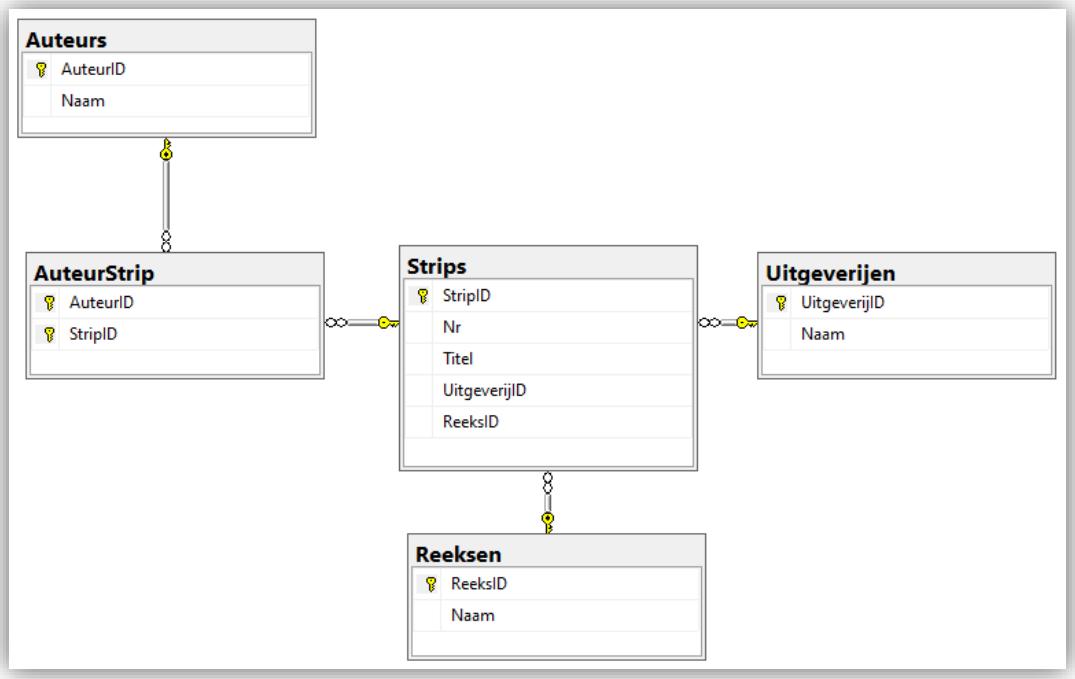
EF – CRUD

Datamodel

In dit hoofdstuk behandelen we CRUD (Create Read Update Delete) operaties met Entity Framework Core. We gaan dit doen aan de hand van een voorbeeld waarbij we het volgende databank schema gaan gebruiken. Daarin vinden we twee 1-op-veel relaties zoals de link tussen strips en uitgeverijen, een strip kan worden uitgegeven door slechts één uitgeverij, maar een uitgeverij kan wel meerdere strips uitgeven. De tweede 1-op-veel relatie is deze tussen strips en reeksen, een strip behoort maar tot één reeks terwijl een reeks wel meerdere strips kan bevatten. Er is ook een veel-op-veel relatie, namelijk tussen auteurs en strips, een auteur kan meerdere strips maken en een strip kan ook door meerdere auteurs worden gemaakt.



De klassen die we hiervoor gaan gebruiken zijn om te beginnen de klasse Uitgeverij. Een zeer eenvoudige klasse met een ID en naam als properties.

```
public class Uitgeverij
{
    1 reference
    public Uitgeverij(string naam)...

    0 references
    public int UitgeverijID { get; set; }
    1 reference
    public string Naam { get; set; }
}
```

Een tweede klasse is Reeks, ook hier voorzien we als properties een ID en een naam. Daarnaast bevat een reeks ook een collectie van strips.

```
public class Reeks
{
    1 reference
    public Reeks(string naam)...

    0 references
    public int ReeksID { get; set; }
    1 reference
    public string Naam { get; set; }
    0 references
    public ICollection<Strip> Strips { get; set; } = new List<Strip>();
}
```

