In WinForms met een Database Verbinden

Met behulp van Entity Framework Core (EFCore)

# Introductie

We maken in deze les een applicatie om producten te bekijken. Deze producten vallen onder bepaalde categorieën. Deze oefening lijkt een beetje op de opdracht bij het project ‘Barroc Intens’, toch staat deze oefening daar los van. Laat je dus qua opdracht niet verwarren. *Je kunt delen van de code hier uiteraard wel vrij gebruiken in je project.*

Afbeelding met tafel

Automatisch gegenereerde beschrijving

We gaan Windows Forms in combinatie met MySQL gebruiken. Dit doen we omdat we al kennis hebben van deze beide technieken.

*Er zijn modernere manieren om met databases te werken, maar om die nu in een project in te zetten is niet slim. Dan zouden we zowel veel nieuws moeten leren als een applicatie bouwen voor een klant. Die combinatie brengt veel extra druk met zich mee. Voor nu is MySQL en WinForms een prima combinatie voor het bouwen van een applicatie die intern bij een organisatie gebruikt gaat worden.*

**Globaal zijn dit de stappen die we in deze les gaan ondernemen:**

* Een project opstarten:
  + Benodigde NuGet paketten installeren
  + Database bouwen
  + Vanuit code verbinden met de database
* De database vanuit de code beheren:
  + Create
  + Read
  + Update
  + Delete
  + Werken met relaties tussen tabellen

*Dit document is losjes gebaseerd op deze handleiding van Microsoft:* [*https://learn.microsoft.com/en-us/ef/core/get-started/winforms*](https://learn.microsoft.com/en-us/ef/core/get-started/winforms)*.*

Inhoudsopgave

[Introductie 1](#_Toc118375693)

[Opbouwen 3](#_Toc118375694)

[Solution opstarten en instellen 3](#_Toc118375695)

[Modellen aanmaken 6](#_Toc118375696)

[Database connectie met DbContext 9](#_Toc118375697)

[Connectie testen 13](#_Toc118375698)

[Werken met de Models 15](#_Toc118375699)

[Data beheren met een DataGridView 15](#_Toc118375700)

[**Oefening 1: Voeg nog een Model toe** 19](#_Toc118375701)

[Data tonen in een ComboBox 19](#_Toc118375702)

[CRUD 21](#_Toc118375703)

[Create 21](#_Toc118375704)

[**Oefening 2: Een product toevoegen** 21](#_Toc118375705)

[Read 23](#_Toc118375706)

[Gerelateerde data lezen 23](#_Toc118375707)

[Update 25](#_Toc118375708)

[Delete 26](#_Toc118375709)

[Foreign Relationships 27](#_Toc118375710)

[Tip: Werk nooit handmatig met ID’s wanneer je Entity Framework gebruikt 27](#_Toc118375711)

[**Oefening 3: Relaties toevoegen, aanpassen en koppelen** 29](#_Toc118375712)

[Extra 31](#_Toc118375713)

[**Andere context: Game Rental Shop (extra oefening)** 31](#_Toc118375714)

[Andere context: What2Watch 32](#_Toc118375715)

# Opbouwen

De volgende stappen zullen de eerste zijn voor ieder project wat gebruik gaat maken van Windows Forms en MySQL:

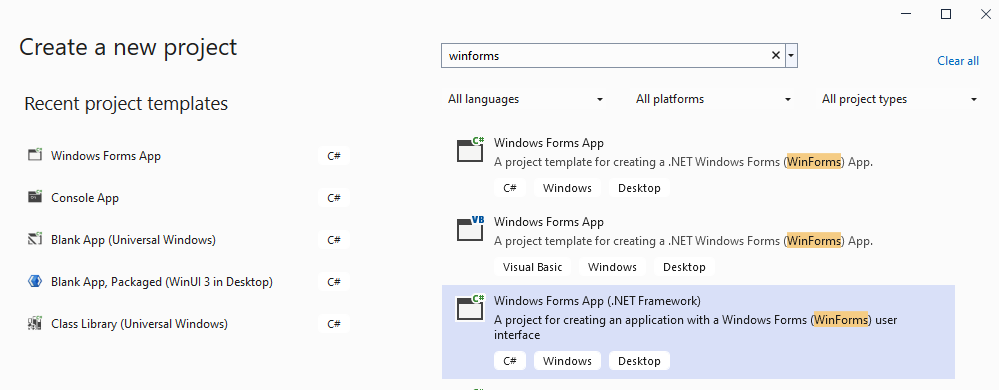
1. Een nieuwe Solution starten
2. De juiste NuGet package(s) binnenhalen
3. Model aanmaken
4. Met een MySQL Database verbinden
5. Met behulp van de modellen de database bouwen en vullen

In de komende paragrafen gaan we dat proces stap-voor-stap door.

## Solution opstarten en instellen

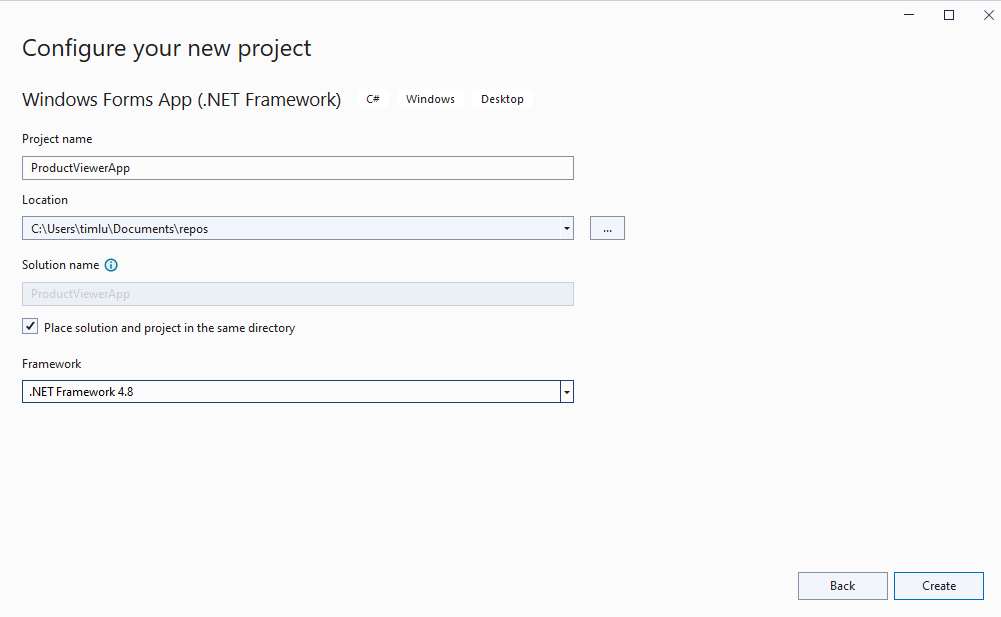
1. Start een WinForms (.NET Framework) project

