oracle Backup & Recovery Handbook"};
    var oracleCursus = (from cursus in entities.Cursussen
                        where cursus.Naam == "Oracle"
                        select cursus).FirstOrDefault();
    if (oracleCursus != null)
    {
        oracleCursus.Boeken.Add(nieuwBoek);
        entities.SaveChanges();
    }
    else
    {
        Console.WriteLine("cursus Oracle niet gevonden");
    }
}
```

11.3 Veel op veel associaties met extra associatieinformatie

11.3.1 De tables Boeken2, Cursussen2 en BoekenCursussen2 toevoegen aan de database.

Je leert hier hoe EF werkt als de tussentable in de veel op veel relaties extra associatieinformatie bevat.

Het script *BoekenEnCursussenMetAssociatieInformatieToevoegen.sql* voegt aan de database *Opleidingen* de tables *Boeken2*, *Cursussen2* en *BoekenCursussen2* toe.

Je voert dit script uit in de SQL Server Management Studio.

De nieuwe tables hebben volgende structuur:

Boeken2			BoekenCursussen2			Cursussen2		
Column Name	Data Type	Allow Nulls	Column Name	Data Type	Allow Nulls	Column Name	Data Type	Allow Nulls
BoekNr	int	<input type="checkbox"/>	BoekNr	int	<input type="checkbox"/>	CursusNr	int	<input type="checkbox"/>
ISBNNr	nvarchar(17)	<input type="checkbox"/>	CursusNr	int	<input type="checkbox"/>	Naam	nvarchar(50)	<input type="checkbox"/>
Titel	nvarchar(50)	<input type="checkbox"/>	VolgNr	int	<input type="checkbox"/>			

De tussentable *BoekenCursussen2* bevat een extra kolom *VolgNr*.

Deze kolom geeft aan in welke volgorde boeken gebruikt worden in één cursus.

Je ziet deze informatie met volgend SQL statement:

```
select Cursussen2.Naam, BoekenCursussen2.VolgNr, Boeken2.Titel
from Cursussen2
inner join BoekenCursussen2 on Cursussen2.CursusNr = BoekenCursussen2.CursusNr
inner join Boeken2 on Boeken2.BoekNr = BoekenCursussen2.BoekNr
order by Cursussen2.Naam, BoekenCursussen2.VolgNr
```

Het resultaat:

Naam	VolgNr	Titel
Access	1	Relational Database Systems
Access	2	Access from the Ground Up
C++	1	C++ : The Core Language
C++	2	C++ : The Complete Reference
C++	3	C++ : For Scientists and Engineers
Oracle	1	Relational Database Systems
Oracle	2	Oracle : A Beginner's Guide
Oracle	3	Oracle : The Complete Reference



Opmerking: *BoekenCursussen2* bevat een samengestelde index op *CursusNr* en *VolgNr* die unieke waarden vereist. Zo kan een cursus geen twee keer hetzelfde volgnr. hebben.

11.3.2 De entities definiëren die bij de tables *Boeken2*, *Cursussen2* en *BoekenCursussen2* horen

Je verwijdert eerst in *Opleidingen.edmx* de entities *Boek* en *Cursus*, want ze krijgen hier een andere opbouw. Je kan een entity verwijderen met een rechtermuisklik op de entity en de opdracht *Delete from Model*.

Je voegt aan *Opleidingen.edmx* entities toe die horen bij de tables *Boeken2*, *Cursussen2* en *BoekenCursussen2*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Tables* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *Boeken2*, *BoekenCursussen2* en *Cursussen2*
- Je kiest *Finish*.

Je ziet de entities *Boeken2*, *Cursussen2* én een associatieentity *BoekenCursussen2*.

De associatieentity bevat de associatieinformatie *VolgNr*.

Er is een één op veel associatie tussen *Boeken2* en *BoekenCursussen2*

én een één op veel associatie tussen *Cursussen2* en *BoekenCursussen2*.

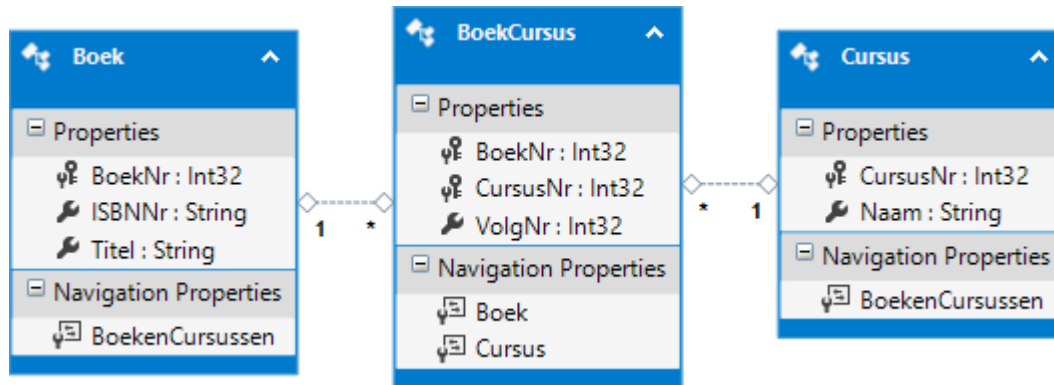
Deze twee associaties samen vormen een veel op veel associatie tussen *Boeken2* en *Cursussen2*.

Je corrigeert enkele namen:

- Je wijzigt de naam van de entity *Boeken2* naar *Boek*
- Je wijzigt in het properties venster de *Entity Set Name* naar *Boeken*
- Je wijzigt de naam van de navigation property *BoekenCursussen2* naar *BoekenCursussen*.
- Je wijzigt de naam van de entity *Cursussen2* naar *Cursus*

- Je wijzigt in het properties venster de *Entity Set Name* naar *Cursussen*
- Je wijzigt de naam van de navigation property *BoekenCursussen2* naar *BoekenCursussen*.
- Je wijzigt de naam van de entity *BoekenCursussen2* naar *BoekCursus*
- Je wijzigt in het properties venster de *Entity Set Name* naar *BoekenCursussen*
- Je wijzigt de naam van de navigation property *Boeken2* naar *Boek*.
- Je wijzigt de naam van de navigation property *Cursussen2* naar *Cursus*.

Het eindresultaat:



- De navigation property *BoekenCursussen* in de entity *Boek* laat toe vanuit een boek te verwijzen naar de associatieinformatie *BoekCursus*. Hier vind je het volgnr. van iedere cursus die dit boek gebruikt. Om andere informatie van deze cursus op te halen gebruik je de navigation property *Cursus* van *BoekCursus*.
- De navigation property *BoekenCursussen* in de entity *Cursus* laat toe vanuit een cursus te verwijzen naar de associatieinformatie *BoekCursus*. Hier vind je het volgnr. van ieder boek die in de cursus gebruikt wordt. Om andere informatie van dit boek op te halen gebruik je de navigation property *Boek* van *BoekCursus*.

11.3.3 De entities gebruiken vanuit je code

Je kan deze nieuwe entities nu gebruiken vanuit je code (Program.cs)

Voorbeeld 1 in de method *Main*: je toont elke cursus en de daarin gebruikte boeken

```
using (var entities = new OpleidingenEntities())
{
    var query =
        from cursus in entities.Cursussen.Include("BoekenCursussen.Boek")
        orderby cursus.Naam
        select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine(cursus.Naam);
        foreach (var boekCursus in cursus.BoekenCursussen)
        {
            Console.WriteLine("\t{0}:{1}", boekCursus.VolgNr, boekCursus.Boek.Titel);
        }
    }
}
```

- (1) *BoekenCursussen.Boek* betekent: Laadt van iedere cursus de gerelateerde informatie van de navigation property *BoekenCursussen* én van de gerelateerde informatie nog eens gerelateerde data van de navigation property *Boek*.

Je ziet volgende informatie:

Access

- 1:Relational Database Systems
- 2:Access from the Ground Up

C++

- 3:C++ : For Scientists and Engineers
- 2:C++ : The Complete Reference
- 1:C++ : The Core Language

Oracle

- 1:Relational Database Systems
- 2:Oracle : A Beginner's Guide
- 3:Oracle : The Complete Reference

Voorbeeld 2 in de method *Main*: je voegt een boek toe

```
var nieuwBoek = new Boek() { ISBNNr = "0-201-70431-5", Titel = "Modern C++ Design" };
var transactionOptions = new System.Transactions.TransactionOptions
{ IsolationLevel = System.Transactions.IsolationLevel.Serializable };
using (var transactionScope = new System.Transactions.TransactionScope(
    System.Transactions.TransactionScopeOption.Required, transactionOptions))
{
    using (var entities = new OpleidingenEntities())
    {
        // Cursus C++ ophalen
        // én het hoogste volgnr. van boek gebruikt in die cursus.
        // Met transactie met isolation level Serializable
        // kan daarna niemand anders een boek toevoegen aan C++ cursus
        // en is het nieuwe volgnr gelijk aan 1 + hoogst gelezen volgnr
        var query = from cursus in entities.Cursussen.Include("BoekenCursussen")
                     where cursus.Naam == "C++"
                     select new { Cursus = cursus,
                                   HoogsteVolgnr = cursus.BoekenCursussen.Max(
                                       boekCursus => boekCursus.Volgnr) };
        var queryResult = query.FirstOrDefault();
        if (queryResult != null)
        {
            entities.BoekenCursussen.Add(new BoekCursus { Boek = nieuwBoek,
                                                            Cursus = queryResult.Cursus, Volgnr = queryResult.HoogsteVolgnr + 1 });
            entities.SaveChanges();
        }
        transactionScope.Complete();
    }
}
```

11.4 Een associatie van een entity class naar zichzelf

11.4.1 Algemeen


Een table in een relationele database kan een foreign key kolom hebben die verwijst naar de primary key van dezelfde table. De bijbehorende entity class heeft dan een navigation property die verwijst naar dezelfde entity class.

11.4.2 De table Cursisten toevoegen aan de database.

Het script *CursistenToevoegen.sql* voegt aan de database *Opleidingen* de table *Cursisten* toe.

Je voert dit script uit in de SQL Server Management Studio.

De nieuwe table heeft volgende structuur:



	Column Name	Data Type	Allow Nulls
Key	CursistNr	int	<input type="checkbox"/>
	Voornaam	nvarchar(50)	<input type="checkbox"/>
	Familienaam	nvarchar(50)	<input type="checkbox"/>
	MentorNr	int	<input checked="" type="checkbox"/>

Een cursist kan een andere cursist als mentor hebben. De kolom *MentorNr* bevat dan het *CursistNr* van de mentor. Een mentor kan meerdere cursisten hebben.

11.4.3 De entity definiëren die bij de table Cursisten hoort

Je voegt aan *Opleidingen.edmx* een entity toe die hoort bij de table *Cursisten*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Tables* open.
- Je klapt daarbinnen *dbo* open.
- Je plaatst een vinkje bij *Cursisten*
- Je kiest *Finish*.

Je ziet een nieuwe entity *Cursisten*

Je ziet in deze entity twee navigation properties: *Cursisten1* en *Cursisten2*.

Aan de hand van deze property namen kan je de betekenis niet inschatten.

Als je *Cursisten1* aanklikt, zie je in het properties venster bij *Multiplicity * (Many)*

Hieruit kan je afleiden dat de navigation property *Cursisten1* de cursisten voorstelt die bij de mentor horen (de beschermelingen van de mentor).

Je hernoemt de navigation property *Cursisten1* dus naar *Beschermelingen*

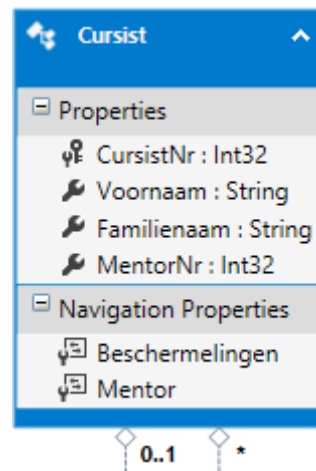
Als je *Cursisten2* aanklikt, zie je in het properties venster bij *Multiplicity 0..1 (Zero or One)*

Je kan hieruit afleiden dat de navigation property *Cursisten2* de mentor voorstelt van een cursist (als die een mentor heeft).

Je hernoemt de navigation property *Cursisten2* dus naar *Mentor*

Je wijzigt de naam van de entity *Cursisten* naar *Cursist*.

Het eindresultaat:



11.4.4 De entities gebruiken vanuit je code

Je kan deze nieuwe entities nu gebruiken vanuit je code (Program.cs)

Voorbeeld 1 in de method *Main*: je toont de cursisten die nog geen mentor hebben:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursist in entities.Cursisten
                where cursist.Mentor == null
                orderby cursist.Voornaam, cursist.Familienaam
                select cursist;
    foreach (var cursist in query)
    {
        Console.WriteLine("{0} {1}", cursist.Voornaam, cursist.Familienaam);
    }
}
```