Voor de veel-op-veel relatie is het in EF Core noodzakelijk om voor de koppeltabel ook een klasse te voorzien. We hebben deze AuteurStrip genoemd en deze bevat naast de IDs naar zowel de auteur als de strip ook een verwijzing naar de objecten zelf.

```
public class AuteurStrip
{
    1 reference
    public AuteurStrip(Auteur auteur, Strip strip)...

    0 references
    public AuteurStrip(int auteurID, int stripID)...

    2 references
    public int AuteurID { get; set; }
    1 reference
    public Auteur Auteur { get; set; }
    2 references
    public int StripID { get; set; }
    1 reference
    public Strip Strip { get; set; }
}
```

De klasse Auteur implementeert de veel-op-veel relatie door middel van een collectie AuteurStrip objecten.

```
public class Auteur
{
    1 reference
    public Auteur(string naam)...
    0 references
    public int AuteurID { get; set; }
    1 reference
    public string Naam { get; set; }
    0 references
    public ICollection<AuteurStrip> StripsLink { get; set; } = new List<AuteurStrip>();
}
```

De klasse Strip implementeert de veel-op-veel relatie op dezelfde wijze als de Auteur klasse. Daarnaast vinden we ook referenties naar de Uitgeverij en de Reeks. We voorzien hier ook een methode om een auteur toe te voegen die dan een koppelobject toevoegt aan de collectie AuteursLink.

```
public class Strip
{
    2 references
    public Strip(int nr, string titel)...
    0 references
    public Strip(int nr, string titel, Uitgeverij uitgever, Reeks reeks) ...
    0 references
    public int StripID { get; set; }
    1 reference
    public int Nr { get; set; }
    1 reference
    public string Titel { get; set; }
    2 references
    public Uitgeverij Uitgever { get; set; }
    2 references
    public Reeks Reeks { get; set; }
    1 reference
    public ICollection<AuteurStrip> AuteursLink { get; set; } = new List<AuteurStrip>();
    1 reference
    public void VoegAuteurToe(Auteur auteur)
    {
        AuteursLink.Add(new AuteurStrip(auteur,this));
    }
}
```

Voor de link met de databank maken we gebruik van de klasse StripContext, die overerft van DbContext en waarin we via de verschillende DbSets de toegang tot de tabellen regelen. Aangezien we met een koppeltabel werken is er een samengestelde sleutel nodig die we definiëren in de OnModelCreating methode.

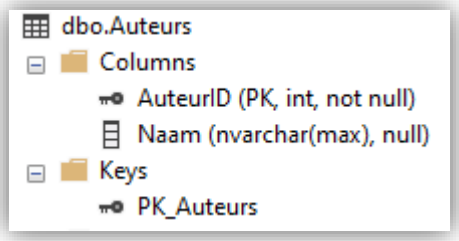
```
public class StripsContext : DbContext
{
    1 reference
    public DbSet<Strip> Strips { get; set; }
    0 references
    public DbSet<Auteur> Auteurs { get; set; }
    0 references
    public DbSet<Reeks> Reeksen { get; set; }
    0 references
    public DbSet<Uitgeverij> Uitgeverijen { get; set; }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Data Source=lenovo-pc\sqlexpress;Initial Catalog=stripsDB;Integrated Security=True");
    }
    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<AuteurStrip>().HasKey(x => new { x.StripID, x.AuteurID });
    }
}
```

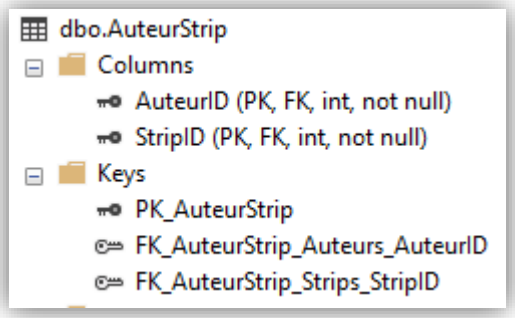
Het aanmaken van het databank model kan dan via de migration tools, zoals verder aangegeven.