*.NET Core ondersteund nog niet de DataGridView control die we gaan gebruiken.*



1. Selecteer een .NET Framework versie hoger dan 4.6.1+

*Wij gaan een NuGet Package gebruiken die enkel die versies van het .NET Framework ondersteunt.*

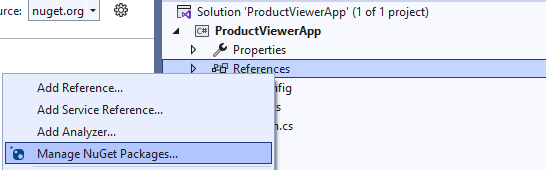


1. We gaan een NuGet package toevoegen aan dit project, die ons gaat helpen om met een MySQL database te verbinden: <https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql>
2. Neem de documentatie van die package door. Je ziet dat het .NET Framework ondersteund wordt enkel bij de Release 3.2.7:

*Afbeelding met tekst, schermafbeelding, monitor, zwart

Automatisch gegenereerde beschrijving*

1. Open de NuGet Package Manager voor het zojuist aangemaakte project:

**

1. Zoek de package in het Browse tabblad op en **selecteer de 3.2.7 versie** die werkt met onze versie van het .NET Framework:

*Afbeelding met tekst

Automatisch gegenereerde beschrijving*

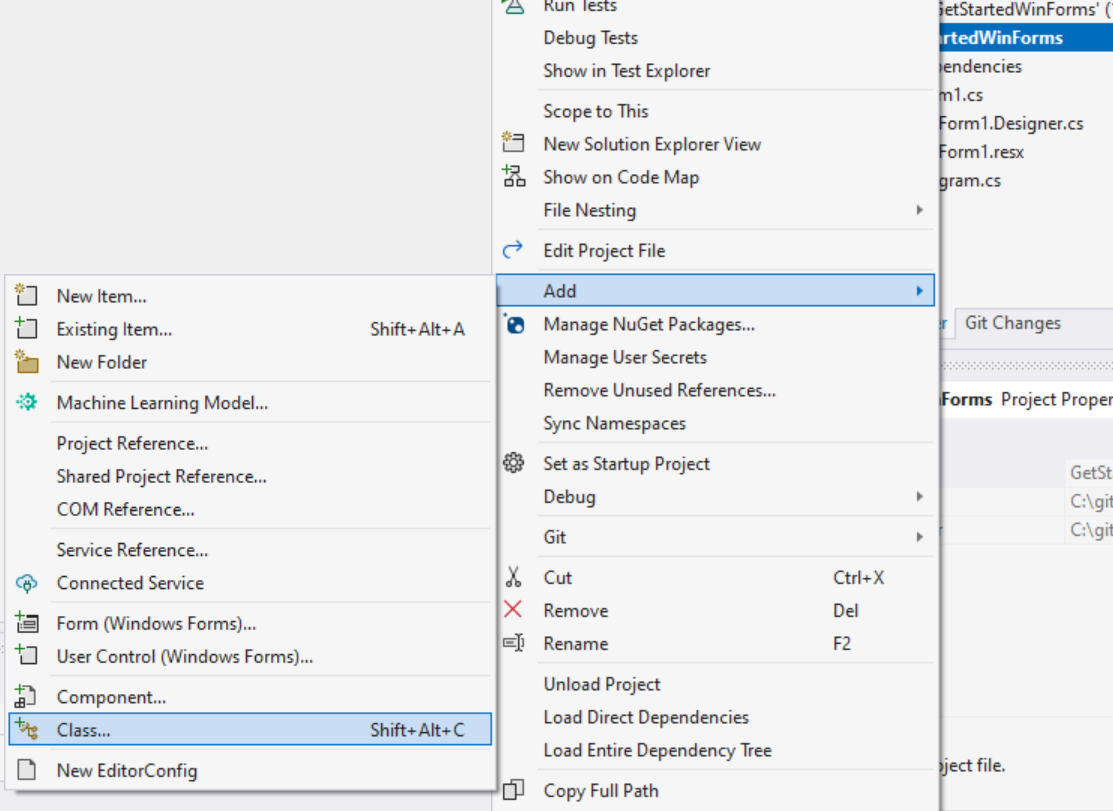
*Je ziet recht onderin de afbeelding hierboven dat deze versie “.NET Standard 2.0” ondersteund. Wanneer je dat ziet werkt de package ook vanaf het .NET Framework 4.6.1 of hoger. Bron:* [*https://learn.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-2-0#select-net-standard-version*](https://learn.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-2-0#select-net-standard-version)

1. Klik op Install/Installeer om dit NuGet package aan ons project toe te voegen.
2. Als laatste stap hernoemen we Form1.cs naar MainForm.cs

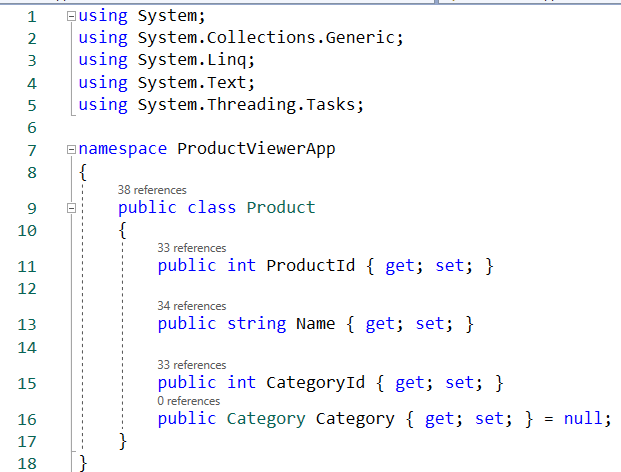
## Modellen aanmaken

We gaan onze database ‘Code First’ opbouwen. Dat wil zeggen dat we eerste classes gaan maken, die we vervolgens tot database tabellen laten omzetten door Entity Framework. *Het alternatief is om op basis van een database ontwerp, onze classes te genereren.*

1. Voeg een nieuwe class toe aan ons project voor onze ‘producten’:



1. Noem de klasse “Product” en plaats het in de file “Product.cs”
2. Pas de klasse aan zodat deze er zo uitziet:



*(Let op dat er een rood error-lijntje staat onder Category, totdat je de volgende stappen doorlopen hebt)*

1. Maak nog een klasse aan voor onze categorieën:
2. Noem de klasse ‘Category’ in de file ‘Category.cs’ en vul die met deze code:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*(*ObservableCollectionListSource komt uit de namespace ‘Microsoft.EntityFrameworkCore.ChangeTracking’, zorg dat die using dus bovenaan staat.)