Voorbeeld 2 in de method *Main*: je toont de cursisten die wel een mentor hebben én hun mentor:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursist in entities.Cursisten.Include("Mentor")
                where cursist.Mentor != null
                orderby cursist.Voornaam, cursist.Familienaam select cursist;
    foreach (var cursist in query)
    {
        var mentor = cursist.Mentor;
        Console.WriteLine("{0} {1}: {2} {3}", cursist.Voornaam, cursist.Familienaam,
            mentor.Voornaam, mentor.Familienaam);
    }
}
```

Voorbeeld 3 in de method *Main*: je toont cursisten die mentor zijn en hun beschermelingen:

```
using (var entities = new OpleidingenEntities())
{
    var query = from mentor in entities.Cursisten.Include("Beschermelingen")
                where mentor.Beschermelingen.Count != 0
                orderby mentor.Voornaam, mentor.Familienaam select mentor;
    foreach (var mentor in query)
    {
        Console.WriteLine("{0} {1}", mentor.Voornaam, mentor.Familienaam);
        foreach (var beschermeling in mentor.Beschermelingen)
        {
            Console.WriteLine("\t{0} {1}", beschermeling.Voornaam, beschermeling.Familienaam);
        }
    }
}
```

Voorbeeld 4 in de method *Main*: cursist 5 wordt de mentor van cursist 6:

```
using (var entities = new OpleidingenEntities())
{
    var cursist5 = entities.Cursisten.Find(5);
    if (cursist5 != null)
    {
        var cursist6 = entities.Cursisten.Find(6);
        if (cursist6 != null)
        {
            cursist5.Beschermelingen.Add(cursist6);
            entities.SaveChanges();
        }
        else
        {
            Console.WriteLine("Cursist 6 niet gevonden");
        }
    }
    else
    {
        Console.WriteLine("Cursist 5 niet gevonden");
    }
}
```

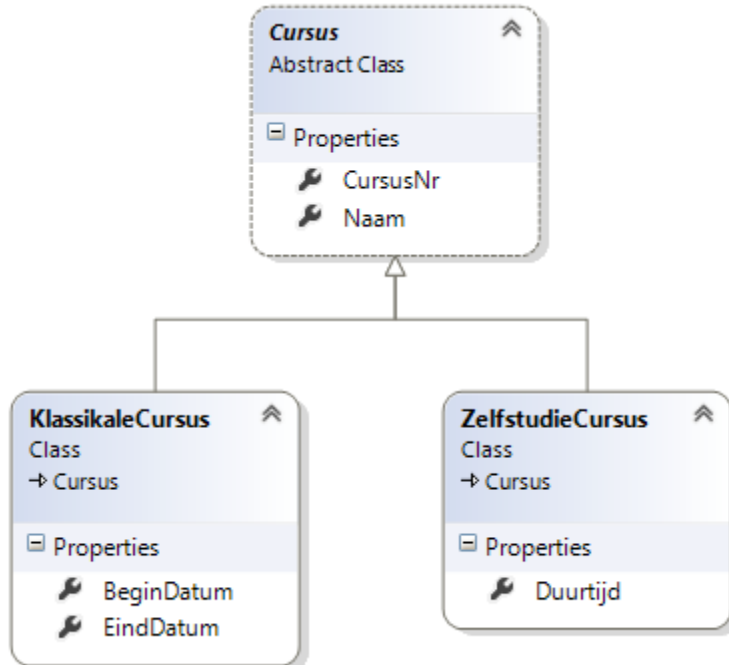


Personeel: zie takenbundel

12 INHERITANCE

12.1 Algemeen

Inheritance bestaat in C# maar niet in een database. Je kan inheritance nabootsen in de database, op drie manieren. Je leert in dit hoofdstuk hoe EF daarmee omgaat. De voorbeeld entiteiten:



12.2 Table per concrete class (TPC)

12.2.1 Algemeen

Table per concrete class is de eerste manier om inheritance na te bootsen in de database.

- Er is één table per concrete class (*KlassikaleCursus*, *ZelfstudieCursus*).
- Er is geen table voor abstracte classes (*Cursus*).

12.2.2 De tables *KlassikaleCursussen* en *ZelfstudieCursussen* toevoegen aan de database

Het script *TablePerConcreteClass.sql* voegt aan de database *Opleidingen* de tables *KlassikaleCursussen* en *ZelfstudieCursussen* toe. Je voert dit script uit in de SQL Server Management Studio.

Deze tables hebben volgende structuur:

KlassikaleCursussen			
	Column Name	Data Type	Allow Nulls
🔑	CursusNr	int	<input type="checkbox"/>
	Naam	varchar(50)	<input type="checkbox"/>
	Van	datetime	<input type="checkbox"/>
	Tot	datetime	<input type="checkbox"/>

ZelfstudieCursussen			
	Column Name	Data Type	Allow Nulls
🔑	CursusNr	int	<input type="checkbox"/>
	Naam	varchar(50)	<input type="checkbox"/>
	Duurtijd	int	<input type="checkbox"/>

Bij Table per concrete class mogen de entities over de bijbehorende tables heen geen dubbele primary key waarden hebben. In ons voorbeeld mag een record uit de table *KlassikaleCursussen* niet eenzelfde primary key waarde hebben als een record uit de table *ZelfstudieCursussen*. Anders krijg je runtime fouten als je straks de records aanspreekt vanuit EF.

De oplossing in ons voorbeeld is als volgt:

- De primary key van de table *KlassikaleCursussen* is gebaseerd op de kolom *CursusNr*. Dit is een autonummer kolom. De nummering begint vanaf 1 en verhoogt per nieuw record met 2. De nummering van de records is dus: 1, 3, 5, 7, ...
- De primary key van de table *ZelfstudieCursussen* is ook gebaseerd op de kolom *CursusNr*. Dit is een autonummer kolom. De nummering begint vanaf 2 en verhoogt per nieuw record met 2. De nummering van de records is dus: 2, 4, 6, 8, ...

Je kan dit inzien vanuit Visual Studio:

- Je klapt in de server explorer de databaseconnectie naar de database *Opleidingen* open.
- Je klapt daarbinnen het onderdeel *Tables* open.
- Je klapt daarbinnen de table *KlassikaleCursussen* open.
- Je selecteert de kolom *CursusNr*.
- Je ziet in het Properties venster bij *Identity Seed 1* (begin nummering).
- Je ziet bij *Identity Increment 2* (verhoging van de nummering).

12.2.3 De entities definiëren die horen bij de tables *KlassikaleCursussen* en *ZelfstudieCursussen*

Je verwijdert eerst in *Opleidingen.edmx* de entities *Boek*, *Cursus* en *BoekCursus*, want de entity *Cursus* krijgt hier een andere opbouw.

Je voede entities toe die horen bij de tables *KlassikaleCursussen* en *ZelfstudieCursussen*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Tables* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *KlassikaleCursussen* en *ZelfstudieCursussen* en je kiest *Finish*.

Je ziet de entities *KlassikaleCursussen* en *ZelfstudieCursussen*

Je corrigeert enkele namen:

- Je wijzigt de naam van de entity *KlassikaleCursussen* naar *KlassikaleCursus*
- Je wijzigt de naam van de entity *ZelfstudieCursussen* naar *ZelfstudieCursus*

12.2.4 De entity *Cursus* toevoegen en inheritance definiëren

Op dit moment is er nog geen inheritance verband tussen *KlassikaleCursus* en *ZelfstudieCursus*.

Je maakt nu dit verband:

- Je maakt een nieuwe entity *Cursus*:
 - Je sleept vanuit de toolbox een *Entity* op de designer.
 - Je wijzigt de naam van *Entity1* naar *Cursus*
 - Je wijzigt in het properties venster *Entity Set Name* naar *Cursussen*
 - Je wijzigt in het properties venster *Abstract* naar *True*. Nu is *Cursus* abstract.
 - Je verwijdert in de entity *Cursus* de property *Id*
- Je brengt de gemeenschappelijke properties *CursusNr* en *Naam* van *KlassikaleCursus* en *ZelfstudieCursus* over naar *Cursus*:
 - Je selecteer de properties *CursusNr* en *Naam* in *KlassikaleCursus*. Je klikt met de rechtermuisknop in deze selectie en je kiest *Cut*
 - Je selecteert *Cursus* met de rechtermuisknop en je kiest *Paste*.
 - Je verwijdert de properties *CursusNr* en *Naam* in *ZelfstudieCursus*
- Je definieert de inheritance tussen *KlassikaleCursus* en *Cursus* en tussen *ZelfstudieCursus* en *Cursus*:
 - Je kiest in de toolbox een *Inheritance* en sleept van *KlassikaleCursus* naar *Cursus*.
 - Je kiest in de toolbox een *Inheritance* en sleept van *ZelfstudieCursus* naar *Cursus*.

Je moet nog definiëren welke kolommen (uit de tables van de database) horen bij de properties *CursusNr* en *Naam* van de entity *Cursus*.

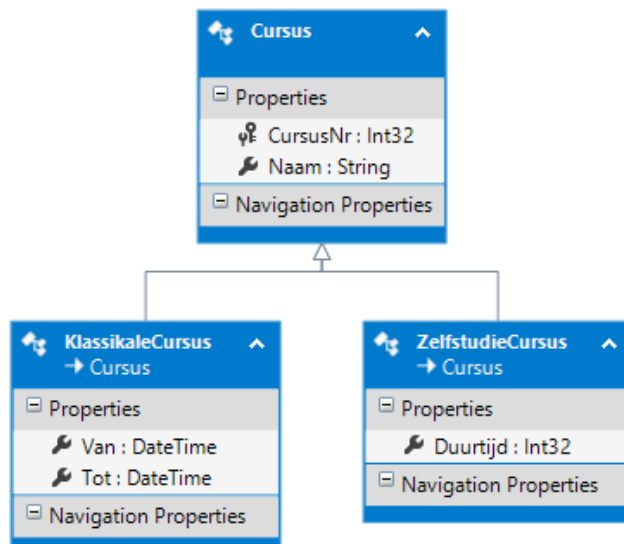
Je kan dit niet definiëren in de designer, wel in de achterliggende XML:

- Je sluit het venster met de designer van *Opleidingen.edmx* en je slaat daarbij op.
- Je klikt in de Solution Explorer met de rechtermuisknop op *Opleidingen.edmx*.
- Je kiest *Open With*
- Je kiest *XML (Text) Editor* en je kiest *OK*.
- De XML bevat een element *edmx:Runtime*
- Dit element bevat een element *edmx:Mappings*
In dit element worden onder andere properties aan kolommen gekoppeld.
- Dit element bevat een element *EntitySetMapping Name="Cursussen"*
- Je wijzigt dit element als volgt:

```
<EntitySetMapping Name="Cursussen">
  <EntityTypeMapping
    TypeName="IsTypeOf(OpleidingenModel.KlassikaleCursus)">
    <MappingFragment StoreEntitySet="KlassikaleCursussen">
      <ScalarProperty Name="CursusNr" ColumnName="CursusNr" />      (1)
      <ScalarProperty Name="Naam" ColumnName="Naam" />              (2)
      <ScalarProperty Name="Tot" ColumnName="Tot" />
      <ScalarProperty Name="Van" ColumnName="Van" />
    </MappingFragment>
    </EntityTypeMapping>
  <EntityTypeMapping
    TypeName="IsTypeOf(OpleidingenModel.ZelfstudieCursus)">
    <MappingFragment StoreEntitySet="ZelfstudieCursussen">
      <ScalarProperty Name="CursusNr" ColumnName="CursusNr" />      (3)
      <ScalarProperty Name="Naam" ColumnName="Naam" />              (4)
      <ScalarProperty Name="Duurtijd" ColumnName="Duurtijd" />
    </MappingFragment>
    </EntityTypeMapping>
</EntitySetMapping>
```

- (1) Je geeft op deze extra lijn aan dat de property *CursusNr* van de entity *KlassikaleCursus* hoort bij de kolom *CursusNr* van de table *KlassikaleCursussen*.
- (2) Je geeft op deze extra lijn aan dat de property *Naam* van de entity *KlassikaleCursus* hoort bij de kolom *Naam* van de table *KlassikaleCursussen*.
- (3) Je geeft op deze extra lijn aan dat de property *CursusNr* van de entity *ZelfstudieCursus* hoort bij de kolom *CursusNr* van de table *ZelfstudieCursussen*.
- (4) Je geeft op deze extra lijn aan dat de property *Naam* van de entity *ZelfstudieCursus* hoort bij de kolom *Naam* van de table *ZelfstudieCursussen*.

Je sluit de XML visie van *Opleidingen.edmx*. In de design view zien de entites er als volgt uit:



Als je de entity *KlassikaleCursus* of de entity *ZelfstudieCursus* selecteert, zie je in het properties venster dat de *Entity Set Name* property de waarde *Cursussen* bevat. Je kan dit niet wijzigen. De property *Entity Set Name* van derived entities (*KlassikaleCursus*, *ZelfstudieCursus*) is altijd gelijk aan de *Entity Set Name* property van hun base entity (*Cursus*).

Een instance van de object context heeft dus een property *Cursussen*, maar geen property *KlassikaleCursussen* of *ZelfstudieCursussen*.

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen select cursus;
var query = from cursus in entities.KlassikaleCursussen select cursus;
}
```

Je kan dus enkel een LINQ query uitvoeren op de property *Cursussen*.