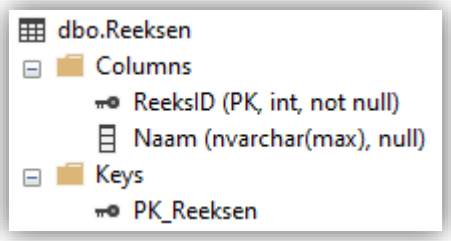
```
PM> Add-Migration InitStrips -project EFcrud
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
```

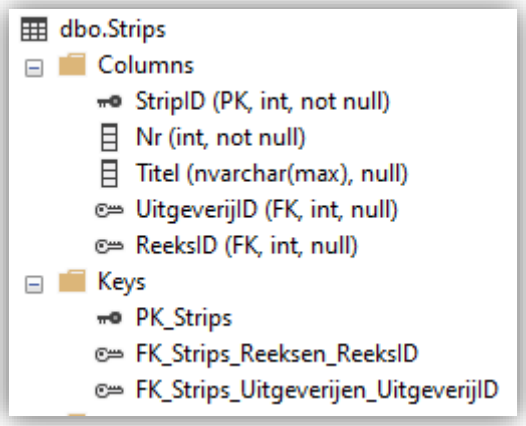
```
PM> Update-Database -project EFcrud
Build started...
Build succeeded.
Applying migration '20200330120110_InitStrips'.
Done.
```

Het resultaat zijn de volgende tabellen met aangegeven kolommen en sleutels :









Create

De eerste actie die we implementeren is het toevoegen van gegevens in de databank. We gaan daarbij gebruik maken van een tekstbestand met daarin de nodige info voor de aanmaak van de objecten. De eerste parameter is de titel van de strip, daarna het nummer, de naam van de reeks, de auteurs (indien meerdere gescheiden door een komma) en de naam van de uitgeverij.

```
De Zwarte Rotsen;7;Kuifje;Hergé;Casterman
De Zonnetempel;14;Kuifje;Hergé;Casterman
Het Sterrenkind;7;Thorgal;Rosinski,Van Hamme;Le Lombard
Billy the Kid;20;Lucky Luke;Goscinny,Morris;Dupuis
Het geheim van de Eenhoorn;11;Kuifje;Hergé;Casterman
Lucky Luke tegen Pat Poker;5;Lucky Luke;Morris;Dupuis
Calamity Jane;30;Lucky Luke;Goscinny, Morris;Dupuis
In het spoor van de Daltons;17;Lucky Luke;Goscinny,Morris;Dupuis
Aniël;36;Thorgal;Rosinski,Yann;Le Lombard
Scharlaken vuur;35;Thorgal;Dorison,Rosinski;Le Lombard
De onzichtbare vesting;19;Thorgal;Rosinski, Van Hamme;Le Lombard
De ijskoningin;1;Thorgal;Rosinski, Van Hamme;Le Lombard
Drie grijsaards in het land van Aran;3;Thorgal;Rosinski,Van Hamme;Le Lombard
```

We implementeren daarvoor de methode InitialiseerDatabank, waarbij we een aantal dictionaries gebruiken om de verschillende objecten (Uitgeverij, Reeks, Auteur) bij te houden en een HashSet voor de Strips.

```
public void InitialiseerDatabank(string path)
{
    Dictionary<string, Uitgeverij> uitgeverijDict = new Dictionary<string, Uitgeverij>();
    Dictionary<string, Reeks> reeksDict = new Dictionary<string, Reeks>();
    Dictionary<string, Auteur> auteurDict = new Dictionary<string, Auteur>();
    HashSet<Strip> stripSet = new HashSet<Strip>();
    using (StreamReader r = new StreamReader(path))...
    using (StripsContext ctx = new StripsContext())...
    Console.WriteLine("Einde DB initialisatie");
}
```

Het eerste deel van de methode bevat code voor het lezen en analyseren van het tekstbestand. We lezen lijn per lijn in, splitsen op basis van het scheidingsteken ';' en kennen de waarden toe aan de variabelen. Komen we een 'Reeks' tegen voor de eerste keer dan maken we een entry in onze dictionary reeksDict en voor uitgeverij en auteur doen we hetzelfde. Daarna maken we een object Strip aan met de verwijzingen naar de objecten uit onze dictionaries. *Het is belangrijk dat de links/referenties van de verschillende Strip-objecten verwijzen naar dezelfde objecten (Uitgeverij, Auteur en Reeks), indien dit niet het geval zou zijn dan worden er telkens nieuwe entries in de databank gemaakt met dezelfde inhoud.*

```
using (StreamReader r = new StreamReader(path))
{
    string line;            string titel;
    int nr;                 string reeks;
    string uitgeverij;      string[] auteurs;
    while ((line = r.ReadLine()) != null)
    {
        string[] ss = line.Split(';').Select(x => x.Trim()).ToArray();
        titel = ss[0];
        nr = int.Parse(ss[1]);
        reeks = ss[2];
        auteurs = ss[3].Split(',').Select(x => x.Trim()).ToArray();
        uitgeverij = ss[4];
        if (!reeksDict.ContainsKey(reeks)) reeksDict.Add(reeks, new Reeks(reeks));
        if (!uitgeverijDict.ContainsKey(uitgeverij))
            uitgeverijDict.Add(uitgeverij, new Uitgeverij(uitgeverij));
        foreach (string auteur in auteurs) {
            if (!auteurDict.ContainsKey(auteur))  auteurDict.Add(auteur, new Auteur(auteur));
        }
        Strip s = new Strip(nr, titel);
        s.Reeks = reeksDict[reeks];
        s.Uitgever = uitgeverijDict[uitgeverij];
        foreach (string auteur in auteurs) {
            s.VoegAuteurToe(auteurDict[auteur]);
        }
        stripSet.Add(s);
    }
}
```

Het wegschrijven van de objecten naar de databank is wel vrij eenvoudig. De objecten zijn allemaal aangemaakt in het geheugen en met elkaar verbonden. Het wegschrijven kan door gebruik te maken van de methode AddRange van de DbSet<Strip> property waar we de strips toevoegen en daarna het oproepen van de SaveChanges methode van onze context.

```
using (StripsContext ctx = new StripsContext())
{
    ctx.Strips.AddRange(stripSet);
    ctx.SaveChanges();
}
```

Het resultaat is dat alle tabellen worden ingevuld met hun relaties.

```
SELECT * FROM [stripsDB].[dbo].[Strips]
SELECT * FROM [stripsDB].[dbo].[Auteurs]
SELECT * FROM [stripsDB].[dbo].[AuteurStrip]
SELECT * FROM [stripsDB].[dbo].[Reeksen]
SELECT * FROM [stripsDB].[dbo].[Uitgeverijen]
```

StripID	Nr	Titel	UitgeverijID	ReeksID
11	35	Scharlaken vuur	5	5
12	19	De onzichtbare vesting	5	5
13	1	De ijskoningin	5	5
14	3	Drie grijsaards in het land van Aran	5	5

AuteurID	Naam
6	Hergé
7	Rosinski
8	Van Hamme
9	Goscinny

AuteurID	StripID
9	17
9	18

ReeksID	Naam
4	Kuifje
5	Thorgal
6	Luck...

UitgeverijID	Naam
4	Casterman
5	Le Lombard

Read

Het opvragen van gegevens met EF is vrij eenvoudig. We kunnen daarbij gebruik maken van de DbSet properties in onze context klasse. In het volgende voorbeeld vragen we alle strips op door de property Strips in StripContext. Met een foreach lus kunnen we ze allemaal overlopen en op het scherm afdrukken.