De Product klasse is redelijk duidelijk. Een aantal eigenschappen van het product worden opgeslagen én via de ‘Category’ eigenschap is er een verwijzing naar een Category-object.

De Category klasse heeft een bijzondere eigenschap ‘Products’. Een categorie kan meerdere producten bevatten. Daarom willen we daar een lijst, maar we gebruiken hier een krachtigere lijst. Een ‘ObservableCollectionListSource’ is een lijst die in de gaten kan worden gehouden door andere code in onze applicatie. Een ObservableCollectionListSource’ lijst roept events aan wanneer er iets verandert. Dat gaat goed uitkomen om wijzigingen direct in onze schermen te laten zien. De DataGridView die we straks gaan gebruiken luistert ook naar die events om zo de laatste data te tonen.

De basis van onze applicatie is nu gebouwd: we hebben dankzij deze klassen duidelijk welke objecten er kunnen zijn en welke eigenschappen ze allemaal hebben.

## Database connectie met DbContext

We gaan werken met een MySQL database. Vervolgens verbinden we vanuit onze C# code met die database. Onze C# code gaat de database structuur ook opbouwen en vullen met test data.

1. Open XAMPP (of Laragon o.i.d.)
2. Start MySQL
3. Maak met PHPMyAdmin (of HeidiSQL o.i.d.) een database aan voor onze applicatie. Noem de database “barroc\_intens\_app”.

Nu onze database klaar staat gaan we vanuit onze applicatie met C# verbinden. Dat doen we doormiddel van een ‘Database Context’. Dat is een structuur bedacht door Microsoft en het gebruikt functionaliteiten uit de Entity Framework Core NuGet package.

1. Open ons project in Visual Studio en maak een nieuwe klasse ‘AppDbContext’
2. Pas de klasse aan zodat deze er zo uit ziet:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Laten we rustig naar deze code kijken, er zijn een aantal bijzonderheden:

* De klasse AppDbContext is erft eigenschappen en methodes over van ‘DbContext’ (die klasse komt uit: ‘Microsoft.EntityFrameworkCore’):



* + In OOP-talen kunnen klassen van elkaar ‘overerven’. Zo kunnen we functionaliteiten van andere ontwikkelaars in onze eigen klassen opnemen.
  + We gebruiken de dubbele punt om overerving aan te geven.
  + Microsoft weet dat applicaties met een database willen kunnen verbinden, daarom hebben ze een DbContext gemaakt die door andere applicaties gebruikt kan worden.
  + Onze AppDbContext verbind met een MySQL database met bepaalde gegevens. Een andere applicatie noemt het misschien OtherDbContext en gebruikt een ander soort database. Beide benutten ze de eigenschappen en methodes die DbContext laat overerven.
* Er zijn twee eigenschappen van het type ‘DbSet’: Dat zijn weer een bijzonder soort lijsten, dit keer zijn het lijsten gekoppeld aan database tabellen. Tussen de < > geven we het type van ieder item in de lijst aan:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

* + De eigenschap Products gaat gekoppeld worden aan een ‘products’ tabel
  + De eigenschap Categories aan een ‘categories’ tabel
* De eerste methode die we zien ziet er zo uit:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

* + Met ‘override’ overschrijven we de ‘OnConfiguring’ methode in DbContext. Standaard doet DbContext niet zo veel. Wij overschrijven de methode en stellen via de opties in (optionsBuilder) dat er met een MySQL database verbonden moet worden.
  + We geven aan de optionsBuilder.UseMySQL(x) een connection string mee. Daarin zit alle informatie die nodig is om te weten:
    - server = waar is de database? Vaak een IP-adres óf localhost als we lokaal werken.
    - database = wat is de naam van de database? Deze databasenaam hebben we aan het begin van dit hoofdstuk gekozen
    - user = wat is de gebruikersnaam van een gebruiker die toegang heeft tot de database? We gebruiken root omdat dat in XAMPP een gebruiker is die overal toegang toe heeft. (In productieomgevingen zouden we een gebruiker met beperkte rechten aanmaken)
    - password = wat is het wachtwoord voor die gebruiker? We laten dit leeg omdat in XAMPP er geen wachtwoord is voor de root gebruiker.
    - Al deze connectiegegevens worden gescheiden door puntkomma tekens.
* In de OnModelCreating methode gaan we onze database ‘seeden’. *Ofwel: vullen met (test-)data*. Kopieer de code op de volgende pagina in de OnModelCreating methode:

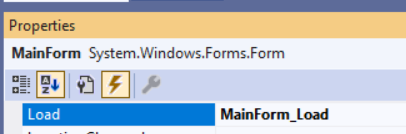
|  |
| --- |
| modelBuilder.Entity<Category>().HasData(  new Category { CategoryId = 1, Name = "Cheese" },  new Category { CategoryId = 2, Name = "Meat" },  new Category { CategoryId = 3, Name = "Fish" },  new Category { CategoryId = 4, Name = "Bread" });  modelBuilder.Entity<Product>().HasData(  new Product { ProductId = 1, CategoryId = 1, Name = "Cheddar" },  new Product { ProductId = 2, CategoryId = 1, Name = "Brie" },  new Product { ProductId = 3, CategoryId = 1, Name = "Stilton" },  new Product { ProductId = 4, CategoryId = 1, Name = "Cheshire" },  new Product { ProductId = 5, CategoryId = 1, Name = "Swiss" },  new Product { ProductId = 6, CategoryId = 1, Name = "Gruyere" },  new Product { ProductId = 7, CategoryId = 1, Name = "Colby" },  new Product { ProductId = 8, CategoryId = 1, Name = "Mozzela" },  new Product { ProductId = 9, CategoryId = 1, Name = "Ricotta" },  new Product { ProductId = 10, CategoryId = 1, Name = "Parmesan" },  new Product { ProductId = 11, CategoryId = 2, Name = "Ham" },  new Product { ProductId = 12, CategoryId = 2, Name = "Beef" },  new Product { ProductId = 13, CategoryId = 2, Name = "Chicken" },  new Product { ProductId = 14, CategoryId = 2, Name = "Turkey" },  new Product { ProductId = 15, CategoryId = 2, Name = "Prosciutto" },  new Product { ProductId = 16, CategoryId = 2, Name = "Bacon" },  new Product { ProductId = 17, CategoryId = 2, Name = "Mutton" },  new Product { ProductId = 18, CategoryId = 2, Name = "Pastrami" },  new Product { ProductId = 19, CategoryId = 2, Name = "Hazlet" },  new Product { ProductId = 20, CategoryId = 2, Name = "Salami" },  new Product { ProductId = 21, CategoryId = 3, Name = "Salmon" },  new Product { ProductId = 22, CategoryId = 3, Name = "Tuna" },  new Product { ProductId = 23, CategoryId = 3, Name = "Mackerel" },  new Product { ProductId = 24, CategoryId = 4, Name = "Rye" },  new Product { ProductId = 25, CategoryId = 4, Name = "Wheat" },  new Product { ProductId = 26, CategoryId = 4, Name = "Brioche" },  new Product { ProductId = 27, CategoryId = 4, Name = "Naan" },  new Product { ProductId = 28, CategoryId = 4, Name = "Focaccia" },  new Product { ProductId = 29, CategoryId = 4, Name = "Malted" },  new Product { ProductId = 30, CategoryId = 4, Name = "Sourdough" },  new Product { ProductId = 31, CategoryId = 4, Name = "Corn" },  new Product { ProductId = 32, CategoryId = 4, Name = "White" },  new Product { ProductId = 33, CategoryId = 4, Name = "Soda" }); |

Nu onze Database Context hebben moeten we die bij het opstarten van onze applicatie enkel nog laten uitvoeren:

1. Voeg boven in onze MainForm klasse een veld toe waar we onze Database Context gaan opslaan:



1. Maak een event handler die luistert naar het opstarten van onze eerste form:



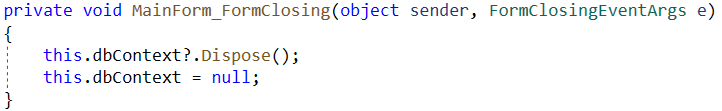
1. Vul de event handler met deze code:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

* + Bij het laden maken we een nieuwe AppDbContext aan en slaan die op in ons veld in de Form. Vanaf dit punt kunnen we hier in de MainForm bij de database context.
  + Met ‘this.dbContext.Database.EnsureCreated()’ zorgen we dat de database structuur gebouwd wordt. Als je met PHPMyAdmin (of HeidiSQL) in de database kijkt, dan zijn (nadat deze regel is uitgevoerd) de tabellen en kolommen aangemaakt.
  + Entity Framework gebruikt de classes en eigenschappen om te bepalen welke tabel- en kolomnamen er moeten komen.
  + EnsureCreated maakt de database als die er niet is én voert de OnModelCreating methode uit in onze database context. Zo kan na het opbouwen van de database ook (test-)data in de tabellen gezet worden.

1. We moeten nog één ding doen: wanneer de Form sluit, netjes de verbinding met de database sluiten. Maak daarvoor een event handler aan voor het FormClosing event en voer daar deze code in:



*Dispose zorgt ervoor dat de context zichzelf opruimt. We zetten hier dbContext op null zodat de opgeruimde verbinding niet zomaar weer gebruikt kan worden.*

## Connectie testen

Om te testen of we toegang hebben tot de database gegevens kunnen we een stukje test code schrijven in de MainForm\_Load. Neem de hieronder gemarkeerd code over, zet een breakpoint onderaan de methode, start de applicatie en inspecteer de variabelen:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Als het goed is zie je dat er 5 categories zijn én dat de eerste category 10 producten bevat.*

**Hier wat uitleg over de code hierboven:**

* 
  + Haalt alle categorieën op en zet ze in een List (waar je doorheen kunt ‘loopen’)
* Afbeelding met tekst

  Automatisch gegenereerde beschrijving
  + Dit zegt eigenlijk:
    - Haal van de eerste (0) categorie regel (Entry) in de database
    - de Products eigenschap op (dat is een verzameling/Collection)
    - Laad al die gerelateerde regels (Load)
  + We halen hier alle producten op die horen bij deze category.
  + Bij een een-op-meer relatie gebruiken we Collection. Bij een een-op-een relatie: Reference (zie hoofdstuk Extra onderaan dit document)
* 
  + Nu de products in de category geladen zijn kunnen we die naar een lijst omzetten.

Als je gegevens uit andere tabellen niet expliciet laad op deze manier, dan lijkt de lijst leeg.

Entity Framework voert op de achtergrond database query’s uit om de gegevens op te halen.

# Fist run of the applicationWerken met de Models

## Data beheren met een DataGridView

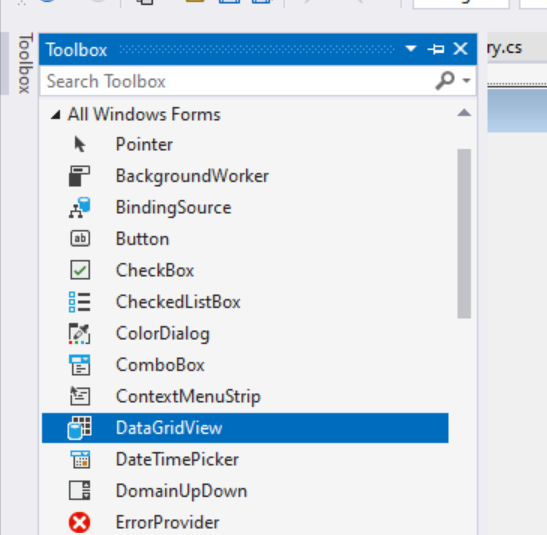
**De snelste manier om met een database te werken is via een DataGridView. Je kunt ervoor zorgen dat de gebruiker makkelijk data kan zien, bewerken en verwijderen in een tabelvorm.**

Een DataGridView (zie screenshot rechts) is een element waarin gegevens in tabelvorm gepresenteerd worden. Daarnaast kunnen gegevens direct in de tabel aangepast worden.