Als je enkel de klassikale cursussen wil lezen, geef je dit aan in het *where* deel van je query:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen
                where cursus is KlassikaleCursus
                select cursus;
}
```

12.2.5 De entities gebruiken vanuit je code

Voorbeeld 1 in de method *Main*: je toont de naam van alle cursussen:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen
                orderby cursus.Naam
                select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine(cursus.Naam);
    }
}
```

Je toont in dit voorbeeld de naam van iedere cursus, maar niet of het om een klassikale cursus of een zelfstudiecursus gaat.

Je kan aan een object de naam van de class vragen waartoe het object behoort. Je voert daartoe op dat object de method *GetType()* uit. Deze method geeft je een object van de class *Type*. Je vraagt aan dit object de property *Name*. Je wijzigt de regel *Console.WriteLine(... naar*

```
Console.WriteLine(cursus.Naam + ' ' + cursus.GetType().Name);
```

Voorbeeld 2 in de method *Main*: je toont enkel klassikale cursussen:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen
                where cursus is KlassikaleCursus
                orderby cursus.Naam
                select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine(cursus.Naam);
    }
}
```

Voorbeeld 3 in de method *Main*: je voegt een zelfstudie cursus toe:

```
using (var entities = new OpleidingenEntities())
{
    entities.Cursussen.Add(new ZelfstudieCursus
    {
        Naam = "Spaanse correspondentie",
        Duurtijd = 6
    });
    entities.SaveChanges();
}
```


12.3 Table per hierarchy (TPH)

Dit is de tweede manier om inheritance na te bootsen in de database.

Hierbij bevat de database één table voor alle classes uit de inheritance hiërarchy.

Het script *TablePerClassHierarchy.sql* voegt aan de database *Opleidingen* de table *Cursussen3* toe.

Je voert dit script uit in de SQL Server Management Studio.

Cursussen3			
	Column Name	Data Type	Allow Nulls
	CursusNr	int	<input type="checkbox"/>
	Naam	varchar(5...	<input type="checkbox"/>
	Van	datetime	<input checked="" type="checkbox"/>
	Tot	datetime	<input checked="" type="checkbox"/>
	Duurtijd	int	<input checked="" type="checkbox"/>
	SoortCursus	char(1)	<input type="checkbox"/>

Je houdt in de kolom *SoortCursus* bij of het om een klassikale cursus of een zelfstudie cursus gaat:

- K betekent klassikale cursus.
- Z betekent zelfstudie cursus.

12.3.1 De entities definiëren die horen bij de Cursussen3

Je verwijdert eerst in *Opleidingen.edmx* de entities *Cursus*, *KlassikaleCursus* en *ZelfstudieCursus*, want deze entities krijgen andere eigenschappen.

Je voegt een eerste entity toe die hoort bij de table *Cursussen3*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Tables* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *Cursussen3* en je kiest *Finish*.

Je doet enkele aanpassingen in de entity *Cursussen3*.

- Je wijzigt de naam van de entity *Cursussen3* naar *Cursus*
- Je wijzigt in het properties venster *Entity Set Name* naar *Cursussen*.
- Je wijzigt in het properties venster *Abstract* naar *True*.

12.3.2 De entity KlassikaleCursus toevoegen en inheritance definiëren

- Je maakt een nieuwe entity *KlassikaleCursus*:
 - Je sleept vanuit de toolbox een *Entity* op de designer.
 - Je wijzigt de naam van *Entity1* naar *KlassikaleCursus*
 - Je verwijdert de property *Id*
- Je definieert de inheritance tussen *KlassikaleCursus* en *Cursus*
 - Je kiest in de toolbox een *Inheritance* en sleept van *KlassikaleCursus* naar *Cursus*.
- Je brengt de properties die specifiek zijn voor een klassikale cursus over van *Cursus* naar *KlassikaleCursus*
 - Je selecteer de properties *Van* en *Tot* in *Cursus*.
Je klikt met de rechtermuisknop in deze selectie en je kiest *Cut*
 - Je selecteert *KlassikaleCursus* met de rechtermuisknop en je kiest *Paste*.
- Je associeert *KlassikaleCursus* met de records uit de table *Cursussen3* die in de kolom *SoortCursus* de waarde *K* hebben
 - Je selecteert *KlassikaleCursus* met de rechtermuisknop en je kiest *Table Mapping*
 - Je selecteert *<Add a Table or View>* en je kiest *Cursussen3*.
 - Je selecteert *<Add a Condition>* en je kiest *SoortCursus*
 - Je laat *Operator* op de waarde *=*
 - Je tikt bij *Value/Property* de waarde *K*

12.3.3 De entity *ZelfstudieCursus* toevoegen en inheritance definiëren

- Je maakt een nieuwe entity *ZelfstudieCursus*:
 - Je sleept vanuit de toolbox een *Entity* op de designer.
 - Je wijzigt de naam van *Entity1* naar *ZelfstudieCursus*
 - Je verwijdert de property *Id*
- Je definieert de inheritance tussen *ZelfstudieCursus* en *Cursus*
 - Je kiest in de toolbox een *Inheritance* en sleept van *ZelfstudieCursus* naar *Cursus*.
- Je brengt de property die specifiek is voor een zelfstudie cursus over van *Cursus* naar *ZelfstudieCursus*
 - Je selecteer de property *Duurtijd* in *Cursus* met de rechtermuisknop en je kiest *Cut*
 - Je selecteert *ZelfstudieCursus* met de rechtermuisknop en je kiest *Paste*.
- Je associeert *ZelfstudieCursus* met de records uit de table *Cursussen3* die in de kolom *SoortCursus* de waarde *Z* hebben
 - Je selecteert *ZelfstudieCursus* met de rechtermuisknop en je kiest *Table Mapping*
 - Je selecteert *<Add a Table or View>* en je kiest *Cursussen3*.
 - Je selecteert *<Add a Condition>* en je kiest *SoortCursus*
 - Je laat *Operator* op de waarde *=*
 - Je tikt bij *Value/Property* de waarde *Z*
- Je verwijdert de property *SoortCursus* uit *Cursus*. Deze dient enkel om intern in EF het onderscheid te maken tussen een klassikale cursussen en zelfstudie cursussen. Je hoeft de letter in de kolom niet rechtstreeks te lezen in de rest van de applicatie.

Het eindresultaat in de designer is hetzelfde als op het einde van hoofdstuk 12.2.4

12.3.4 De entities gebruiken vanuit je code

Je gebruikt deze entities op juist dezelfde manier als bij *Table* per concrete class.

Voorbeeld 1 in de method *Main*: je toont de naam en soort van alle cursussen:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen
                orderby cursus.Naam
                select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine("{0}: {1}", cursus.Naam, cursus.GetType().Name);
    }
}
```

Voorbeeld 2 in de method *Main*: je toont enkel van zelfstudie cursussen de naam:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen
                where cursus is ZelfstudieCursus
                orderby cursus.Naam
                select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine(cursus.Naam);
    }
}
```

Voorbeeld 3 in de method *Main*: je voegt een zelfstudie cursus toe:

```
using (var entities = new OpleidingenEntities())
{
    entities.Cursussen.Add(
        new ZelfstudieCursus {Naam = "Duitse correspondentie", Duurtijd = 6});
    entities.SaveChanges();
}
```

12.4 Table per type (TPT)

12.4.1 De tables *Cursussen4*, *KlassikaleCursussen4* en *ZelfstudieCursussen4* toevoegen aan de database

Table per type is de derde manier om inheritance na te bootsen in de database. Bij deze manier bevat de database één table per class uit de inheritance hiërarchy. (*Cursus*, *KlassikaleCursus* en *ZelfstudieCursus*)

Het script *TablePerSubClass.sql* voegt aan de database *Opleidingen* volgende tables toe: *Cursussen4*, *KlassikaleCursussen4* en *ZelfstudieCursussen4*.

Je voert dit script uit in de SQL Server Management Studio.

Deze tables hebben volgende structuur:

KlassikaleCursussen4		
Column Name	Data Type	Allow Nulls
CursusNr	int	<input type="checkbox"/>
Van	datetime	<input type="checkbox"/>
Tot	datetime	<input type="checkbox"/>

Cursussen4		
Column...	Data Type	Allow Nulls
CursusNr	int	<input type="checkbox"/>
Naam	varchar(50)	<input type="checkbox"/>

ZelfstudieCursussen4		
Column Name	Data Type	Allow Nulls
CursusNr	int	<input type="checkbox"/>
Duurtijd	int	<input type="checkbox"/>

- Per klassikale cursus is er één record in de table *Cursussen4* én één bijbehorend record in de table *KlassikaleCursussen4*. Beide records hebben dezelfde waarde in de kolom *CursusNr*.
- Per zelfstudie cursus is er één record in de table *Cursussen4* én één bijbehorend record in de table *ZelfstudieCursussen4*. Beide records hebben dezelfde waarde in de kolom *CursusNr*.
- De kolom *CursusNr* is in de table *Cursussen4* een autonummer kolom. De kolom *CursusNr* is in de tables *KlassikaleCursussen4* en *ZelfstudieCursussen4* geen autonummer veld. Het veld is wel primary key én tegelijk een foreign key die verwijst naar de kolom *CursusNr* in de table *Cursussen4*. Je verzekert met deze foreign key dat in de tables *KlassikaleCursussen4* en *ZelfstudieCursussen4* enkel records kunnen voorkomen die een bijbehorend record hebben in de table *Cursussen4*.

12.4.2 De entities definiëren die horen bij de tables *Cursussen4*, *KlassikaleCursussen4* en *ZelfstudieCursussen4*

Je verwijdert eerst in *Opleidingen.edmx* de entities *Cursus*, *KlassikaleCursus* en *ZelfstudieCursus*. want deze entities krijgen andere eigenschappen.

Je voegt aan *Opleidingen.edmx* entities toe die horen bij de tables *Cursussen4*, *KlassikaleCursussen4* en *ZelfstudieCursussen4*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Tables* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *Cursussen4*, *KlassikaleCursussen4* en *ZelfstudieCursussen4*
- Je kiest *Finish*.

Je ziet de entities *Cursussen4*, *KlassikaleCursussen4* en *ZelfstudieCursussen4*

Je corrigeert enkele namen:

- Je wijzigt de naam van de entity *Cursussen4* naar *Cursus*
- Je wijzigt in het properties venster *Entity Set Name* naar *Cursussen*
- Je wijzigt in het properties venster *Abstract* naar *True*.
- Je wijzigt de naam van de entity *KlassikaleCursussen4* naar *KlassikaleCursus*
- Je wijzigt de naam van de entity *ZelfstudieCursussen4* naar *ZelfstudieCursus*

12.4.3 Inheritance definiëren

- Op dit moment is het verband tussen *Cursus* en *KlassikaleCursus* geen inheritance, maar een associatie 1 – 0..1. Je verwijdert deze associatie
- Op dit moment is het verband tussen *Cursus* en *ZelfstudieCursus* geen inheritance, maar een associatie 1 – 0..1. Je verwijdert deze associatie
- Je definieert een inheritance verband tussen *KlassikaleCursus* en *Cursus*
 - Je kiest in de toolbox een *Inheritance* en sleept van *KlassikaleCursus* naar *Cursus*.
 - Je verwijdert in de *KlassikaleCursus* de property *CursusNr*.
KlassikaleCursus erft al een *CursusNr* van *Cursus*.
- Je definieert de inheritance tussen *ZelfstudieCursus* en *Cursus*
 - Je kiest in de toolbox een *Inheritance* en sleept van *ZelfstudieCursus* naar *Cursus*.
 - Je verwijdert in de *ZelfstudieCursus* de property *CursusNr*. *ZelfstudieCursus* erft al een *CursusNr* van *Cursus*.

Het eindresultaat in de designer is hetzelfde als op het einde van hoofdstuk 12.2.4

12.4.4 De entities gebruiken vanuit je code

Je gebruikt deze entities op juist dezelfde manier als bij Table per concrete class.

Voorbeeld 1 in de method *Main*: je toont de naam en soort van alle cursussen:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen
                orderby cursus.Naam
                select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine("{0}: {1}", cursus.Naam, cursus.GetType().Name);
    }
}
```