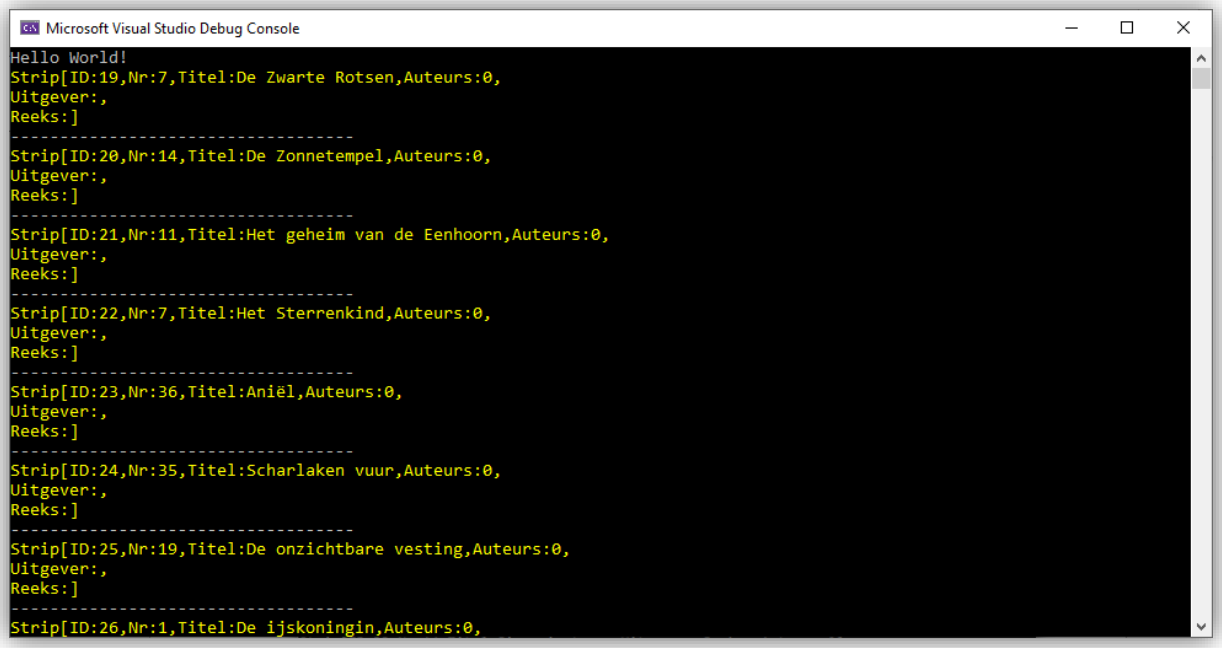
```
public void ToonStrips()
{
    using (StripsContext ctx = new StripsContext())
    {
        foreach (Strip x in ctx.Strips)
        {
            x.ToonDetail();
            Console.WriteLine("-----------------------------------");
        }
    }
}
```

Er is een methode ToonDetail gemaakt waarmee we de Strip info afdrukken, gebruik makend van de ToString methode van Strip zelf (zie volgend code fragment) en we doorlopen ook de veel-op-veel relatie AuteursLink.

```
public void ToonDetail()
{
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine(this);
    Console.ResetColor();
    foreach (var al in AuteursLink)
    {
        Console.WriteLine(al.Auteur);
    }
}
```

```
public override string ToString()
{
    return $"Strip[ID:{StripID},Nr:{Nr},Titel:{Titel},Auteurs:{AuteursLink.Count},\nUitgever:{Uitgever},\nReeks:{Reeks}]";
}
```

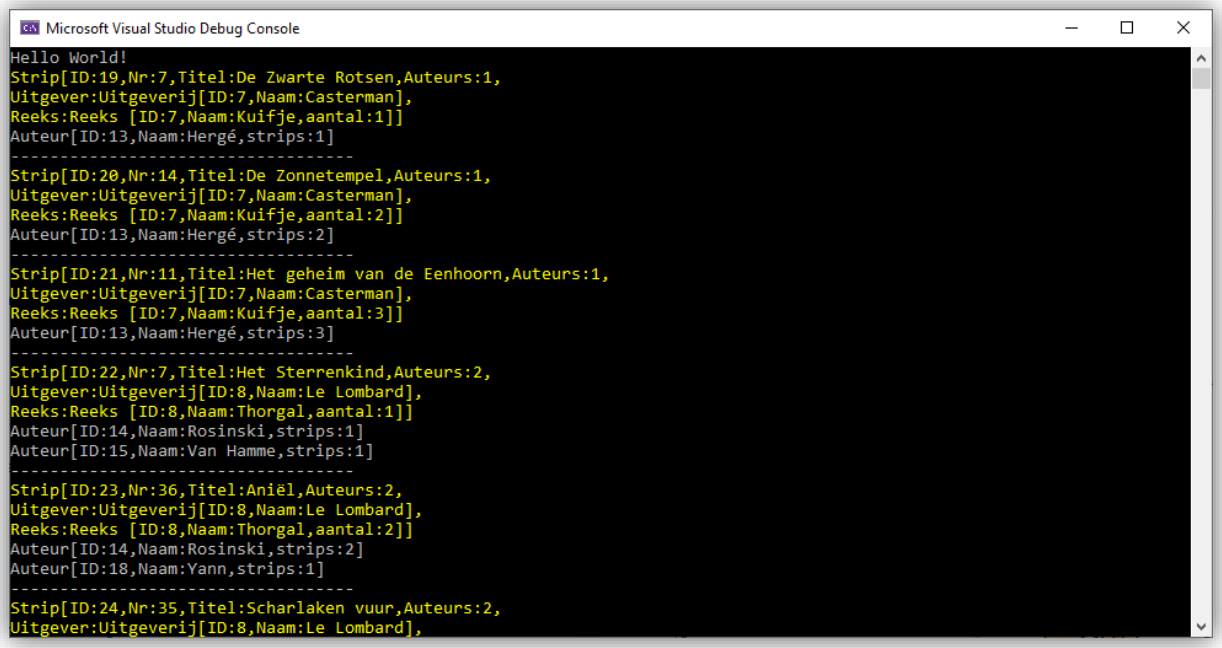
Bekijken we het resultaat dan zien we wel alle strips, maar info over uitgeverij, reeks en auteurs ontbreekt. Het opvragen van een object Strip impliceert niet dat alle relaties worden mee opgenomen.



Om wel alle relaties op te vragen moeten we gebruik maken van de Include methode. Het volgende code fragment toont op welke manier dit kan. Via Include(s=>s.Reeks) voegen we expliciet de info over de Reeks toe, via Include(s=>s.Uitgever) de info over de uitgever en de info over de Auteurs wordt bekomen via de veel-op-veel relatie/klasse AuteursLink. Voor deze laatset gebruiken we eerst Include(s=>s.AuteursLink) om de AuteurLink objecten te laden en om van hieruit verder door te schakelen naar Auteur voegen we daar nog aan toe ThenInclude(l=>l.Auteur). Op deze manier worden alle referenties opgehaald.