Voor Project Barroc Intens zijn er een paar schermen waar dit mogelijk interessant is. Bij de andere schermen kun je een combinatie van Label, TextBox, CheckBox, etc. elementen gebruiken.

We gaan nu twee DataGridView Controls toevoegen aan de Form en die koppelen met onze database (via de AppDbContext):

1. **Haal de test code van het vorige hoofdstuk weg, of zet het in het commentaar. We hebben dat niet meer nodig.**
2. Open de MainForm en voeg vanuit de Toolbox twee DataGridView Controls toe:



*Plaats de controls naast elkaar, links en rechts.*

1. Verander de (name) eigenschap van de linker DataGridView naar ‘dataGridViewCategories’
2. Verander de (name) eigenschap van de rechter DataGridView naar ‘dataGridViewProducts’
3. Voeg ook een Button Control toe met als tekst ‘Save’, geef deze de name ‘buttonSave’
4. Selecteer nu de linker DataGridView en ga via het kleine pijltje naar ‘Add Project Data Source’:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Kies vervolgens ‘Object’:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Klap in het overzicht ons project en namespace open, zoek de ‘Category’ klasse en vink deze aan:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*(Zie je jouw project of klasse niet in deze lijst? Klik dan op Cancel en bouw je project eerst ‘Build > ‘Rebuild Solution’. Volg daarna bovenstaande stappen opnieuw.)*

1. Klik op ‘Finish’, je DataGridView weet nu hoe het gegevens moet laten zien:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Door de DataGridView de klasse te geven, weet deze welke kolommen er komen voor de data die we straks in gaan laden.*

1. Bij de rechter DataGridView gaan we net anders te werk: We hebben namelijk al een Data Source van Category en kunnen via een Category alle producten ophalen (via de Products eigenschap). **Kies bij de Data Source daar dus categoryBindingSource > Products:**

****

We zijn bijna klaar: onze DataGridViews weten wat ze moeten tonen, maar hebben nog geen data. Die gaan we nu uit onze AppDbContext laden:

1. Ga naar de MainForm Load event handler en voeg deze twee regels toe:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Let op dat de naam op de tweede regel overeenkomt met de geselecteerde Data Source in de DataGridView:*

*Afbeelding met tekst

Automatisch gegenereerde beschrijving*

1. Test de applicatie, als het goed is zie je in de linker DataGridView nu alle categorieën.

Voor de rest van de stappen kun je de handleiding van Microsoft volgen vanaf het hoofdstuk ‘Populating the Products view’. Daar ga je zorgen dat de producten van de links geselecteerde categorie geladen worden: <https://learn.microsoft.com/en-us/ef/core/get-started/winforms#populating-the-products-view>.

Voor het opslaan van wijzigingen in de DataGridView kun je verder vanaf het hoofdstuk ‘Saving Changes’ in de Microsoft handleiding: <https://learn.microsoft.com/en-us/ef/core/get-started/winforms#saving-changes>

## **Oefening 1: Voeg nog een Model toe**

Voeg nu ook een ‘Flavour’ (smaak) model toe. Hierin komen smaken als ‘Sweet’, ‘Bitter’ en ‘Salty’. Een product heeft 1 smaak. Gebruik deze checklist om te controleren of je klaar bent:

* Er is een klasse voor deze Model aangemaakt
* Andere Models hebben de juiste relatie en kolom om met deze Model te verbinden
* Jouw AppDbContext bevat een DbSet voor deze nieuwe tabel
* In de OnModelCreating heb je:
  + een seeder toegevoegd om ten minste 3 flavours in de database te zetten
  + de andere seeders aangepast om ieder product een smaak te geven

## Data tonen in een ComboBox

Naast de DataGridView is er nog een Control waarmee je (dankzij EF) makkelijk data kunt tonen: de ComboBox. Een ComboBox ken je misschien ook wel als een ‘Dropdown’ of een soort ‘<select>’. Het toont een lijstje van opties.

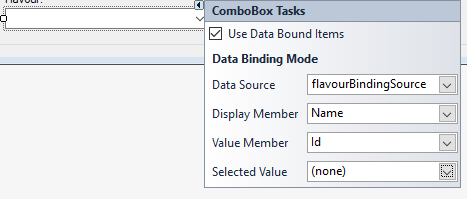
Om er data in te verbinden doorloop je dezelfde stappen als bij de DataGridView:

1. Sleep een ComboBox op je Form
2. Klik op het witte pijltje bij de ComboBox
3. Vink nu ‘Use Data Bound Items’ aan
4. Kies de data source op dezelfde manier als bij de DataGridView:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Vervolgens kies je welke eigenschap van het Model de ‘Display Member’ is. Van ieder Model in de data source (ofwel database) wordt deze eigenschap aan de gebruiker getoond.

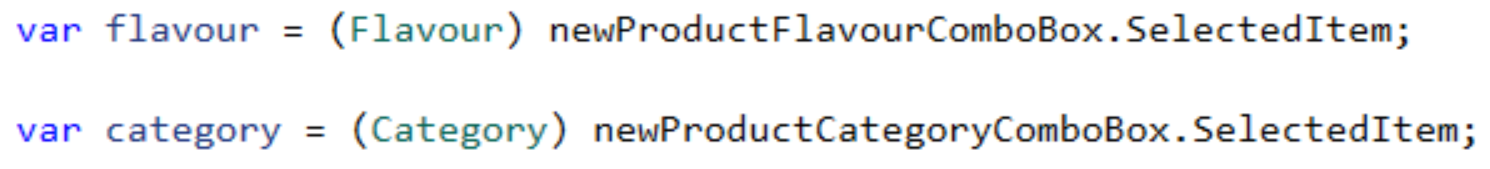


1. De ‘Value Member’ bepaalt welke waarde je krijgt als je de ComboBox vraagt om de geselecteerde waarde:



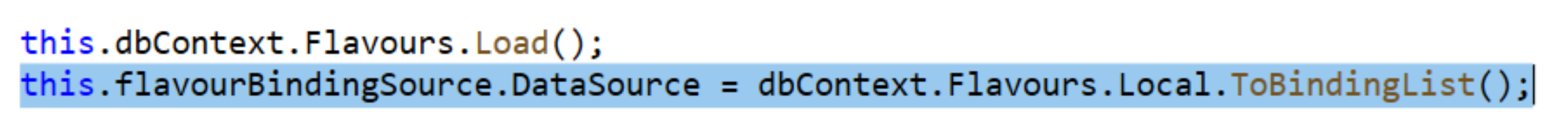
*We casten dit naar een ‘int’ omdat de ‘Id’ eigenschap een int is. Als je je muis zweeft over de ‘SelectedValue’ van een ComboBox zie je dat het datatype eigenlijk een ‘object’ is. Object kan van alles zijn, dit doet Microsoft omdat jij voor Id kiest (int), maar iemand anders misschien ‘Name’ kiest (string) of in een andere applicatie zelfs een heel ander datatype.* ***Je moet hier dus altijd casten naar het juiste type.***