Voorbeeld 2 in de method *Main*: je toont van alle cursussen, behalve zelfstudiecursussen de naam:

```
using (var entities = new OpleidingenEntities())
{
    var query = from cursus in entities.Cursussen
                where !(cursus is ZelfstudieCursus)
                orderby cursus.Naam
                select cursus;
    foreach (var cursus in query)
    {
        Console.WriteLine(cursus.Naam);
    }
}
```

Voorbeeld 3 in de method *Main*: je voegt een zelfstudie cursus toe:

```
using (var entities = new OpleidingenEntities())
{
    entities.Cursussen.Add(
        new ZelfstudieCursus { Naam = "Italiaanse correspondentie", Duurtijd = 6 });
    entities.SaveChanges();
}
```



Zichtrekeningen - spaarrekeningen: zie takenbundel

13 COMPLEX TYPES

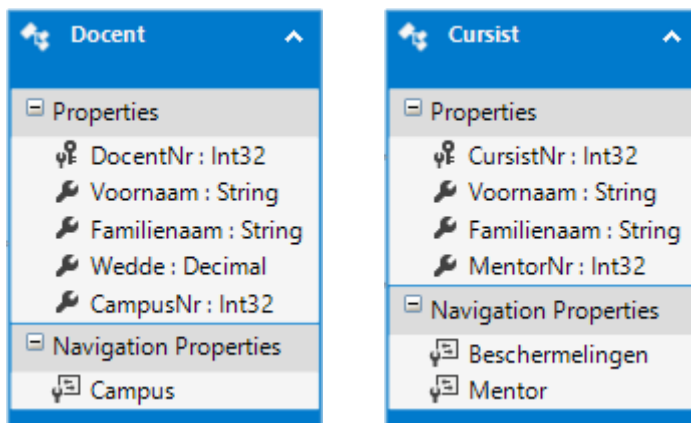
13.1 Algemeen

Je leerde in het hoofdstuk 2.2 (Granularity) dat je de data in een database maar op twee niveaus kan opsplitsen: tables en columns (binnen die tables).

Je hebt bij object oriëntatie geen beperking bij het opsplitsen van data:

- een class kan gewone properties bevatten (int, string, ...)
- een class kan ook navigation properties bevatten die verwijzen naar andere classes.
Deze andere classes kunnen zelf gewone properties en navigation properties bevatten ...

In het entity data model *Opleidingen.edmx* komen de properties *Voornaam* en *Familienaam* als scalar properties voor in de entities *Docent* én *Cursist*:



Een betere object georiënteerde voorstelling van deze twee properties is als volgt:

Je maakt een class *Naam*, die deze properties *Voornaam* en *Familienaam* bevat.

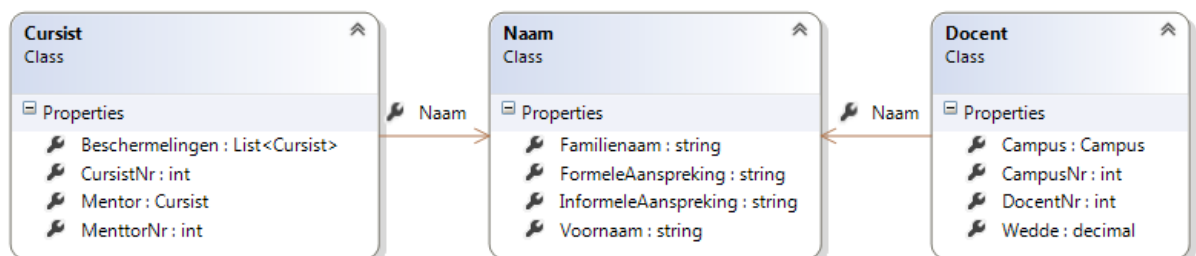
Je kan in deze class ook properties toevoegen die typisch zijn voor een naam. Voorbeelden:

- de property *InformeleAanspreking* (deze geeft *Hallo Eddy* terug als de voornaam *Eddy* is)
- de property *FormeleAanspreking* (deze geeft *Geachte Eddy Wally* terug als de voornaam *Eddy* is en de familienaam *Wally* is).

Je vervangt in *Docent* én *Cursist* de properties *Voornaam* en *Familienaam* door één property van het type *Naam*.

Op deze manier hebben deze entities met deze property nog altijd een voornaam en een familienaam, maar je kan vanuit deze entities ook de properties *InformeleAanspreking* en *FormeleAanspreking* gebruiken, die één keer gedefinieerd zijn in de class *Naam*.

Docent, *Cursist* en *Naam* als een class diagram:



Een class, zoals *Naam*, die je gebruikt vanuit één of meerdere entities, noemt men een *complex type*. Een complex type heeft volgende beperkingen:

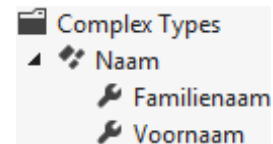
- Een complex type kan niet erven van een ander complex type.
- Een property die als type een complex type heeft, de property *Naam* in de entities *Docent* en *Cursist* kan niet leeg zijn (nullable).

13.2 Een complex type maken op basis van een bestaande entity

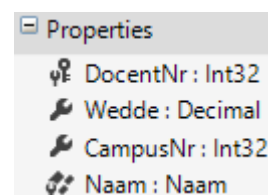
Je kan een complex type maken op basis van een bestaande entity:

- Je kiest de entity properties die je wil overbrengen naar een complex type:
 - Je klikt in *Docent* op de property *Voornaam*.
 - Je geeft een Ctrl+klik op de property *Familienaam*.
- Je klikt met de rechtermuisknop op één van deze properties.
- Je kiest *Refactor* en daarna *Move to New Complex Type*.
- Je wijzigt in het venster *Model Browser* de naam van het nieuwe complex type van *ComplexType1* naar *Naam*.

Als je dit complex type openklapt in de Model Browser, zie je dat het de properties *Familienaam* en *Voornaam* bevat.



Visual Studio heeft in *Docent* de properties *Voornaam* en *Familienaam* vervangen door een property *ComplexProperty*. Deze property heeft als type het complex type *Naam*. Je wijzigt de property naam naar *Naam* (de property beschrijft de naam van de docent).



Visual Studio heeft de table mappings van de *Docent* juist aangepast:

- De property *Naam.Familienaam* hoort bij de kolom *Familienaam*
- De property *Naam.Voornaam* hoort bij de kolom *Voornaam*

Column Mappings

DocentNr : int	↔	DocentNr : Int32
Voornaam : nvarchar	↔	Naam.Voornaam : String
Familienaam : nvarchar	↔	Naam.Familienaam : String
Wedde : decimal	↔	Wedde : Decimal
CampusNr : int	↔	CampusNr : Int32

Je verwijdert in *DocentUitbreiding.cs* de readonly property *Naam* die je vroeger maakte.

13.3 Een complex type herbruiken in een andere entity

Je zal nu het complex type *Naam* herbruiken in *Cursist*:

- Je verwijdert in *Cursist* de properties *Voornaam* en *Familienaam*
- Je voegt een property met als naam *Naam* en als type *Naam* (het complex type) toe:
 - Je klikt met de rechtermuisknop ergens in *Cursist*.
 - Je kiest *Add New* en daarna *Complex Property*
 - Je wijzigt de naam van de property van *ComplexProperty* naar *Naam*. Het type van de property is het complex type *Naam*, omdat er momenteel maar één complex type is. Als je diagram meerdere complex types bevat, kan je het type van de property instellen in het properties venster.
- Je corrigeert de table mappings van *Cursist*:
 - Je klikt met de rechtermuisknop in *Cursist*.
 - Je kiest *Table Mapping*.
 - Je ziet dat de kolommen *Voornaam* en *Familienaam* niet meer gekoppeld zijn aan *Cursist* properties
 - Je kiest bij de kolom *Voornaam* de property *Naam.Voornaam*
 - Je kiest bij de kolom *Familienaam* de property *Naam.Familienaam*.

13.4 Complex type als partial class

Het complex type *Naam* is een partial class. Je kan deze class uitbreiden door een source file *NaamUitbreiding.cs* toe te voegen aan het project, waarin je dezelfde class definieert als een partial class. Je voegt in deze source properties en of methods toe:

```
namespace EFCursus
{
    public partial class Naam
    {
        public override string ToString()
        {
            return Voornaam + ' ' + Familiennaam;
        }
        public string InformeleBegroeting
        {
            get
            {
                return "Hallo " + Voornaam;
            }
        }
        public string FormeleBegroeting
        {
            get
            {
                return "Geachte " + Voornaam + ' ' + Familiennaam;
            }
        }
    }
}
```

13.5 Voorbeeldgebruik

Je spreekt als voorbeeld de cursisten aanspreken met hun informele begroeting in de method *Main*

```
using (var entities = new OpleidingenEntities())
{
    foreach (var cursist in
        (from eenCursist in entities.Cursisten select eenCursist))
    {
        Console.WriteLine(cursist.Naam.InformeleBegroeting);
    }
}
```

14 ENUMS

14.1 Algemeen

Als de inhoud van een variabele één waarde kan zijn die je kiest uit een beperkt aantal waarden, is het in C# aan te raden

- deze waarden te definiëren in een enum.
- deze enum te gebruiken als type voor de variabele.

Voorbeelddeclaratie van een enum:

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

Voorbeeldgebruik van deze enum in een variabele mijnGeslacht:

```
Geslacht mijnGeslacht = Geslacht.Man;
```

14.2 Voordelen van een enum

14.2.1 De compiler controleert de inhoud van een enum variabele

De C# compiler controleert dat de variabele mijnGeslacht enkel de waarde Geslacht.Man of Geslacht.Vrouw kan bevatten. Je hebt meer kans op tikfouten als je geen enum gebruikt als type van de variabele mijnGeslacht, maar een string:

```
string mijnGeslacht = "D";
```

De variabele bevat een verkeerde waarde (je verwachtte enkel "M" of "V").

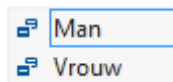
De C# compiler controleert de inhoud van een string echter niet.

Je ontdekt deze fout hopelijk tijdens het uittesten van de code !

14.2.2 Visual Studio helpt bij het invullen van een enum variabele

Visual Studio toont bij het invullen van een enum variabele een popup venstertje met de mogelijk waarden.

```
Geslacht mijnGeslacht = Geslacht.
```



Visual Studio kan dit niet bij het invullen van een string variabele.

14.3 Enums en EF

Het type van een property van een entity of complex type kan een enum zijn.

Ook bij zo'n property hoort een kolom in de table in de database.

Een database kent geen enum als type van een kolom van een table.

Sommigen kiezen dan als kolomtype varchar, anderen kiezen int.

EF kan een enum property associëren met een kolom in de table in de database, als het type van die kolom een geheel getal type is (jammer genoeg niet als het type van de kolom varchar is).

Je zal bijvoorbeeld van iedere Docent entity een extra property Geslacht bijhouden.

Het type van deze property zal een enum Geslacht zijn.

Het script *GeslachtToevoegen.sql* voegt aan de table *Docenten* van de database *Opleidingen* een kolom *Geslacht* toe. Het type van deze kolom is int.

Dit script vult bij de bestaande records (mannen) de inhoud van deze kolom met 1.

Dit script voegt ook enkele records met vrouwelijke docenten toe en vult de inhoud van de kolom *Geslacht* in deze records met 2.

Het geslacht wordt dus in de kolom *Geslacht* voorgesteld met 1 (Man) of 2 (Vrouw).