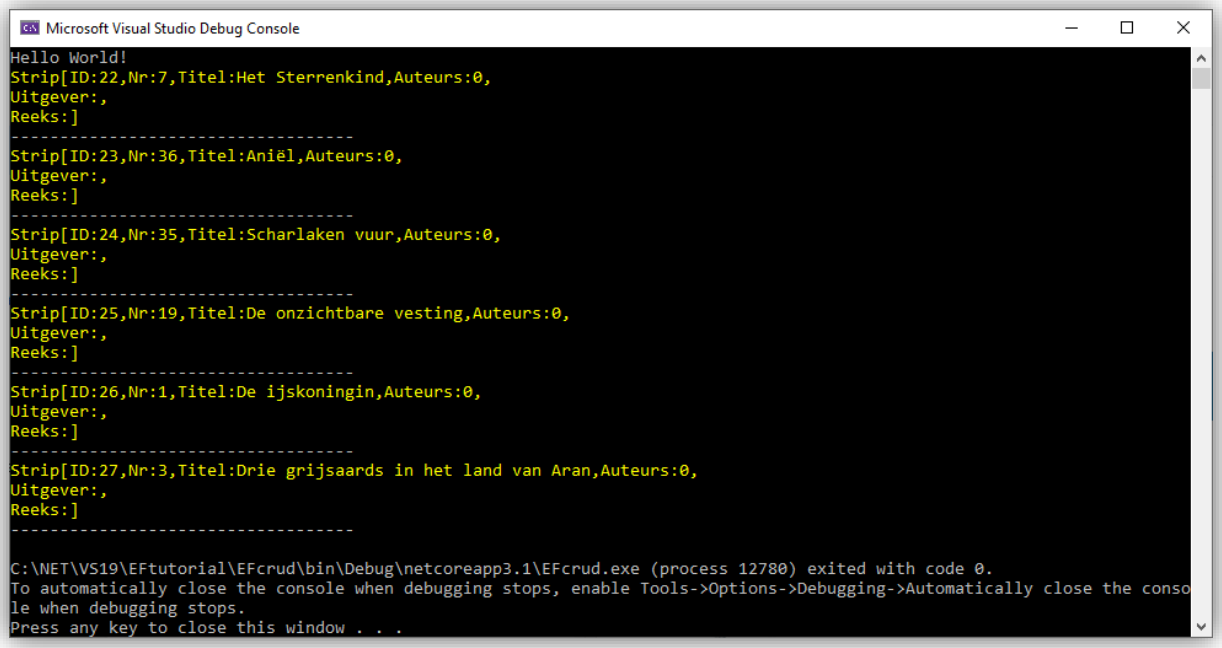
```
public void ToonStripsInclude()
{
    using (StripsContext ctx = new StripsContext())
    {
        foreach (Strip x in ctx.Strips.Include(s=>s.Reeks)
            .Include(s=>s.Uitgever)
            .Include(s=>s.AuteursLink).ThenInclude(l=>l.Auteur))
        {
            x.ToonDetail();
            Console.WriteLine("-----------------------------------");
        }
    }
}
```

Bekijken we het resultaat dan zien we nu inderdaad alle info die gerelateerd is aan het object Strip.



Voor het opvragen van gegevens kan er ook gebruik gemaakt worden van filters. Dit is volledig analoog met wat we in LINQ hebben gedaan en gaan we dan ook niet in detail bekijken. We tonen hier enkel een voorbeeld waarbij enkel de strips uit de reeks Thorgal worden opgevraagd.

```
public void ToonStripsFilter()
{
    using (StripsContext ctx = new StripsContext())
    {
        foreach (Strip x in ctx.Strips.Where(s=>s.Reeks.Naam=="Thorgal"))
        {
            x.ToonDetail();
            Console.WriteLine("-----------------------------------");
        }
    }
}
```



We hebben reeds gezien (deel EF – Hoe werkt het ?) dat EF alle opgevraagde objecten trackt (bijhoud wat hun status is). Wanneer we dat niet wensen, bijvoorbeeld we halen enkel gegevens op om te tonen, dan kunnen we dat ook aangeven door de methode AsNoTracking() aan te roepen. In dat gevalworden wel alle gegevens opgehaald, maar zal EF die niet opvolgen en besparen we dus heel wat geheugen. In het volgende code fragment zien we hoe we deze methode kunnen gebruiken.

```
public void ToonStripsIncludeAsNoTracking()
{
    using (StripsContext ctx = new StripsContext())
    {
        foreach (Strip x in ctx.Strips.Include(s => s.Reeks)
            .Include(s => s.Uitgever)
            .Include(s => s.AuteursLink).ThenInclude(l => l.Auteur)
            .AsNoTracking())
        {
            x.ToonDetail();
            Console.WriteLine("-----------------------------------");
        }
    }
}
```

Update

Voor het update van een object met EF maken we gebruik van de tracking die is ingebouwd in EF Core. Het komt er op neer dat we een object opvragen, het aanpassen en daarna de methode SaveChanges oproepen van onze context klasse. Aangezien EF zelf bijhoudt wat de veranderingen zijn hoeven we dat zelf niet expliciet te doen en volstaat het om dit over te laten aan EF. We tonen dit aan met een eenvoudig voorbeeld. We zullen de strip met ID 26 opvragen en aanpassen.

De beginsituatie is als volgt :

StripID	Nr	Titel	UitgeverijID	ReeksID
24	35	Scharlaken vuur	8	8
25	19	De onzichtbare vesting	8	8
26	1	De ijskoningin	8	8
27	3	Drie grijsaards in het land van Aran	8	8
28	20	Billy the Kid	9	9

In het code fragment vragen we deze strip op, passen de titel en het nummer aan en bewaren de veranderingen.

```
public void UpdateStrip()
{
    using (StripsContext ctx = new StripsContext())
    {
        Strip strip = ctx.Strips.Single(x => x.StripID == 26);
        strip.Titel = "Het sterrenkind";
        strip.Nr = 7;
        ctx.SaveChanges();
    }
}
```

Het resultaat is dan :

StripID	Nr	Titel	UitgeverijID	ReeksID
24	35	Scharlaken vuur	8	8
25	19	De onzichtbare vesting	8	8
26	7	Het sterrenkind	8	8
27	3	Drie grijsaards in het land van Aran	8	8
28	20	Billy the Kid	9	9

Delete

Voor het verwijderen van elementen uit de databank maken we gebruik van de Remove methode van de DbSet property. We demonstreren dit aan de hand van een voorbeeld. We starten met de volgende tabel die 16 rijen bevat.

StripID	Nr	Titel	UitgeverijID	ReeksID
19	7	De Zwarte Rotsen	7	7
20	14	De Zonnetempel	7	7
21	11	Het geheim van de Eenhoorn	7	7
22	7	Het Sterrenkind	8	8
23	36	Aniël	8	8
24	35	Scharlaken vuur	8	8
25	19	De onzichtbare vesting	8	8
26	7	Het sterrenkind	8	8
27	3	Drie grijsaards in het land van Aran	8	8
28	20	Billy the Kid	9	9
29	5	Lucky Luke tegen Pat Poker	9	9
30	30	Calamity Jane	9	9
31	17	In het spoor van de Daltons	9	9
32	15	Alex	9	10
33	14	Simon is terug	9	10
34	11	Delta	9	10

Er zijn nu twee manieren om een element uit deze tabel te verwijderen. Een eerste methode bestaat eruit om het object (Strip) eerst op te vragen en het daarna als parameter mee te geven aan de Remove methode. Dit vraag echter het uitvoeren van twee SQL statements, namelijk een eerste voor het element op te vragen en een tweede om het te verwijderen. Er is echter ook een tweede mogelijkheid om het element te verwijderen, namelijk door een object Strip mee te geven aan de Remove methode die enkel de PK (StripID) bevat. Uiteindelijk kijkt EF enkel naar de PK van het meegegeven object. Op deze manier is er maar één SQL statement nodig om het element te verwijderen.

```
public void VerwijderStrip()
{
    using (StripsContext ctx = new StripsContext())
    {
        Strip strip = ctx.Strips.Single(x => x.StripID == 26);
        ctx.Strips.Remove(strip);

        ctx.Strips.Remove(new Strip() { StripID = 27 });
        ctx.SaveChanges();
    }
}
```

Het resultaat ziet er dan als volgt uit, de twee rijen zijn verwijderd uit de tabel.

StripID	Nr	Titel	UitgeverijID	ReeksID
19	7	De Zwarte Rotsen	7	7
20	14	De Zonnetempel	7	7
21	11	Het geheim van de Eenhoorn	7	7
22	7	Het Sterrenkind	8	8
23	36	Aniël	8	8
24	35	Scharlaken vuur	8	8
25	19	De onzichtbare vesting	8	8
28	20	Billy the Kid	9	9
29	5	Lucky Luke tegen Pat Poker	9	9
30	30	Calamity Jane	9	9
31	17	In het spoor van de Daltons	9	9
32	15	Alex	9	10
33	14	Simon is terug	9	10
34	11	Delta	9	10

Referenties

https://www.thereformedprogrammer.net/updating-many-to-many-relationships-in-entity-framework-core/

https://docs.microsoft.com/en-us/ef/core/modeling/

https://docs.microsoft.com/en-us/ef/core/querying/

https://docs.microsoft.com/en-us/ef/core/querying/tracking