1. In plaats van de ‘Value Member’ is het logischer om het gehele object op te halen, dat geselecteerd is in de ComboBox via SelectedItem:



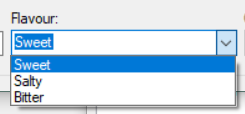
*Wat het type is van een SelectedItem ligt aan de DataSource die je instelt op de ComboBox. Je moet het resultaat altijd ‘casten’, want SelectedItem heeft als type ‘object’.*

1. Tot slot moeten we bij het opstarten van de applicatie zorgen dat de ComboBox de juiste DataSource ingesteld krijgt op de zojuist ingestelde BindingSource:



*(We laden hier ook alvast alle Flavours uit de database, zodat de ComboBox sowieso alle flavours toont)*

De ComboBox ziet er nu zo uit:



1. Om te zorgen dat de gebruiker niet in de DropDown kan typen stellen we van deze Control de DropDownStyle eigenschap in op DropDownList i.p.v. DropDown:

Afbeelding met tafel

Automatisch gegenereerde beschrijving

## CRUD

### Create

Wanneer je geen gebruik maakt van de DataGridView én je wilt handmatig gegevens toevoegen. Dan kun je:

1. *Of,* het object direct toevoegen aan de database verzameling:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. *Of,* het object toevoegen aan een ‘parent’ die al in de database staat:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Hoe dan ook moeten we, na een of meerdere toevoegingen of wijzigingen, altijd opslaan:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

We maken dus eerst een instantie van onze Model ‘Product’. Daarna vullen we de eigenschappen en vervolgens zetten we het in de Products DbSet van onze Database Context.

Om te zorgen dat deze wijziging wordt opgeslagen hoef je dat enkel aan de Database Context te vragen:

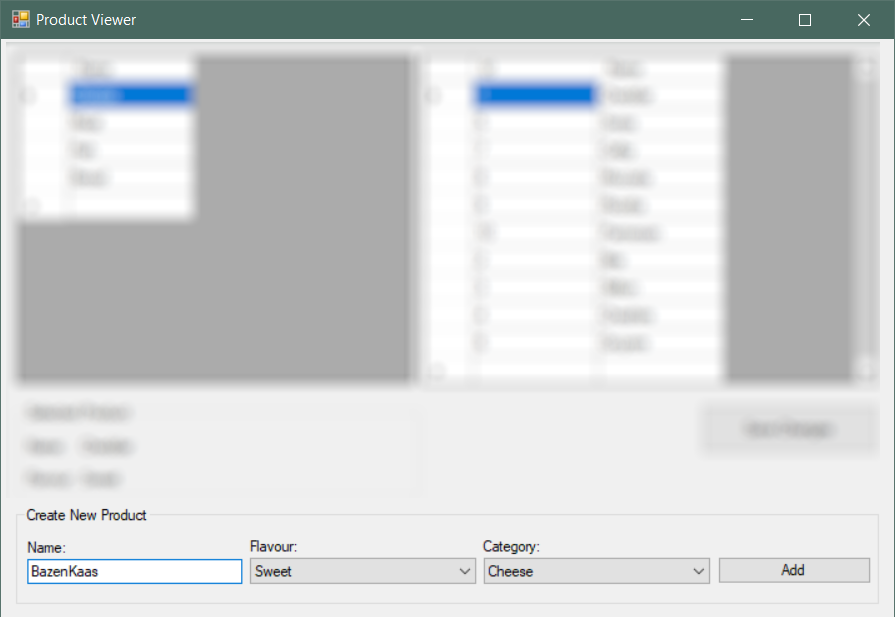


#### **Oefening 2: Een product toevoegen**

In de bovenstaande uitleg las je over de volgende puzzelstukjes:

* Hoe kunnen we gegevens (zoals Flavour) in een ComboBox laten zien
* Ook zien we hoe we de ‘Id’ krijgen van de in de ComboBox geselecteerde Flavour
* Hoe we een product kunnen toevoegen

Pas die kennis nu toe door de volgende functionaliteit in te bouwen: Zorg dat de gebruiker een Product kan toevoegen met een formulier dat er zo uitziet:



### Read

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Afbeelding met tekst

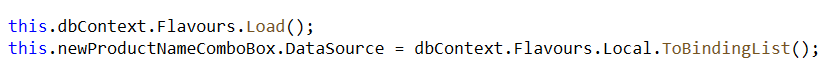
Automatisch gegenereerde beschrijving

#### Gerelateerde data lezen

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Omdat de Flavour gegevens van een Product in een andere tabel staan moeten die gegevens ‘Expliciet’ geladen worden. Dat hebben we eerder gedaan door alle Flavour gegevens te laden:



Een andere manier is om alleen de Flavour gegevens te laden van de benodigde producten:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Hier zeggen we eigenlijk: “Hey database, deze category heeft een regel (Entry) in onze database. Die heeft een relatie tot meerdere producten via de Products eigenschap (het zijn er meerdere, dus Collection). Maak daar eens een Query voor én zorg dat je daarbij voor ieder product de smaak ophaalt via de Flavour eigenschap van een product. Laad al die gegevens.

### Update

Zoals we tot nu toe hebben gezien is het werken met de database vrijwel “onzichtbaar” dankzij EFCore. In plaats daarvan werken we met objecten. EFCore regelt op de achtergrond het uitvoeren van queries.

**Om gegevens aan te passen wijzigen we simpelweg de eigenschappen van een object. Vervolgens vragen we EFCore met DbContext.SaveChanges() om de wijzigingen in de database te verwerken.**

Bijvoorbeeld:

**Afbeelding met tekst

Automatisch gegenereerde beschrijving**

Maak een aanpassing aan het object.

Verwerk wijzigingen in database

In DataGridView geselecteerde categorie *(komt al uit database via Data Binding)*

De laatste regel forceert de DataGridView om de data opnieuw op te halen. Pas dan ziet deze dat de categorie is aangepast.