Je voert dit script uit in de SQL Server Management Studio.

Je zal dit getal in Docent voorstellen met een enum Geslacht.

- Je klikt met de rechtermuisknop in *Opleidingen.edmx* en je kiest *Update Model from Database*.
- Je kiest *Finish*
- Visual Studio heeft aan *Docent* een property *Geslacht* van het type *int32* toegevoegd.
- Je klikt met de rechtermuisknop op deze property en je kiest *Convert to Enum*.
- Je tikt bij *Enum Type Name* de naam van de enum die je wenst aan te maken: *Geslacht*
- Je tikt onder *Member Name* de eerste mogelijke waarde in deze enum: *Man*
Je tikt daarnaast (onder *Value*) het getal dat Man voorstelt in de database: 1
- Je tikt op een nieuwe regel onder *Member Name* de tweede mogelijke waarde in deze enum: *Vrouw*. Je tikt daarnaast (onder *Value*) het getal dat Man voorstelt in de database: 2



Opmerking: als je *Value* leeg laat, associeert EF de eerste enum waarde (Man) met de waarde 0 in de database, de tweede enum waarde (Vrouw) met de waarde 1, ...

Als EF records leest uit de tabel docenten, zet hij de getallen in de kolom Geslacht om naar de bijbehorende enum waarden in de property Geslacht van Docent entities.

Je ziet dit met volgend voorbeeld in de method Main

```
using (var entities = new OpleidingenEntities())
{
    foreach (var docent in entities.Docenten)
    {
        Console.WriteLine("{0}:{1}", docent.Naam, docent.Geslacht);
    }
}
```

Als EF een Docent entity toevoegt als een nieuw record in de table docenten, zet hij de enum waarde in de property Geslacht om naar het bijbehorend getal in de kolom Geslacht.

Je ziet dit met volgend voorbeeld in in de method Main

```
using (var entities = new OpleidingenEntities())
{
    entities.Docenten.Add(
        new Docent { Naam = new Naam { Voornaam = "Brigitta", Familienaam = "Roos" },
        Wedde = 2000, Geslacht = Geslacht.Vrouw, CampusNr = 1 });
    entities.SaveChanges();
}
```

15 VIEWS

15.1 Algemeen

Een database kan naast tables ook views en stored procedures bevatten.

Je ziet in dit hoofdstuk hoe je views aanspreekt met EF.

Een view is een virtuele table die data voorstelt uit één of meerdere echte tables.

15.2 Een view aanmaken

Je maakt met het volgend SQL statement bijvoorbeeld een view met de naam *WestVlaamseCampussen* die de campussen voorstelt uit West-Vlaanderen:

```
create view WestVlaamseCampussen as
select * from Campussen
where PostCode between '8000' and '8999'
```

Er zijn beperkingen bij het aanmaken van een view, waaronder :

- Je kan order by niet gebruiken
- Je kan geen parameters definiëren

15.3 De data van een view lezen vanuit SQL

Je leest met het volgend SQL statement de data van de view:

```
select * from WestVlaamseCampussen
```

15.4 De voorbeeldview

Het script *BestBetaaldeDocentenPerCampusMaken.sql* voegt aan de database een view toe met de naam *BestBetaaldeDocentenPerCampus*.

Deze view leest per campus de docent(en) met de hoogste wedde.

Je voert dit script uit in de SQL Server Management Studio.

Je kan de definitie van de view zien in Visual Studio op volgende manier:

- Je klappt in de Server Explorer de databaseconnectie *Opleidingen* open.
- Je klappt daarbinnen het onderdeel *Views* open.
- Je selecteert de view *BestBetaaldeDocentenPerCampus* met de rechtermuisknop en je kiest *Open View Definition*.

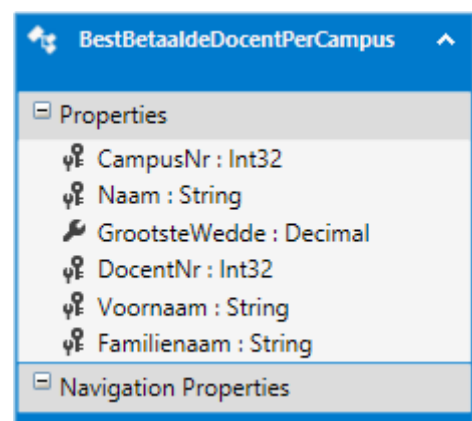
Je kan het resultaat van de view zien in Visual Studio op volgende manier:

- Je selecteert in de Server Explore de view *BestBetaaldeDocentenPerCampus* met de rechtermuisknop en je kiest *Show Results*. Je ziet dat het resultaat niet gesorteerd is. Je lost dit straks op door te sorteren in een LINQ query.

15.5 De entities definiëren die horen bij de view

Je voegt aan *Opleidingen.edmx* een entity toe die hoort bij de view die je maakte

- Je klikt met de rechtermuisknop in *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klappt in het tabblad *Add* het onderdeel *Views* open.
- Je klappt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *BestBetaaldeDocentenPerCampus*
- Je kiest *Finish* en je negeert de warning.
- Je ziet de entity *BestBetaaldeDocentenPerCampus*
- Je wijzigt de naam van de entity naar *BestBetaaldeDocentPerCampus*.



15.6 De entities aanspreken vanuit code

```
using (var entities = new OpleidingenEntities())
{
    var query = from bestBetaaldeDocentPerCampus
                 in entities.BestBetaaldeDocentenPerCampus
                 orderby bestBetaaldeDocentPerCampus.CampusNr,
                        bestBetaaldeDocentPerCampus.Voornaam,
                        bestBetaaldeDocentPerCampus.Familienaam
                 select bestBetaaldeDocentPerCampus;
    var vorigCampusNr = 0;
    foreach (var bestbetaaldeDocentPerCampus in query)
    {
        if (bestbetaaldeDocentPerCampus.CampusNr != vorigCampusNr)
        {
            Console.WriteLine("{0} {1} Grootste wedde:",
                              bestbetaaldeDocentPerCampus.Naam, bestbetaaldeDocentPerCampus.GrootsteWedde);
            vorigCampusNr = bestbetaaldeDocentPerCampus.CampusNr;
        }
        Console.WriteLine("\t{0} {1}",
                          bestbetaaldeDocentPerCampus.Voornaam, bestbetaaldeDocentPerCampus.Familienaam);
    }
}
```



Totale saldo per klant: zie takenbundel

16 STORED PROCEDURES

16.1 Algemeen

Een stored procedure is een procedure die in de database is opgeslagen.

- Een stored procedure kan één of meerdere SQL statements bevatten. Dit kunnen select, insert, update en/of delete statements zijn.
- Een stored procedure kan ook parameters bevatten.

16.2 Een stored procedure aanmaken

Je maakt met het volgend SQL statement een stored procedure.

Deze stored procedure heeft twee parameters *VanPostCode* en *TotPostCode*.

De stored procedure leest de campussen met een postcode die valt tussen deze parameters:

```
create procedure CampussenVanTotPostCode(@VanPostCode nvarchar(10),
    @TotPostCode nvarchar(10))
as
select * from Campussen
where Postcode between @VanPostCode and @TotPostCode
order by PostCode, Naam
```

16.3 Een stored procedure oproepen vanuit SQL

Je voert met het volgend statement de stored procedure uit:

```
execute campussenvantotpostcode '9000', '9999'
```

Het script *CampussenVanTotPostCodeMaken.sql* voegt aan de database de stored procedure *CampussenVanTotPostCode* toe. Je voert dit script uit in de SQL Server Management Studio.

16.4 Stored procedures oproepen met EF

Je kan een stored procedure oproepen vanuit EF als:

- De stored procedure data terug geeft in de vorm van entities, of
- De stored procedure data terug geeft in de vorm verschillend van entities, of
- De stored procedure geen data terug geeft, of
- De stored procedure een scalar value terug geeft

16.4.1 Een stored procedure die data terug geeft in de vorm van entities

Sommige stored procedures lezen data en geven die aan de applicatie die de procedure oproept.

Deze data kan dezelfde kolommen bevatten als één van de tables van de database.

Dan kan deze data ook voorgesteld worden als een verzameling entities.

De stored procedure *CampussenVanTotPostCode* leest bijvoorbeeld alle kolommen uit de table *Campussen*. Als je deze stored procedure oproept vanuit je applicatie, kan het resultaat van deze stored procedure voorgesteld worden als een verzameling *Campus* entities.

Je ziet hier hoe je zo'n stored procedure oproept met EF.

Je voegt de stored procedure toe aan *Opleidingen.edmx*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Stored Procedures* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *CampussenVanTotPostCode* en je kiest *Finish*.

Je ziet geen nieuw onderdeel in de designer, maar Visual Studio heeft aan de object context class (*OpleidingenEntities*) een method toegevoegd die zelf de stored procedure oproept.

Jij zal vanuit je code de stored procedure oproepen door deze nieuwe method op te roepen.

De method naam is gelijk aan de bijbehorende stored procedure naam: *CampussenVanTotPostCode*

Je ziet dit op volgende manier:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Model Browser*.
- Je klapt *OpleidingenModel* open.
- Je klapt daarbinnen *Function Imports* open
- Je ziet de function import *CampussenVanTotPostCode*
- Je klapt *CampussenVanTotPostCode* open.
- Je ziet de twee parameters van de method:
@VanPostCode en *@TotPostCode*.
- Als je één van deze parameters aanklikt, zie je in het properties venster bij *Type* dat deze parameters in de method het type String hebben.
- Je geeft aan dat de function Campus Entities teruggeeft
 - Je selecteert *CampussenVanTotPostCode* in de *Model Browser*
 - Je klikt in het properties venster op de knop ... bij *Return Type*
 - Je kiest bij *Returns a Collection of* voor *Entities*
 - Je kiest bij *Entities* voor *Campus*
 - Je kiest *OK*

Je kan in de *Main* van *Program.cs* de method *CampussenVanTotPostCode* oproepen, die op zijn beurt de stored procedure oproept. De method geeft je een verzameling Campus entities, waarover je bijvoorbeeld kan itereren met *foreach*:

```
using (var entities = new OpleidingenEntities())
{
    foreach (var campus in entities.CampussenVanTotPostCode("8000", "8999"))
    {
        Console.WriteLine("{0}: {1}", campus.Naam, campus.PostCode);
    }
}
```

16.4.2 Een stored procedure die data terug geeft in een vorm die niet overeenstemt met de structuur van een entity

Sommige stored procedures bieden data aan, waarvan de kolomstructuur niet overeenstemt met de property structuur van één van je entity classes.

Het script *AantalDocentenPerVoornaamMaken.sql* voegt aan de database *Opleidingen* de stored procedure *AantalDocentenPerVoornaam* toe.

Je voert dit script uit in de SQL Server Management Studio.

Deze stored procedure geeft een lijst met per voornaam de voornaam en het aantal docenten die deze voornaam heeft:

Voornaam	Aantal
Adelin	1
Adolf	1
Adolphe	1
Albert	9
...	

De structuur van één rij uit deze lijst komt niet overeen met de structuur van één van je entities.

Als je deze stored procedure integreert in EDMX, maakt Visual Studio een class die één rij uit deze lijst weerspiegelt: een class met een string property *Voornaam* en een int property *Aantal*.

Je voegt de stored procedure toe aan *Opleidingen.edmx*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Stored Procedures* open.
- Je klapt daarbinne het onderdeel *dbo* open.
- Je plaatst een vinkje bij *AantalDocentenPerVoornaam*
- Je kiest *Finish*.

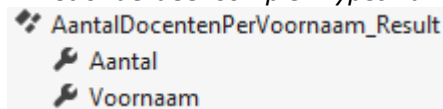
Visual Studio heeft aan de object context class (*OpleidingenEntities*) een method toegevoegd die zelf de stored procedure oproept. Jij zal vanuit je code de stored procedure oproepen door deze nieuwe method op te roepen.

De method naam is dezelfde als de bijbehorende stored procedure naam:

AantalDocentenPerVoornaam

Je ziet dit op volgende manier:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Model Browser*.
- Je klapt *OpleidingenModel* open.
- Je klapt daarbinnen *Function Imports* open
- Je selecteert de function import *AantalDocentenPerVoornaam*
- Je ziet in het Properties Venster bij Return Type *AantalDocentenPerVoornaam_Result*
Dit is de class die Visual Studio heeft aangemaakt als een weerspiegeling van één rij uit de resultaatdata van de stored procedure.
- Je ziet de structuur van deze class ook in de Model Browser, in het onderdeel *Complex Types* van *OpleidingenModel*



Je kan in de *Main* van *Program.cs* de method *AantalDocentenPerVoornaam* oproepen, die op zijn beurt de stored procedure oproept. De method geeft je een verzameling objecten van het type *AantalDocentenPerVoornaam_Result* terug:

```
using (var entities = new OpleidingenEntities())
{
    foreach (var voornaamAantal in entities.AantalDocentenPerVoornaam ())
    {
        Console.WriteLine("{0} {1}", voornaamAantal.Voornaam, voornaamAantal.Aantal);
    }
}
```

16.4.3 Een stored procedure die geen data terug geeft

Sommige stored procedures voegen records toe, wijzigen records of verwijderen records, maar geven daarna geen data terug aan de applicatie die de stored procedure heeft opgeroepen.

De volgende stored procedure geeft een weddeverhoging aan alle docenten:

```
create procedure WeddeVerhoging(@Percentage decimal(5,2))
as
update Docenten
set wedde = wedde * (1 + @Percentage / 100)
```

Het script *WeddeVerhogingMaken.sql* voegt aan de database *Opleidingen* de stored procedure *WeddeVerhoging* toe.

Je voert dit script uit in de SQL Server Management Studio.

Je ziet nu hoe je zo'n stored procedure oproept met EF.

Je voegt de stored procedure toe aan *Opleidingen.edmx*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Stored Procedures* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *WeddeVerhoging*
- Je kiest *Finish*.

Visual Studio heeft aan de object context class (*OpleidingenEntities*) een method toegevoegd die zelf de stored procedure oproept. Jij zal vanuit je code de stored procedure oproepen door deze nieuwe method op te roepen.

De method naam is gelijk aan de achterliggende stored procedure naam: *WeddeVerhoging*

Je ziet dit op volgende manier:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Model Browser*.
- Je klapt *OpleidingenModel* open.
- Je klapt daarbinnen *Function Imports* open
- Je ziet de function import *WeddeVerhoging*
- Je klapt *WeddeVerhoging* open.
- Je ziet de parameter van de method: *@Percentage*.
- Als je deze parameter aanklikt, zie je in het properties venster bij *Type* dat deze parameter in de method het type *Decimal* heeft.

Je kan in de *Main* van *Program.cs* de method *WeddeVerhoging* oproepen, die op zijn beurt de stored procedure oproept. De method geeft je het aantal aangepaste records terug:

```
Console.WriteLine("Opslagpercentage:");
decimal percentage;
if (decimal.TryParse(Console.ReadLine(), out percentage))
{
    using (var entities = new OpleidingenEntities())
    {
        var aantalDocentenAangepast = entities.WeddeVerhoging(percentage);
        Console.WriteLine("{0} docenten aangepast", aantalDocentenAangepast);
    }
}
else
{
    Console.WriteLine("Tik een getal");
}
```

16.4.4 Een stored procedure die data leest als een scalar value

Een scalar value is één enkele waarde: data met één rij én één kolom.

De volgende stored procedure geeft een scalar value terug:

```
create procedure AantalDocentenMetFamilienaam(@Familienaam nvarchar(50))
as
select count(*)
from Docenten
where Docenten.Familienaam = @Familienaam
```

Het script *AantalDocentenMetFamilieNaamMaken.sql* maakte in de database *Opleidingen* de stored procedure *AantalDocentenMetFamilieNaam*.

Je voert dit script uit in de SQL Server Management Studio.

Je ziet nu hoe je zo'n stored procedure oproept met EF.

Je voegt de stored procedure toe aan *Opleidingen.edmx*:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Update Model from Database*.
- Je klapt in het tabblad *Add* het onderdeel *Stored Procedures* open.
- Je klapt daarbinnen het onderdeel *dbo* open.
- Je plaatst een vinkje bij *AantalDocentenMetFamilieNaam*

Visual Studio heeft aan de object context class (*OpleidingenEntities*) een method toegevoegd die zelf de stored procedure oproept. Jij zal vanuit je code de stored procedure oproepen door deze nieuwe method op te roepen.

De method naam is gelijk aan de achterliggende stored procedure naam:

AantalDocentenMetFamilieNaam

Je ziet dit op volgende manier:

- Je klikt met de rechtermuisknop in de achtergrond van *Opleidingen.edmx*.
- Je kiest *Model Browser*.
- Je klapt *OpleidingenModel* open.
- Je klapt daarbinnen *Function Imports* open
- Je ziet de function import *AantalDocentenMetFamilieNaam*
- Je klapt *AantalDocentenMetFamilieNaam* open.
- Je ziet de parameter van de method: *@Familienaam*.
- Als je deze parameter aanklikt, zie je in het properties venster bij *Type* dat deze parameter in de method het type *String* heeft.

Je kan in de method *Main* de method *AantalDocentenMetFamilienaam* oproepen, die op zijn beurt de stored procedure oproept. De method geeft je een object van het type *ObjectResult* terug. Je voert op dit object de *First* method uit om de scalar returnwaarde van de stored procedure te lezen.

```
Console.WriteLine("Familienaam:");
var familienaam = Console.ReadLine();
using (var entities = new OpleidingenEntities())
{
    var aantalDocenten = entities.AantalDocentenMetFamilienaam(familienaam);
    Console.WriteLine("{0} docent(en)", aantalDocenten.First());
}
```



Administratieve kost: zie takenbundel

17 CODE FIRST

17.1 Algemeen

Je gebruikte tot nu een EDMX bestand om het verband te leggen tussen tables uit de database en classes in je programma.

Je zal nu de verbanden uitdrukken in C# code, onder andere met attributen (woorden tussen [en] die je voor een class of een variabele schrijft).

Ook de namen van classes en variabelen zijn belangrijk voor de configuratie.
Dit heet "Configuration by Convention".

Bij code first kan je een nieuwe database maken op basis van de C# code of met een bestaande database werken.

17.2 De entity classes voor de nieuwe database

Je maakt in Visual Studio een Console Application project met de naam *CodeFirstCursus*.

Je downloadt de laatste versie van de EF library en gebruikt die in het project

- Je klikt met de rechtermuisknop op het project in de Solution Explorer
- Je kiest *Manage NuGet Packages*
- Je kiest links *Online*
- Je kiest rechts *EntityFramework*
- Je kiest *Install*

Je voegt een entity class *Instructeur* toe.

```
using System;
namespace CodeFirstCursus
{
    public class Instructeur (1)
    {
        public int Id { get; set; } (2)
        public string Voornaam { get; set; } (3)
        public string Familiennaam { get; set; }
        public decimal Wedde { get; set; }
        public DateTime InDienst { get; set; }
        public void Opslag(decimal percentage)
        {
            Wedde *= (1M + percentage / 100M);
        }
    }
}
```

- (1) CodeFirst aanziet de naam van de class + het teken s als de naam van de table (*Instructeurs*) die bij de entity class zal horen.
- (2) Als CodeFirst een property vindt met de naam *Id*, aanziet CodeFirst die als de property die hoort bij de primary key kolom. De kolomnaam is ook *Id*.
- (3) De naam van de kolom die hoort bij een property is dezelfde als de naam van de property.

Je voegt een entity class *Campus* toe.

```
namespace CodeFirstCursus
{
    public class Campus (1)
    {
        public int CampusId { get; set; } (2)
        public string Naam { get; set; }
    }
}
```

- (1) CodeFirst aanziet de naam van de class als de naam van de table (*Campus*) die bij de entity class hoort. CodeFirst voegt geen s toe, omdat de naam van de class al eindigt op s.

- (2) Als CodeFirst een property vindt waarvan de naam gelijk is aan de naam van de class, gevolgd door *Id*, aanziet CodeFirst deze property als degene die bij de primary key hoort. De kolomnaam is ook *CampusId*. Er ligt nog geen verband tussen *Instructeur* en *Campus*. Je legt dit verband verder in de cursus.

17.3 De DbContext class

Bij een EDMX maakte Visual Studio een DbContext class voor je.

De DbContext class is het aanspreekpunt naar de database.

Bij CodeFirst maak je de DbContext class zelf.

Je voegt een class *VDABContext* toe

```
using System.Data.Entity;
namespace CodeFirstCursus
{
    public class VDABContext : DbContext           (1)
    {
        public DbSet<Instructeur> Instructeurs { get; set; }           (2)
        public DbSet<Campus> Campussen { 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*.

17.4 De connectionString

Je maakt in *App.config* een connectionString naar de nieuwe databse, onder *</startup>*

```
<connectionStrings>
  <add name="VDABContext"                               (1)
    providerName="System.Data.SqlClient"
    connectionString="Server=.\SQLEXPRESS;Database=VDAB;Trusted_Connection=true;" />
</connectionStrings>
```

- (1) De *name* van de connectionString moet gelijk zijn aan je *DbContext* class.


17.5 De DbContext gebruiken


Je wijzigt *Program.cs*

```
using System;
namespace CodeFirstCursus
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var context = new VDABContext())
            {
                var jean = new Instructeur
                {
                    Voornaam = "Jean", Familienaam = "Smits", Wedde = 1000,
                    InDienst = new DateTime(1994, 8, 1)
                };
                context.Instructeurs.Add(jean);
                context.SaveChanges();
                Console.WriteLine(jean.Id);
                // zoeken op primary key
                Console.WriteLine(context.Instructeurs.Find(1).Familienaam);
            }
        }
    }
}
```

Je ziet dat je de entity classes en de context gebruikt op dezelfde manier als ze zouden gemaakt zijn met een EDMX bestand.

Je voert het programma uit. Je ziet daarna met de SQL Server Management Studio een nieuwe database *VDAB*, met de tables *Instructeurs* en *Campussen*

Instructeurs	
	Id
	Voornaam
	Familienaam
	Wedde
	InDienst

Campus	
	CampusId
	Gemeente

17.6 De database opnieuw maken

Je voegt aan de class *Instructeur* een property *HeeftRijbewijs* toe

```
public bool HeeftRijbewijs { get; set; }
```

Je vult deze property in bij de entity die je maakt in *Program.cs*

```
var jean = new Instructeur
{
    Voornaam = "Jean", Familienaam = "Smits", Wedde = 1000,
    InDienst = new DateTime(1966, 8, 1), HeeftRijbewijs=true
};
```

De table *Instructeurs* bevat echter geen kolom *HeeftRijbewijs*.

Je kan dit oplossen door volgende regel als eerste te tikken in de method *Main* van *Program.cs*

```
System.Data.Entity.Database.SetInitializer(
    new DropCreateDatabaseIfModelChanges<VDABContext>()); // using System.Data.Entity nodig (1)
```

- (1) Met deze opdracht controleert CodeFirst of de tables in de database nog dezelfde structuur hebben als de bijbehorende entity classes.
 Als dit niet het geval is, verwijdert CodeFirst de database en maakt hem opnieuw aan.
 Het verwijderen lukt enkel als geen ander programma (bijvoorbeeld SQL Server Management Studio)de database heeft geopend.

Je voert het programma uit.

De table *Instructeurs* bevat daarna ook een kolom *HeeftRijbewijs*.

17.7 De aangemaakte table structuur verfijnen

17.7.1 Expliciet de table naam instellen

Als je de automatische table naam niet goed vindt, kan je die instellen met het attribuut *Table* bij de entity class. Je tikt voor class *Campus*

```
[Table("Campussen")] //using System.ComponentModel.DataAnnotations.Schema;
```

17.7.2 Expliciet de kolomnaam instellen

Als je de automatische kolom naam niet goed vindt, kan je die instellen met het attribuut *Column* bij de property. Je tikt voor de property *Wedde* in de class *Instructeur*

```
[Column("maandwedde")] //using System.ComponentModel.DataAnnotations.Schema;
```

17.7.3 Een kolom instellen als verplicht in te vullen

Als je de kolom, die bij een property hoort, wil instellen als verplicht in te vullen, doe je dit met het attribuut *Required* bij de property. Je tikt voor de property *Naam* in de class *Campus* :

```
[Required] //using System.ComponentModel.DataAnnotations
```


17.7.4 Een kolom instellen als niet verplicht in te vullen

Een kolom die bij een property hoort met een primitief data type (*int*, *long*, *bool*, ...) is standaard verplicht in te vullen. Je kan dit aanpassen door de property nullable te maken.

Je wijzigt de property *HeeftRijbewijs* in de class *Instructeur*

```
public bool? HeeftRijbewijs { get; set; }
```

17.7.5 Het maximum aantal tekens in een varchar kolom instellen

Je kan met het attribuut *StringLength* het maximum aantal tekens in een *varchar* kolom instellen.

Je tikt voor de property *Naam* in de class *Campus* volgende regel:

```
[StringLength(50)]
```

17.7.6 Het kolomtype instellen

Je kan met het attribuut *Column* het kolomtype instellen dat bij een property hoort.

Bij een *Date* property hoort standaard een *datetime* kolom. Je tikt voor de property *InDienst* van de class *Instructeur* volgende regel, om een *date* kolom te bekomen

```
[Column(TypeName="date")]
```

17.7.7 De property instellen die bij de primary key hoort.

Je kan met het attribuut *Key* instellen welke property bij de primary key hoort.

Je wijzigt de property *Id* van de class *Instructeur* naar

```
[Key] // using System.ComponentModel.DataAnnotations;  
public int InstructeurNr { get; set; }
```

Je wijzigt *jean.Id* naar *jean.InstructeurNr* in *Program.cs*

17.7.8 Een primary key die geen int met autonumber is

Default is de primary key bij CodeFirst een int met autonumber. Als je de primary key van nieuwe entities zelf invult in je C# code, duid je dit aan met het attribuut *DatabaseGenerated*

Je voegt volgende class toe:

```
using System.ComponentModel.DataAnnotations;  
using System.ComponentModel.DataAnnotations.Schema;  
namespace CodeFirstCursus  
{  
    [Table("Landen")]  
    public class Land  
    {  
        [Key, DatabaseGenerated(DatabaseGeneratedOption.None)]  
        public string LandCode { get; set; }  
        public string Naam { get; set; }  
    }  
}
```

Je voegt volgende property toe aan de class *VDABContext*

```
public DbSet<Land> Landen { get; set; }
```

Je kan het programma uitproberen. De database wordt geschrapt en opnieuw aangemaakt.

17.8 Complex type

Zowel een instructeur als een campus krijgen een adres, bestaande uit straat, huisnummer, postcode en gemeente. Dit worden in beide tables *Instructeurs* en *Campussen* 4 extra kolommen. In de classes is het vervelend om aan beide classes vier properties toe te voegen.

Een beter oplossing is deze vier properties één keer te beschrijven in een class *Adres* (een complex type). De class *Instructeur* en *Campus* hebben dan één nieuwe property, van het type *Adres*.

Je voegt de class *Adres* toe:

```
using System.ComponentModel.DataAnnotations.Schema;
[ComplexType]
public class Adres
{
    public string Straat { get; set; }
    public string HuisNr { get; set; }
    public string PostCode { get; set; }
    public string Gemeente { get; set; }
}
```

(1)

(1) Je tikt *ComplexType* bij een class die een complex type voorstelt.

Je voegt aan de class *Campus* een property toe: `public Adres Adres { get; set; }`

Je voegt aan de class *Instructeur* een property toe: `public Adres Adres { get; set; }`



Je wijzigt in *Program.cs* de variabele *jean*:

```
var jean = new Instructeur
{
    Voornaam = "Jean", Familienaam = "Smits", Wedde = 1000,
    InDienst = new DateTime(1966, 8, 1), HeeftRijbewijs=true,
    Adres = new Adres
    {
        Straat="Keizerslaan", HuisNr="11", PostCode="1000", Gemeente="Brussel"
    }
};
```

Je kan het programma uitproberen.

De database wordt geschrapt en opnieuw aangemaakt.

De tables hebben nu volgende structuur

Instructeurs	Campussen
 InstructeurNr	 CampusId
Voornaam	Naam
Familienaam	Adres_Straat
maandwedde	Adres_HuisNr
InDienst	Adres_PostCode
HeeftRijbewijs	Adres_Gemeente
Adres_Straat	
Adres_HuisNr	
Adres_PostCode	
Adres_Gemeente	

Je kan het herhalend woord *Adres_* in de kolomnamen verwijderen door bij de properties van de class *Adres* het attribuut *Column* te tikken:

```
using System.ComponentModel.DataAnnotations.Schema;
[ComplexType]
public class Adres
{
    [Column("Straat")]
    public string Straat { get; set; }
    [Column("HuisNr")]
    public string HuisNr { get; set; }
    [Column("PostCode")]
    public string PostCode { get; set; }
    [Column("Gemeente")]
    public string Gemeente { get; set; }
}
```

17.9 Inheritance

17.9.1 Table per hierarchy (TPH)

Deze manier om inheritance voor te stellen is de default manier in CodeFirst. Er is één table voor alle classes met een inheritance verband.

Je voegt volgende classes toe

```
using System.ComponentModel.DataAnnotations.Schema;
namespace CodeFirstCursus
{
    [Table("Cursussen")]
    public abstract class Cursus
    {
        public int Id { get; set; }
        public string Naam { get; set; }
    }
}
```

```
using System;
namespace CodeFirstCursus
{
    public class KlassikaleCursus : Cursus
    {
        public DateTime Van { get; set; }
        public DateTime Tot { get; set; }
    }
}
```

```
namespace CodeFirstCursus
{
    public class ZelfstudieCursus : Cursus
    {
        public int AantalDagen { get; set; }
    }
}
```

Je voegt volgende property toe aan de class *VDABContext*

```
public DbSet<Cursus> Cursussen { get; set; }
```


Je wijzigt de code in de *using* van de *Main* van *Program.cs*

```
context.Cursussen.Add(new KlassikaleCursus { Naam="Frans in 24 uur",
    Van=DateTime.Today, Tot=DateTime.Today});
context.Cursussen.Add(new ZelfstudieCursus { Naam = "Engels in 24 uur",
    AantalDagen=1});
context.SaveChanges();
```

Je kan het programma uitproberen.

De database wordt geschrapt en opnieuw aangemaakt.

De bijbehorende table:

Cursussen	
	Id
	Naam
	Van
	Tot
	AantalDagen
	Discriminator

CodeFirst heeft een kolom *Discriminator* toegevoegd om het onderscheid te maken tussen klassikale cursussen en zelfstudiecursussen.

CodeFirst vult deze kolom met de waarde *KlassikaleCursus* of *ZelfstudieCursus*.

Je kan de kolomnaam en de kolomwaarden instellen in je *DbContext* class

Je voegt volgende method toe aan *VDABContext*

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) (1)
{
    modelBuilder.Entity<KlassikaleCursus>()
        .Map(m => m.Requires("Soort").HasValue("K")); (2)
    modelBuilder.Entity<ZelfstudieCursus>()
        .Map(m => m.Requires("Soort").HasValue("Z"));
}
```

(1) CodeFirst voert deze method uit bij het aanmaken van de database.

(2) Met deze opdracht geef je de discriminator kolom de naam *Soort* en geef je aan dat deze kolom de waarde *K* moet bevatten bij een klassikale cursus.

Je kan het programma uitproberen.

17.9.2 Table per type (TPT)

Bij TPT is er één table per class in de inheritance structuur.

CodeFirst past TPT toe als je ook een *Table* attribuut tikt bij de derived classes

Je tikt voor de class *KlassikaleCursus* volgende regel:

```
[Table("Klassikalecursussen")] // using System.ComponentModel.DataAnnotations.Schema;
```

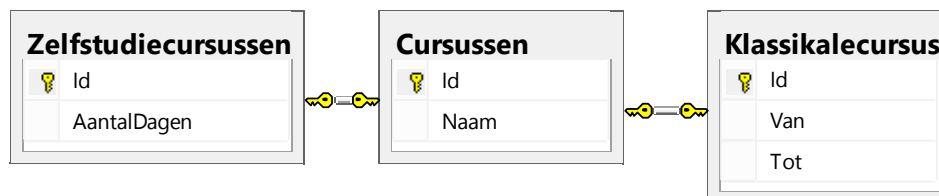
Je tikt voor de class *ZelfstudieCursus* volgende regel:

```
[Table("Zelfstudiecurssussen")] // using System.ComponentModel.DataAnnotations.Schema;
```

Je verwijdert in de class *VDABContext* de method *OnModelCreating*

Je kan het programma uitproberen. De database wordt geschrapt en opnieuw aangemaakt.

De bijbehorende tables:



17.9.3 Table per concret class (TPC)

Bij TPC is er één table per concrete subclass in de inheritance structuur.

CodeFirst past TPC toe als de primary key geen int met autonumber is.

Je verwijdert het attribuut *Table* bij de class *Cursus* en je wijzigt de property *Id* van die class

```
[DatabaseGenerated(DatabaseGeneratedOption.Identity)] (1)
public Guid Id { get; set; } // using System; (2)
```

(1) Je geeft aan dat de inhoud van de primary key bij een nieuw record door de database wordt ingevuld (en niet in je code).


(2) Je kiest *Guid* als type van de property.
SQL Server zal bij een nieuw record een unieke string invullen in de primary key.


Je voegt aan de class *VDABContext* de method *onModelCreating* toe

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Entity<KlassikaleCursus>().Map(m => m.MapInheritedProperties()); (1)
    modelBuilder.Entity<ZelfstudieCursus>().Map(m => m.MapInheritedProperties());
}
```

(1) Je geeft met de method *MapInheritedProperties* aan dat Code First alle properties (ook de geërfde properties) in de table moet plaatsen die bij de class hoort.

Je kan het programma uitproberen.
De bijbehorende tables:

Klassikalecursus	
	Id
	Naam
	Van
	Tot

Zelfstudiecurssussen	
	Id
	Naam
	AantalDagen

17.10 Associaties tussen entities

17.10.1 Één op veel associaties

Je voegt een één op veel relatie toe tussen *Campus* en *Instructeur*

Je voegt volgende property toe aan de class *Campus*

```
public virtual ICollection<Instructeur> Instructeurs { get; set; }
// using System.Collections.Generic; (1)
```

- (1) Deze property stelt de verzameling instructeurs voor die bij de huidige campus horen. Het type van zo'n property moet *ICollection* zijn bij code first. Je kan met *foreach* itereren over een *ICollection*. Zo'n property moet *virtual* zijn, anders werkt lazy loading niet.

Je voegt volgende property toe aan de class *Instructeur*

```
public virtual Campus Campus { get; set; } (1)
public int CampusId { get; set; } (2)
```

- (1) Deze property stelt de campus voor die bij de instructeur hoort. Zo'n property moet *virtual* zijn, anders werkt lazy loading niet.
- (2) Deze property stelt het campusnummer voor van de campus die bij de instructeur hoort. De property naam moet gelijk zijn aan de property die in *Campus* de primary key voorstelt.

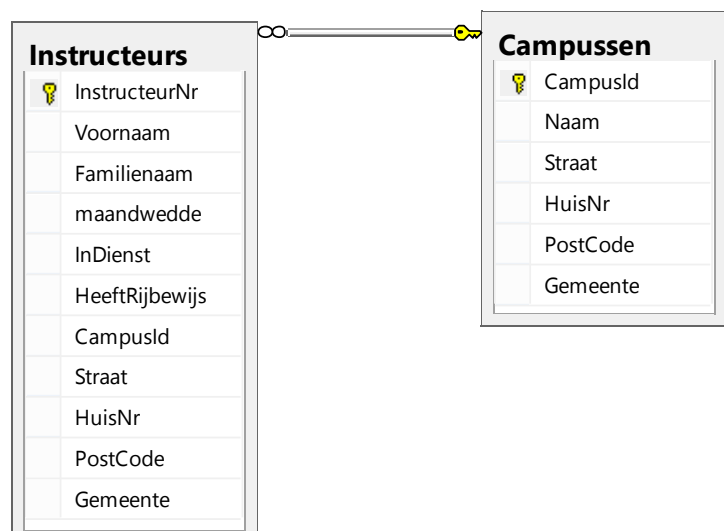
Je wijzigt de code in de *using* van de *Main* van *Program.cs*

```
var campus = new Campus { Naam = "Delos",
    Adres = new Adres { Straat = "Vlamingstraat", HuisNr = "10", PostCode = "8560",
        Gemeente = "Wevelgem" } };
var jean = new Instructeur { Voornaam = "Jean", Familienaam = "Smits", Wedde = 1000,
    InDienst = new DateTime(1966, 8, 1), HeeftRijbewijs = true,
    Adres = new Adres { Straat = "Keizerslaan", HuisNr = "11", PostCode = "1000",
        Gemeente = "Brussel" },
    Campus = campus };
context.Campussen.Add(campus);
context.Instructeurs.Add(jean);
context.SaveChanges();
```

Je kan het programma uitproberen.

De database wordt geschrapt en opnieuw aangemaakt.

De table *Instructeurs* bevat nu een foreign-key kolom *CampusId*, die verwijst naar de table *Campussen*:



17.10.2 Veel-op-veel associaties

Als twee classes een veel-op-veel associatie hebben, bevatten beide classes een *ICollection* property die verwijst naar de andere class.

Het voorbeeld is een veel-op-veel associatie tussen verantwoordelijkheden en instructeurs.

Je voegt een class *Verantwoordelijkheid* toe

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace CodeFirstCursus
{
    [Table("Verantwoordelijkheden")]
    public class Verantwoordelijkheid
    {
        public int VerantwoordelijkheidId { get; set; }
        public string Naam { get; set; }
        public virtual ICollection<Instructeur> Instructeurs { get; set; }
    }
}
```

Je voegt aan de class *Instructeur* een property *Verantwoordelijkheden* toe

```
public virtual ICollection<Verantwoordelijkheid> Verantwoordelijkheden { get; set; }
// using System.Collections.Generic;
```

Je voegt volgende property toe aan de class *VDABContext*

```
public DbSet<Verantwoordelijkheid> Verantwoordelijkheden { get; set; }
```

Je voegt volgende code toe in *Program.cs*, voor *context.SaveChanges()*;

```
var verantwoordelijkheid = new Verantwoordelijkheid { Naam = "EHBO" };
jean.Verantwoordelijkheden =
    new List<Verantwoordelijkheid> { verantwoordelijkheid };
// using System.Collections.Generic;
context.Verantwoordelijkheden.Add(verantwoordelijkheid);
```

Je kan het programma uitproberen. De database wordt geschrapt en opnieuw aangemaakt.



Je kan de naam van de tussentabel en de namen van de kolommen in die tussentabel instellen, met een extra opdracht in de method *onModelCreating* van de class *VDABContext*

```
modelBuilder.Entity<Instructeur>()
    .HasMany(i => i.Verantwoordelijkheden)
    .WithMany(v => v.Instructeurs)
    .Map(c => c.ToTable("InstructeursVerantwoordelijkheden"))
    .MapLeftKey("VerantwoordelijkheidID")
    .MapRightKey("InstructeurNr");
```

(1) Je vermeldt bij *Entity* tussen < en > de naam van één van beide classes.

(2) Je vermeldt bij *HasMany* de property in de class bij (1) die verwijst naar de tweede class.

- (3) Je vermeldt bij *WithMany* de property in de tweede class die verwijst naar de class bij (1)
- (4) Je vermeldt bij *ToTable* de naam van de tussentabel.
- (5) Je vermeldt bij *MapLeftKey* de kolomnaam die hoort bij de tweede class in de tussentabel.
- (6) Je vermeldt bij *MapRightKey* de kolomnaam die hoort bij de class bij (1) in de tussentabel.

Je kan het programma uitproberen.

De database wordt geschrapt en opnieuw aangemaakt.



17.10.3 Een associatie naar dezelfde tabel

Het voorbeeld is een class *Cursist*. Iedere cursist heeft een mentor, een mentor kan meerdere cursisten als beschermeling hebben.

Je voegt de class *Cursist* toe

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace CodeFirstCursus
{
    [Table("Cursisten")]
    public class Cursist
    {
        [Key]
        public int CursistId { get; set; }
        public string Voornaam { get; set; }
        public string Familienaam { get; set; }
        public virtual ICollection<Cursist> Beschermelingen { get; set; }      (1)
        [InverseProperty("Beschermelingen")]                               (2)
        public virtual Cursist Mentor { get; set; }                         (3)
    }
}
```

- (1) Deze property stelt de beschermelingen van één mentor voor.
- (2) De property bij (3) stelt de mentor van een cursist voor. Je vermeldt bij *InverseProperty* de property (1) die de andere kant van de associatie voorstelt.

Je voegt volgende property toe aan de class *VDABContext*

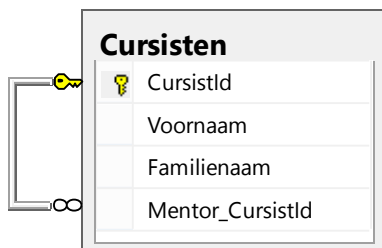
```
public DbSet<Cursist> Cursisten { get; set; }
```

Je wijzigt de code in de *using* van de *Main* van *Program.cs*

```
Cursist joe = new Cursist { Voornaam = "Joe", Familienaam = "Dalton", };
Cursist averell = new Cursist { Voornaam = "Averell", Familienaam = "Dalton",
    Mentor = joe };
context.Cursisten.Add(joe);
context.Cursisten.Add(averell);
context.SaveChanges();
```

Je kan het programma uitproberen. De database wordt geschrapt en opnieuw aangemaakt.

Je krijgt volgende nieuwe table:

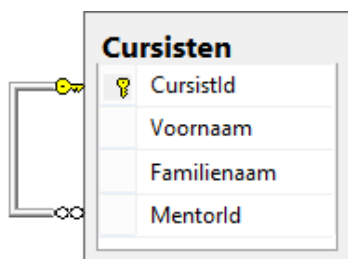


Je ziet dat EF zelf een kolomnaam verzonnen heeft die hoort bij de private variabele Mentor. Je kan deze kolomnaam zelf kiezen met het attribuut ForeignKey.

Je tikt [`ForeignKey("MentorId")`] voor de private variabele Mentor. en je voegt volgende private variabele toe:

```
public int? MentorId { get; set; }
```

Als je nu het programma uitvoert, ziet de table er als volgt uit:



Code first: zie takenbundel

18 WPF

Om deze cursus af te ronden gebruik je EF in een WPF applicatie.

Je kan daarbij zowel edmx als code first gebruiken. Je gebruikt in het voorbeeld code first.

Je gebruikt in het voorbeeld de table Cursisten van de database Opleidingen

18.1 De entity classes voor de nieuwe database

Je maakt in Visual Studio een WPF project met de naam *EFInWPF*.

Je downloadt de laatste versie van de EF library en gebruikt die in het project

- Je klikt met de rechtermuisknop op het project in de Solution Explorer
- Je kiest *Manage NuGet Packages*
- Je kiest links *Online*
- Je kiest rechts *EntityFramework*
- Je kiest *Install*

Je voegt een entity class *Cursist* toe.

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace EFInWPF
{
    [Table("Cursisten")]
    public class Cursist
    {
        [Key]
        public int CursistNr { get; set; }
        public string Voornaam { get; set; }
        public string Familienaam { get; set; }
        public virtual ICollection<Cursist> Beschermelingen { get; set; }
        [InverseProperty("Beschermelingen")]
        [ForeignKey("MentorNr")]
        public virtual Cursist Mentor { get; set; }
        public int? MentorNr { get; set; }
        public string Naam
        {
            get
            {
                return Voornaam + ' ' + Familienaam;
            }
        }
    }
}
```

18.2 De DbContext class

Je voegt een class *VDABContext* toe

```
using System.Data.Entity;
namespace EFInWPF
{
    public class VDABContext : DbContext
    {
        public DbSet<Cursist> Cursisten { get; set; }
    }
}
```

18.3 De connectionstring

Je maakt in *App.config* een connectionstring naar de database, onder `</startup>`

```
<connectionStrings>
  <add name="VDABContext"
        providerName="System.Data.SqlClient"
        connectionString=
          "Server=.\SQLEXPRESS;Database=Opleidingen;Trusted_Connection=true;" />
</connectionStrings>
```

18.4 Een instantie van de DbContext class

Je voegt aan *MainWindow.xaml.cs* een instantie toe van de *VDABContext* class

```
private VDABContext context = new VDABContext();
```

Je sluit deze *VDABContext* wanneer de gebruiker het venster sluit

```
protected override void OnClosing(System.ComponentModel.CancelEventArgs e)
{
    context.Dispose();
}
```

18.5 Een ListBox tonen met data uit de database

Je toont in het venster de namen van cursisten die zelf mentor zijn.

Je voegt daartoe volgende regels toe aan *MainWindow.xaml*, onder `<Grid>`

```
<ListBox Name="listMentors" DisplayMemberPath="Naam" />
```

Je voegt volgende method toe aan de class *MainWindow*.

Je zoekt in die code de cursisten die mentor zijn.

Je maakt van die cursisten een List en gebruikt die als bron voor de ListBox

```
private void VullListMentors()
{
    listMentors.ItemsSource = (from cursist in context.Cursisten
                               where cursist.Beschermelingen.Count() != 0
                               orderby cursist.Voornaam, cursist.Familienaam
                               select cursist).ToList();
}
```

Je roept die method op in de constructor van de class *MainWindow*.

```
public MainWindow()
{
    InitializeComponent();
    VullListMentors();
}
```

Je kan de applicatie uitproberen.

18.6 Een ListBox met gerelateerde data tonen

Als de gebruiker in de ListBox een mentor selecteert,

toon je in een tweede ListBox de beschermelingen van die mentor

Je vervangt in *MainWindow.xaml* de regel `<ListBox .../>` door

```
<Grid.RowDefinitions>
  <RowDefinition Height="*" />
  <RowDefinition Height="5" />
  <RowDefinition Height="*" />
</Grid.RowDefinitions>
<ListBox Name="listMentors" DisplayMemberPath="Naam" Grid.Row="0"
        SelectionChanged="listMentorsSelectionChanged" />
<GridSplitter Grid.Row="1" HorizontalAlignment="Stretch" />
<ListBox Name="listBeschermelingen" DisplayMemberPath="Naam" Grid.Row="2" />
```

Je voegt volgende method toe aan de class *MainWindow*.

Als de gebruiker in de ListBox met mentors een item selecteert, vraag je de beschermelingen van die mentor en je gebruikt die beschermelingen als bron voor de tweede ListBox

```
private void ListMentorsSelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (listMentors.SelectedItem != null)
    {
        var mentor = (Cursist)listMentors.SelectedItem;
        listBeschermelingen.ItemsSource = mentor.Beschermelingen;
    }
}
```

Je kan de applicatie uitproberen.

18.7 Een DataGrid tonen met data uit de database

Je zal de beschermelingen tonen in een DataGrid.

Je vervangt in *MainWindow.xaml* de regel `<ListBox Name="listBeschermelingen" ... />` door

```
<DataGrid Name="gridBeschermelingen" Grid.Row="2" IsReadOnly="True"
    AutoGenerateColumns="False">
    <DataGrid.Columns>
        <DataGridTextColumn Binding="{Binding Voornaam}" Header="Voornaam" Width="*" />
        <DataGridTextColumn Binding="{Binding Familiennaam}" Header="Familiennaam"
            Width="*" />
    </DataGrid.Columns>
</DataGrid>
```

Je vervangt in de method *ListMentorsSelectionChanged* van de class *MainWindow* *listBeschermelingen* door *gridBeschermelingen*

Je kan de applicatie uitproberen.

18.8 Data wijzigen

De gebruiker zal de voornaam en/of familiennaam van beschermelingen kunnen corrigeren.

Hij drukt daarna op een knop *Opslaan* in het menu om deze correcties in de database op te slaan.

Je doet volgende wijzigingen in *MainWindow.xaml*

- Je verwijdert in de regel `<DataGrid ...> IsReadOnly="True"`
- Je tikt voor `<Grid>` volgende regels


```
<DockPanel>
    <Menu DockPanel.Dock="Top">
        <MenuItem Header="Opslaan" Click="Opslaan" />
    </Menu>
```
- Je tikt na `</Grid>` volgende regel


```
</DockPanel>
```

Je voegt volgende method toe aan de class *MainWindow*.

Deze method wordt opgeroepen als de gebruiker in het menu *Opslaan* kiest

```
private void Opslaan(object sender, RoutedEventArgs e)
{
    context.SaveChanges();
    // Opdat de wijzigingen die je in de grid deed ook zichtbaar zijn in de list:
    VullistMentors();
}
```

Je wijzigt in de class *Cursist ICollection* naar *ObservableCollection*.

Het type *ObservableCollection* stelt, zoals *ICollection* een verzameling voor.

Je kan data in een *ICollection* tonen in WPF controls, maar niet wijzigen.

Je kan data in een *ObservableCollection* in WPF controls tonen én wijzigen.

Je kan de applicatie uitproberen.

19 COLOFON

Sectorverantwoordelijke:	Ortaire Uyttersprot
Cursusverantwoordelijke:	Jean Smits
Medewerkers:	Hans Desmet
Versie:	2/4/2015
Nummer dotatielijst:	