### Delete

Een bepaald product verwijderen gaat als volgt:

1. Zorg dat je een referentie hebt naar het Product (bijvoorbeeld vanuit de geselecteerde regel in een DataGridView):

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Vraag de Database Context om die regel te verwijderen:



1. Net als bij de andere wijzigingen moeten we expliciet aangeven dat de wijzigingen doorgevoerd moeten worden:



1. In plaats van het meteen aanroepen van de ‘SaveChanges’, kun je de regel ook eerst rood maken. De gebruiker kan dan zelf kiezen om wijzigingen op te slaan, door op de ‘Save’ knop te drukken:

**

## Foreign Relationships

We zagen bij het hoofdstuk CRUD > Read al een uitleg over het lezen van gerelateerde data. Ook in de ProductViewer voorbeeld applicatie kun je terugzien hoe relaties precies werken. In dit hoofdstuk bekijken we wat tips, trucs en oefeningen die je kunt gebruiken bij het ontwikkelen van applicaties.

**Herhaling:** Een Foreign Key verwijst naar de Primary Key van een entiteit in een andere tabel.

|  |  |
| --- | --- |
| **Voorbeeld van Primary en Foreign Keys met twee tabellen**  De Primary Key’s zijn geel, de Foreign Key is groen.  De Foreign Key KlasId verwijst naar Id in de klassen tabel. | |
| **studenten**   |  |  |  | | --- | --- | --- | | Id | Naam | KlasId | | 12345 | Janiek | 2 | | 12346 | Noah | 1 | | **klassen**   |  |  | | --- | --- | | Id | Naam | | 1 | TTR4-SSD3A | | 2 | TTR4-SSD3B | |

### Tip: Werk nooit handmatig met ID’s wanneer je Entity Framework gebruikt

**Onthoudt als vuistregel** dat je dankzij Entity Framework/EFCore NOOIT\* handmatig met de ID’s en Foreign Key ID’s hoeft te werken.

*NOOIT\* = Zeg nooit nooit. Er zijn uitzonderingen waar je wel met ID’s moet werken. Een voorbeeld is in de Seeders. Daar moet je ID’s en Foreign Key ID’s aangeven. Maar in de rest van de code komt het uiterst zelden voor.*

Als je denkt dat het nodig is om handmatig een ID of Foreign Key te vullen dan mis je waarschijnlijk de makkelijkere manier. Die makkelijke manier is het beste te demonstreren met de ProductViewer als voorbeeld:

**We gaan een nieuw product opslaan bij een bestaande category.**

De situatie is als volgt:

1. De Category Model heeft een ObservableCollectionListSource<Product>:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Dankzij deze lijst kan EFCore zien dat er een relatie is tussen Category en Products*

1. **Kijk naar de Category model en vul voor jezelf de stippellijn in:** Een Category kan …… product/producten hebben.
2. In de Product Model zien we hoe die aan een Category gekoppeld is:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*De werkelijke koppeling in de database is met behulp van de Foreign Key ‘CategoryId’.*

1. In de Product Model zien we naast de Foreign Key (CategoryId), ook een extra eigenschap: Category. Deze heeft als type de klasse ‘Category’. EFCore gebruikt dit om, als we een product uit de database halen, hier de bijbehorende category in te koppelen.

Laten we nu eens kijken hoe we een nieuw product toevoegen aan een bestaande Category:

1. We halen een bestaande Category op, bijvoorbeeld uit een DataGridView. In dit voorbeeld pakken we simpelweg de eerste uit de tabel:

*Afbeelding met tekst

Automatisch gegenereerde beschrijving*

1. Nu voegen we een nieuw product toe aan die categorie, via die ObservableCollectionListSource<Product> eigenschap in Category:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Deze wijzigingen staan enkel in de DbContext en zijn nog niet verwerkt. Doe dat met:



Deze stappen kun je altijd volgen wanneer je een ‘parent’ hebt die al in de database staat (Category in dit geval) en daar een nieuwe ‘child’ aan toe wilt voegen (Product in dit geval).

**Entity Framework** zal nu op de achtergrond ongeveer dit doen:

* *🎵Beep-boop🎵 ik ben een computerscript*
* Ik bekijk voor iedere categorie, iedere eigenschap: is er iets verandert?
* *“Hey! Ik zie dat er een Product is bijgekomen in Products van de eerste categorie!”*
* Ik vul bij die Product nu de CategoryId voor je in en zal de Category ook instellen.
* Ik zal nu een INSERT query maken en die uitvoeren voor je.
* Ik doe dit bovenstaande opnieuw voor alle andere models.
* *🎵Beep-boop🎵 einde van het computerscript.*

Entity Framework zal queries efficienter uitvoeren waar mogelijk. Bijvoorbeeld door alle INSERTs tegelijk te versturen, of door bij SELECT queries voor verschillende tabellen JOINs te gebruiken.

### **Oefening 3: Relaties toevoegen, aanpassen en koppelen**

Je krijgt een startpunt van een variant van de ProductViewer. In deze variant bewerk je categorieën en producten via TextBoxes (in plaats van via de DataGridView). Je kunt via diezelfde tekstvelden ook nieuwe categorieën en/of producten toevoegen.

1. Download, unzip en open het startpunt *ProductViewerApp\_Updaten\_Foreign\_Relations.zip* van Itslearning
2. De code is niet af en tot je bij stap *h* bent zullen er foutmeldingen zijn.
3. Zoek in de code de AddCategory() methode. Je gaat hier zelf de code schrijven om een categorie aan te maken. Volg de instructies 1 t/m 3 in het commentaar.
4. Zoek nu de AddProduct() methode en doe hetzelfde opnieuw, maar dan voor Product.
5. Zoek nu de BtnCategoryAdd\_Click event handler. Deze wordt aangeroepen wanneer de gebruiker op de ‘Add Category’ knop klikt.
6. Je hebt al een category (dankzij de AddCategory methode0, maar die staat nog niet in de database. Volg instructies 5 en 6 in het commentaar en voeg de category toe aan de database (zoals je in het hoofdstuk CRUD > Create geleerd hebt).
7. Zoek nu de BtnProductAdd\_Click event handler. Volg de instructies 7 en 8 in het commentaar om het nieuw aangemaakt product nu aan de geselecteerde category (bestaat in database) toe te voegen. Gebruik wat je geleerd hebt in dit hoofdstuk.
8. Je hebt nu genoeg code geschreven om te gaan testen. Er is één fout die kan verschijnen:

Wanneer we een nieuwe categorie toevoegen, die selecteren en vervolgens een nieuw product toevoegen krijgen we deze foutmelding:

*Afbeelding met tekst

Automatisch gegenereerde beschrijving*

1. product.Category is null. Om deze fout te begrijpen moeten we een stap terug. Als we in de Call Stack rechtsonderin kijken zien we dat het misgaat in de DgvProducts\_SelectionChanged event handler. Maar daarvoor is in onze code iets gebeurt. Klik op de regel die begint met ‘BtnProductAdd\_Click’.
2. Visual Studio laat ons zien dat het vanaf deze regel misgaat:

Afbeelding met tekst

Automatisch gegenereerde beschrijving Afbeelding met tekst

Automatisch gegenereerde beschrijving

1. Neem de tijd om te ontleden wat er misgaat. Gebruik eventueel breakpoints om stap-voor-stap door de code te gaan. Als je de reden denkt te weten, of je komt er niet aan uit, kijk dan pas naar deze uitleg:

We kunnen nu een idee krijgen wat er misgaat:

1. We voegen een nieuw product toe aan de category via category.Products
2. Omdat category.Products een ObservableCollectionListSource is zal de DataGridView de wijziging meteen oppikken en verwerken.
3. Het nieuwe product wordt op de DataGridView getoond.
4. Omdat het in onze nieuwe categorie, het eerste product is, wordt deze automatisch geselecteerd.
5. Daardoor wordt de DgvProducts\_SelectionChanged event handler aangeroepen.
6. Maar de ‘SaveChanges’ is pas op de volgende regel. Dus EFCore heeft ons product nog niet aan de category kunnen koppelen.
7. We moeten er in de DgvProducts\_SelectionChanged event handler dus rekening mee houden, dat een product mogelijk nog (even) geen category gekoppeld heeft.
8. Haal de if-statement uit het commentaar bij instructie 9 in de DgvProducts\_SelectionChanged event handler.
9. Ga terug naar de BtnProductAdd\_Click event handler, volg instructie 10 en sla de wijzigingen op.

De applicatie werkt nu volledig. Test het geheel en doorloop alle code nog eens. Controleer voor jezelf dat je begrijpt wat de code doet. Stel vragen aan de docent als je ze hebt of.

# Extra

In de Microsoft handleiding kun je lezen hoe je de gebruiker slechts bepaalde kolommen kunt laten bewerken/zien: <https://learn.microsoft.com/en-us/ef/core/get-started/winforms#configuring-what-is-displayed>

## **Andere context: Game Rental Shop (extra oefening)**

1. Download, unzip en open het startpunt ‘’ van Itslearning
2. Zorg dat je een database genaamd ‘shopminimalexample’ hebt aangemaakt in jouw MySQL server (via HeidiSQL of PHPMyAdmin)
3. Deze applicatie lijkt goed te werken. Test maar eens door:
   1. Alle klanten te tonen
   2. Alle games te tonen
   3. Een game uit te lenen aan een klant
   4. Dan weer alle klanten te tonen
4. Je ziet dat de gegevens netjes in de DbContext komen te staan en in alle menu’s te zien zijn.
5. Maar wanneer we de applicatie herstarten gaat het mis. Voer dezelfde tests nu eens uit.
6. Je krijgt foutmeldingen omdat de AppDbContext niet de gerelateerde data ophaalt. Dit was bij de eerste tests geen probleem, omdat wij die data net hadden ingevoerd. Maar na opnieuw opstarten moet de gerelateerde data uit de database geladen worden.
7. **Neem even de tijd om de code door te nemen en te begrijpen. Gebruik daarbij eventueel breakpoints om stap-voor-stap door de code te lopen.**
8. Waar we eerder de Reference en Collection methodes gebruikte om data op te halen voor een dbContext.Entry, gaan we dat nu met ‘Include’ doen. Het is een andere manier, die hetzelfde bereikt.
9. We kunnen de ‘Include’ methode gebruiken wanneer we een query maken. Een voorbeeld van een Query is de ‘Where’-methode:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Je herkent een query aan het IQueryable type. Die methode voert nog niets uit op de database, maar pas wanneer we .ToList() of .First() aanroepen zal de query uitgevoerd worden.

Het voordeel hiervan is dat we één lange query maken die we naar de database sturen ipv veel losse queries. Dat is efficienter omdat databases “dichterbij” de data zitten, dan onze code.

1. Zoek de ReturnGame methode, lees instructie 1 in het commentaar, verwijder dat commentaar en voer het als volgt uit:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*(Op de achtergrond zal Entity Framework een JOIN query doen voor rentals met games. Wij hoeven hier niet verder over na te denken.)*

1. Zorg nu dat bij de rentals ook de customer wordt geladen. Voeg daarvoor *nog* een Include toe op dezelfde manier op de plek van //…
2. Volg op dezelfde manier instructie 2 in de ViewAllCustomers methode
3. Volg op dezelfde manier instructie 3 in de ViewAllGames methode
4. Test de applicatie weer. Nu zijn de errors verholpen omdat de gerelateerde/Foreign gegevens wel worden opgehaald.

## Andere context: What2Watch

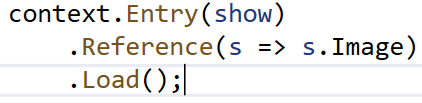
In deze voorbeelden is een Show een Netflix Show. Een Episode hoort bij één Show. Een Show heeft meerdere Episodes. Er is ook een Image, een Image hoort bij één Show of één Episode.

* Gerelateerde data (uit een andere tabel) moet geladen worden met

Explicit, Eager of Lazy loading:

<https://docs.microsoft.com/en-us/ef/core/querying/related-data/explicit>

Voorbeeld voor een-op-een relatie:



*Image is hier een relatie naar een ander model/tabel*

Of om meteen bij het ophalen van de Shows ook de gerelateerde data te joinen:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

*Hier laden we meteen bij het opvragen van alle shows óók de bijbehorende Image en Episodes informatie.*

**Veel voorkomende errors**

* InvalidOperationException bij loop door data:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

**Oorzaak:** context.Shows doet iedere foreach herhaling een nieuwe query en houdt binnen de loop de database verbinding bezet. Hierdoor kan de context niet opnieuw benaderd worden.

**Oplossing:** De hele query uitvoeren doormiddel van ToList, voordat de loop gestart wordt:

