HoGent

BEDRIJF
EN
ORGANISATIE

Hoofdstuk 7 : Entity Framework Core

https://github.com/WebIII/07thEntityFramework

Hoofdstuk 7 : Entity Framework Core

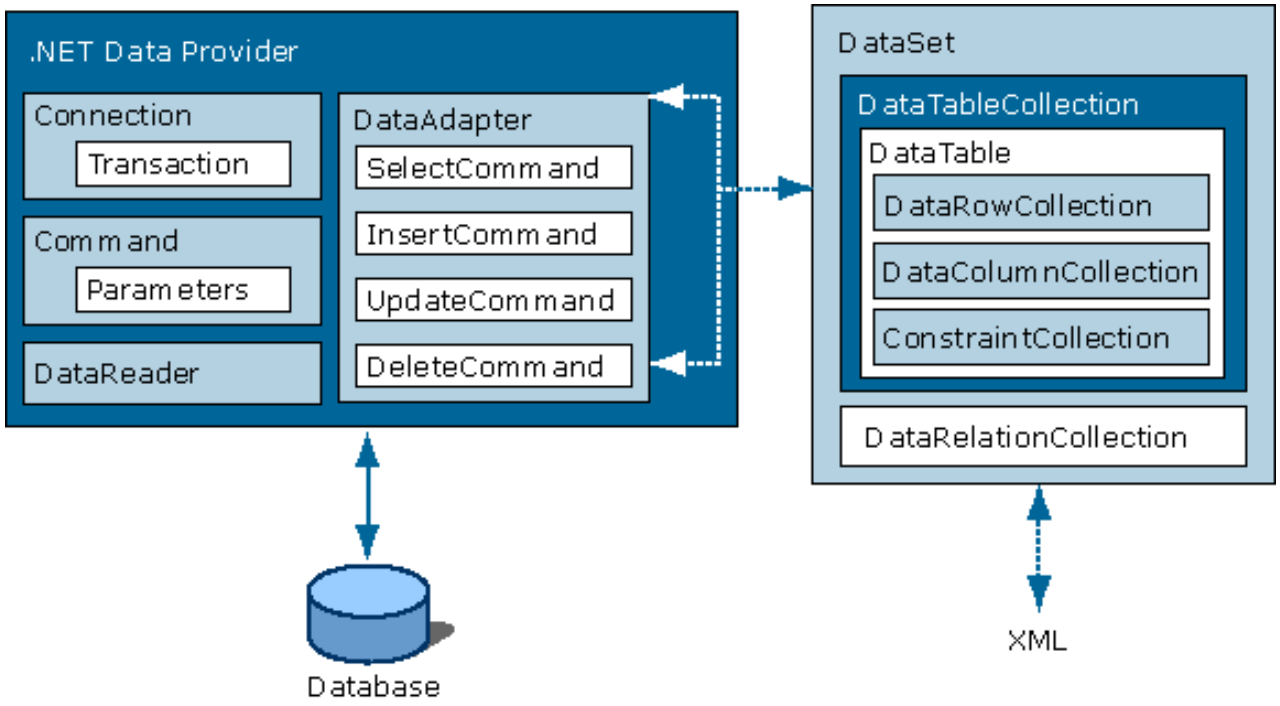
1. Inleiding
2. Entity Framework Core - Code First :
    1. Installatie EF Core
    2. De Persistentieklasse
    3. Aanmaken domeinmodel volgens Code First Workflow
    4. Fluent API
    5. Associaties
    6. Overerving
3. Seeding van de database
4. Querying en saving data

Inleiding

HoGent

1. Inleiding

- ADO.NET
  - Library om volledig zelf de persistentielaag te bouwen



1. Inleiding

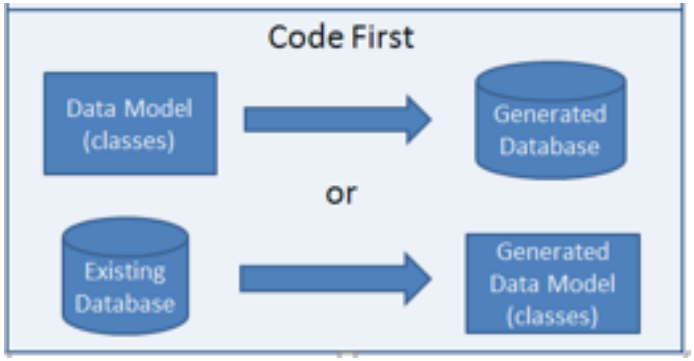
- Entity Framework Core
  - Is een open source **cross platform ORM** (Object Relational mapper) framework
  - Werkt met **relationele en niet relationele** datastores. Voor een overzicht van de providers zie https://docs.microsoft.com/en-us/ef/
  - Data access gebeurt o.b.v. een model
  - https://github.com/aspnet/EntityFramework/wiki/Roadmap, zie backlog features

1. Inleiding

- Entity Framework Core
  - Genereert de persistentielaag: infrastructuur om objecten te **mappen** naar database en viceversa
    - Mapt klassen naar tabellen, properties naar kolommen in tabel
    - Mapt objecten naar rijen
    - Mapt associaties naar FK relaties
    - Ondersteunt overerving
  - Een **API, Linq to Entities**, voor het opvragen en manipuleren van de objecten. De acties worden vertaald naar queries op de database
    - .Net taal syntax, gecompileerd!
    - Onafhankelijk van de backend SQL dialect, OO taal
  - Documentatie op https://docs.microsoft.com/en-us/ef/.

1. Inleiding

- Het opbouwen van een database/model: 2 manieren
  - Code-First : je bouwt het model en genereert de database
    - 2 mogelijke werkwijzen
      - Drop-Create database
      - Migrations : bestaande database verder aanpassen of nieuwe database stap per stap opbouwen (http://www.learnentityframeworkcore.com/migrations)
  - Database-First: vanuit een bestaande database genereer je het model
    - Commando scaffold-dbcontext. Meer op https://docs.microsoft.com/enus/ef/core/miscellaneous/cli/dotnet

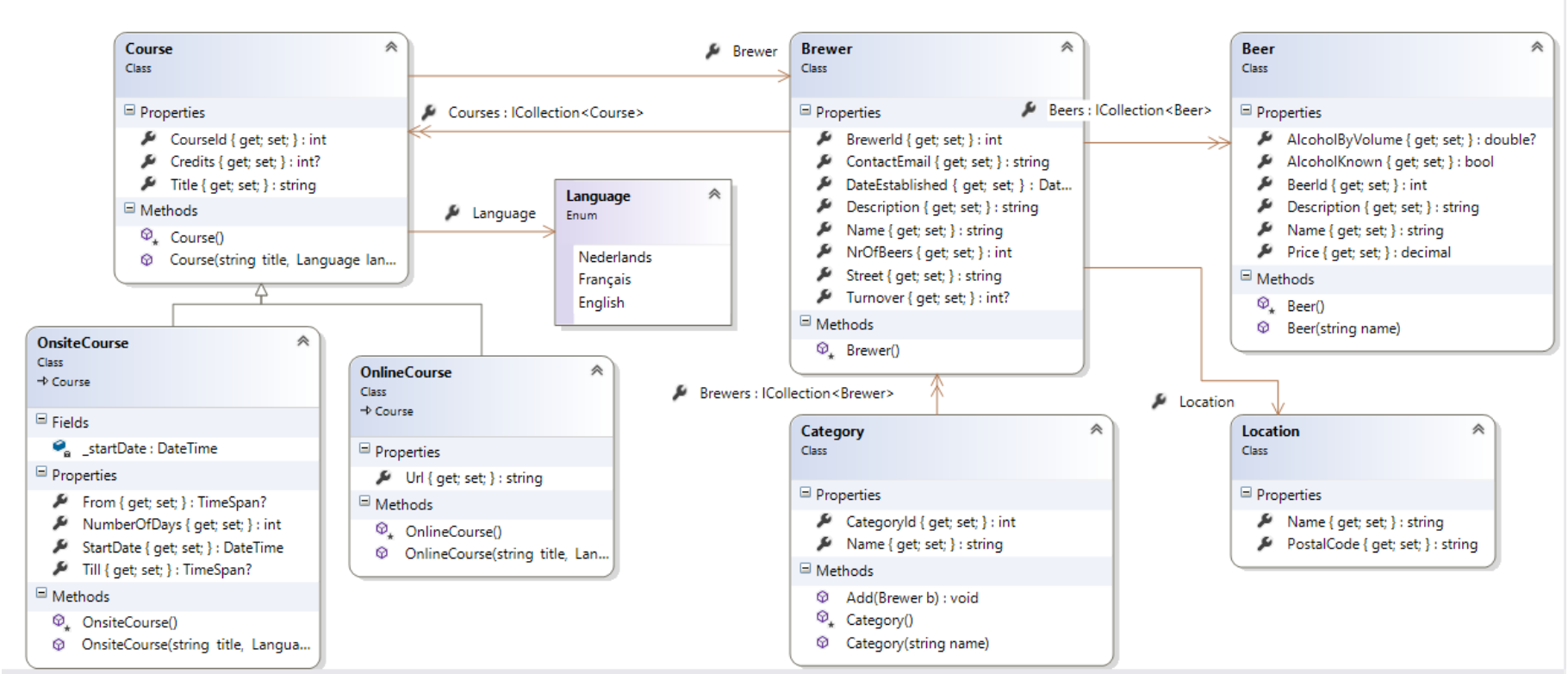


1. Inleiding

- **Code-First: model -> new database**
  1. Installeer Entity Framework Core
  2. Maak de persistentielaag aan en configureer de database provider
  3. Maak domein model aan (of voer aanpassingen door)
  4. Drop en creëer de database.
  5. Customiseer de mapping waar nodig
  6. Terug naar 3 totdat database correct is aangemaakt

Inleiding

- Het domein model



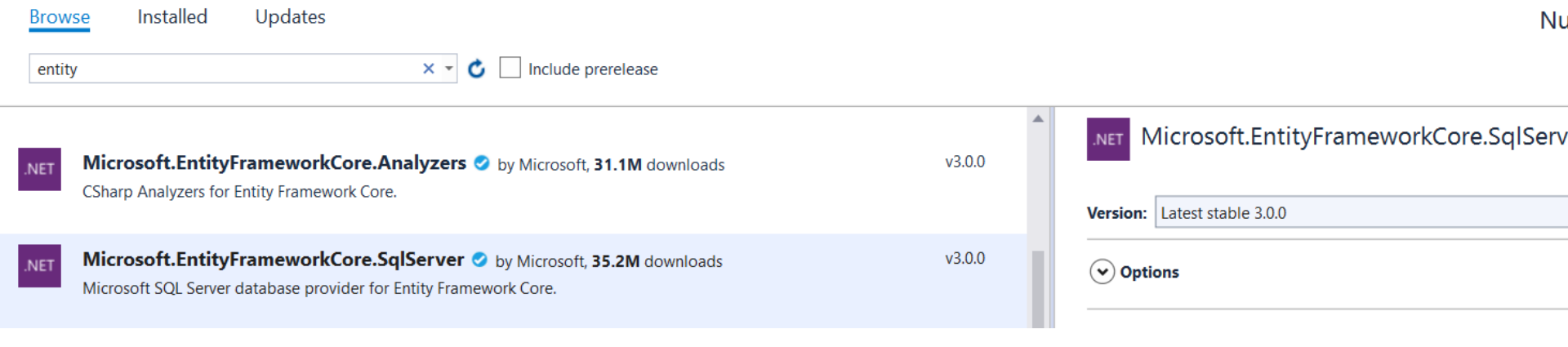
Entity Framework Core – To a new database (Code First workflow)

Installatie EF Core

HoGent

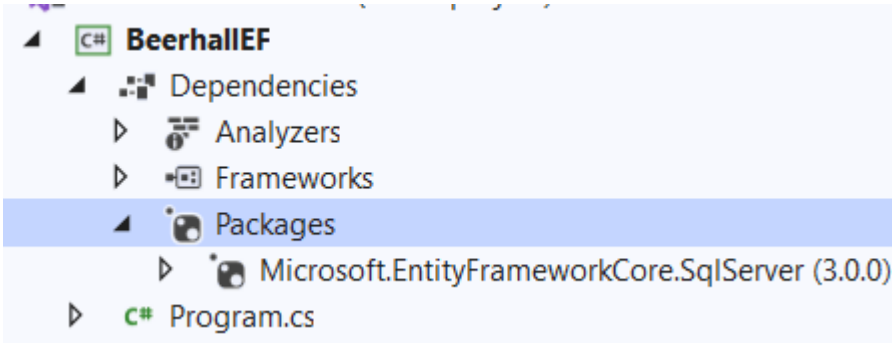
Installatie EF Core

- Maak een nieuw **.Net Core Console** applicatie aan, noem dit **BeerhallEF**
- Voeg **Entity Framework Core** toe
  - Rechtsklik source BeerhallEF in solution explorer > **Manage Nuget Packages**
  - Installeer de nuget package voor de gewenste database provider
    - Zoek naar **Microsoft.EntityFrameworkCore.SqlServer** en klik Install v3.0.0



Installatie EF Core

- Installatie van **Microsoft.EntityFrameworkCore.SqlServer**
  - Of via Tools > Nuget package Manager Console
    - Install-Package Microsoft.EntityFrameworkCore.SqlServer
  - voegt een Dependency toe (zie solution explorer)



  - Build je project, zodat de packages geinstalleerd worden en csproj wordt aangepast.(rechtsklik project > edit project file)

```
<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.0.0" />
</ItemGroup>
```

Installatie EF Core (Tools)

- Installatie van **EntityFrameworkCore.Tools**
  - Geeft de mogelijkheid om via een CLI commando's te runnen:

  - Build, en bekijk de .csproj file via Edit

```
<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.0.0" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="3.0.0">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
  </PackageReference>
</ItemGroup>
```

Entity Framework Core – To a new database (Code First workflow)

De persistentieklasse (DbContext)

HoGent

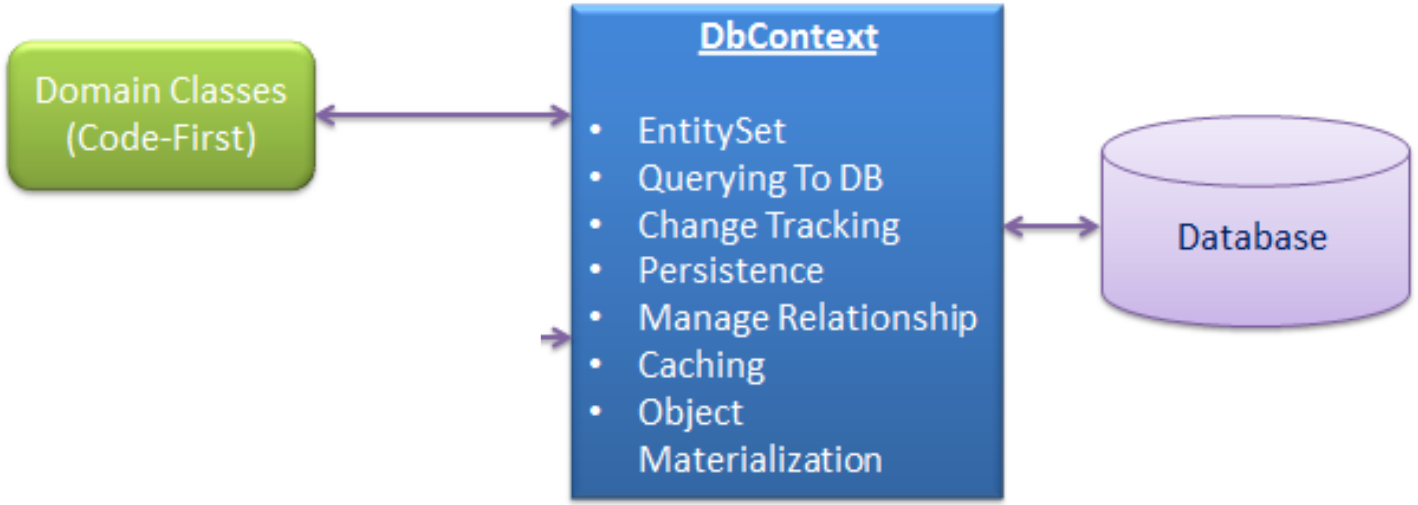
De persistentieklasse

- Maak een **klasse** aan die erft van **DbContext**
  - ”this derived context represents a session with the database, allowing you to query and save data”
  - Maak een **folder Data** binnen het project
  - Voeg een **klasse ApplicationDbContext** toe
    - erf van DbContext (namespace Microsoft.EntityFrameworkCore): voorziet in alle functionaliteiten van EF om met de database te communiceren

```
using Microsoft.EntityFrameworkCore;

namespace BeerhallEF.Data
{
    public class ApplicationDbContext:DbContext
    {
    }
}
```

De persistentieklasse

- DbContext(namespace Microsoft.EntityFrameworkCore)



De persistentieklasse

- Configureer de **database provider**
  - Meer op https://docs.microsoft.com/en-us/ef/core/miscellaneous/configuring-dbcontext
  - DbContext vereist een instantie van DbContextOptions(Builder).
    - In console app : override OnConfiguring
    - In web app: dependency injection

```
public class ApplicationDbContext:DbContext
{
  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
  {
    var connectionstring =
            @"Server=.;Database=Beerhall;Integrated Security=True;";
    optionsBuilder.UseSqlServer(connectionstring);
  }
```

Deze code wordt uitgevoerd bij het aanmaken van een nieuwe instantie van ApplicationDbContext

Tip : typ in **override** en dan krijg je een lijst van de methodes

De persistentieklasse

commit “Add ApplicationDbContext”

- Pas program.cs aan

```
class Program
{
    static void Main(string[] args)
    {
        using (ApplicationDbContext context = new ApplicationDbContext())
        {
            context.Database.EnsureDeleted();
            context.Database.EnsureCreated();
            Console.WriteLine("Database created");
        }
    }
}
```

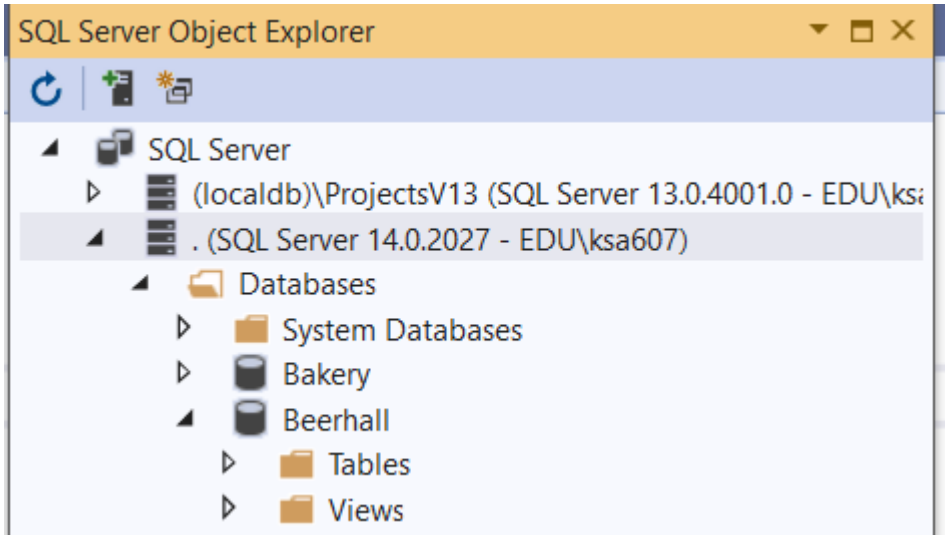
Indien de database bestaat dan wordt deze eerst verwijderd.

De database wordt gecreëerd indien deze nog niet bestaat

- Run. De database wordt gecreëerd.
- using: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-statement

De persistentieklasse

- Bekijk het resultaat
  - View > SQL Server Object Explorer
    - Klik op knop “Add SQL server”, server name = . (of localhost)



  - Of bekijk de database in SQL Server Management Explorer (zie verder)

Entity Framework Core – To a new database (Code First workflow)

Aanmaken Domain model volgens code first workflow

HoGent

EF Code First workflow

- Bouwen van domein model adhv de **code first workflow**
  1. Pas het domein model aan: creëer een nieuwe domeinklasse of wijzig een bestaande klasse, voeg associaties toe, …
  2. Drop en creëer de database
  3. Bekijk de gegenereerde tabellen
  4. Pas, indien nodig, de mapping aan
  5. Herhaal vorige stappen tot de database correct is aangemaakt
  6. Vul de database met sample data
  7. Commit

EF Code First workflow

STAP 1

Maak domein model aan

```
namespace BeerhallEF.Models
{
  public class Brewer
  {
    #region Properties

    public int BrewerId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string ContactEmail { get; set; }
    public DateTime? DateEstablished { get; set; }
    public string Street { get; set; }
    public int? Turnover { get; set; }

    #endregion
  }
}
```

```
namespace BeerhallEF.Data
{
  public class ApplicationDbContext : DbContext
  {
    public DbSet<Brewer> Brewers { get; set; }
```

STAP 2

Genereer de database

Brewers

Column Name	Data Type	Allow Nulls
BrewerId	int	☐
ContactEmail	nvarchar(MAX)	☑
DateEstablished	datetime2(7)	☑
Description	nvarchar(MAX)	☑
Name	nvarchar(MAX)	☑
Street	nvarchar(MAX)	☑
Turnover	int	☑
		☐

EF Code First workflow

- Best practice : Streef naar kleine stappen

Stap 1: domein model

- Pas “indien nodig” de ApplicationDbContext klasse aan
  - Bevat een **DbSet** voor elke domeinklasse waarvoor een overeenkomstige tabel in de database bestaat

```
public class ApplicationDbContext : DBContext
{
   public DbSet<Brewer> Brewers { get; set; }
}
```

    - “indien nodig”:
      - EF doet aan type discovery (zie verder)
      - Aggregate roots (zie verder)

Stap 1: domein model

- **DbSet** (namespace System.Data.Entity)
  - ~ Repository
  - Lijst van (in memory) objecten van een bepaald type die de persistentielaag ter beschikking stelt
  - Deze lijst kan je bevragen a.d.h.v. Linq to Entities (zie verder)

Stap 2: genereer de database

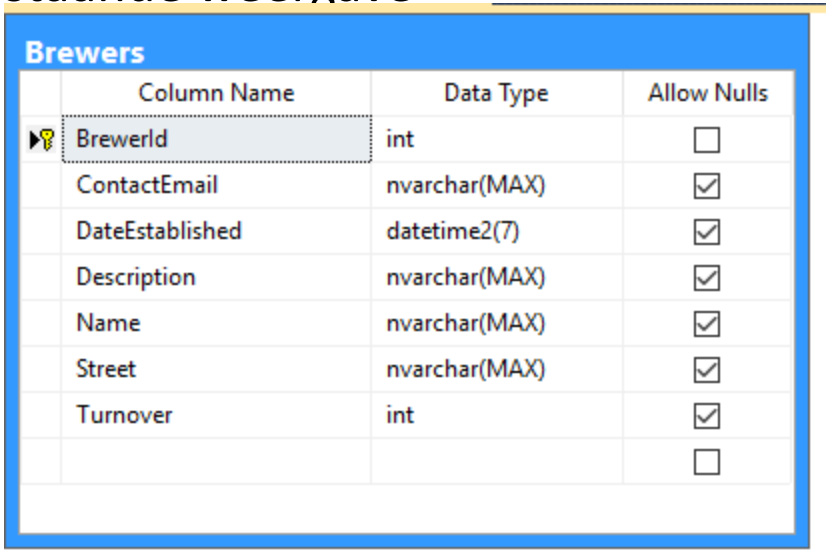
- Run de applicatie
  - De database bevat nu een tabel Brewers. EF Code First hanteert “**convention over configuration**”.

- Bekijk de database
  - **SQL Server Object Explorer** : Tabel Brewers : (dubbelklik toont ontwerp tabel)

	Name	Data Type	Allow Nulls
🔑	BrewerId	int	☐
	ContactEmail	nvarchar(MAX)	☑
	DateEstablished	datetime2(7)	☑
	Description	nvarchar(MAX)	☑
	Name	nvarchar(MAX)	☑
	Street	nvarchar(MAX)	☑
	Turnover	int	☑
			☐

Stap 2: genereer de database

- Bekijk de database
  - Of start **SQL Server Management Studio**
    - Connecteer met . of localhost
    - Je kan hier een ERD aanmaken van de tabellen
    - Klap Beerhall open, rechtsklik “Database Diagrams” > New database diagram. Selecteer de gewenste tabellen.
    - Selecteer de tabellen op het diagram, klik Tabel view > Standard voor onderstaande weergave



EF Conventies bij mappen naar db

- EF gebruikt voor het omzetten van een klasse naar een tabel in de database volgende conventies:
  - Elke klasse wordt een tabel
  - Elke property wordt een kolom in de tabel
  - Primary Keys
  - Enums

  - Opm: dit zijn conventies, we zullen later zien hoe we dit zelf kunnen customizeren

EF Conventies bij mappen naar db

- Conventie: een klasse wordt een tabel
  - **naam tabel**
    - Is die van de DbSet of indien geen DbSet, de naam van de klasse
  - de klasse moet voldoen aan volgende **voorwaarden**:
    - **public** visibility
    - **not sealed**
    - De klasse moet ook opgenomen zijn als DbSet property in de DbContext, of vernoemd worden in OnModelCreating of navigeerbaar zijn via navigational properties in een opgenomen model

EF Conventies bij mappen naar db

- Conventies: elke property wordt een kolom in de tabel
  - **naam kolom** wordt de naam van de property
    - voorbeeld: property **Turnover,** klasse Brewer=> kolom **Turnover,** tabel Brewers
  - **datatype kolom** (wordt gekozen door de Data Provider)

C# datatype	SQL server datatype	Allow Nulls
string	nvarchar(MAX)	yes
bool	bit	no
int	int	no
float	real	no
double	float	no
decimal	decimal(18, 2)	No
DateTime	Datetime2(7)	No

voor de nullable versies van deze types (bool?, int?, …) wordt Allow Nulls yes…

  - de property moet voldoen aan volgende **voorwaarden**:
    - moet getter en setter hebben
      - de setter hoeft niet public te zijn

EF Conventies bij mappen naar db

- Conventies: bepaling van de Primary Key
  - De property met naam **Id** of **<classname>Id** wordt de PK
    - niet hoofdlettergevoelig
    - voorbeeld:
      - In klasse Brewer: property Id, ID, BrewerId, BrewerID, …
  - Property van type **int, long, short**
    - autonummering: een **identity** kolom
  - Property van type **string**
    - geen autonummering, type nvarchar(450)
  - Property van type **Guid**
    - geen autonummering, type Guid

EF Conventies bij mappen naar db

- Conventies: Enum property
  - wordt gemapt naar een kolom van het type **int**
    - als property nullable is dan NULL allowed anders niet
    - het is de ordinale waarde van de enum die wordt opgeslaan in de tabel
  - er wordt **geen aparte tabel** voorzien met de mogelijke waarden van de enum
  - Voorbeeld: zie later klasse Course en Enum Language

- Conventies : Spatial data : Point type
  - Meer op https://docs.microsoft.com/en-us/ef/core/modeling/spatial

EF Conventies bij mappen naar db

- Conventies
  - Voorbeeld: EF mapping Brewer klasse -> Brewers tabel

```
public class Brewer
{
    #region Properties

    public int BrewerId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string ContactEmail { get; set; }
    public DateTime? DateEstablished { get; set; }
    public string Street { get; set; }
    public int? Turnover { get; set; }

    #endregion
}
```

mapped by convention

Brewers

	Column Name	Data Type	Allow Nulls
🔑	BrewerId	int	☐
	ContactEmail	nvarchar(MAX)	☑
	DateEstablished	datetime2(7)	☑
	Description	nvarchar(MAX)	☑
	Name	nvarchar(MAX)	☑
	Street	nvarchar(MAX)	☑
	Turnover	int	☑
			☐

EF Conventies bij mappen naar db

- Conventies
  - Oefening : Hoe wordt de klasse Beer gemapt?

```
public class Beer
{
    #region Properties

    public int BeerId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public double? AlcoholByVolume { get; set; }
    public bool AlcoholKnown => AlcoholByVolume.HasValue;
    public decimal Price { get; set; }

    #endregion
}
```

EF Code First workflow

- Voldoet de mapping? Fluent API to the rescue

Entity Framework Core – To a new database (Code First workflow)

Fluent API

HoGent

Fluent API

- Voldoet de (gegenereerde) db niet dan kan je per klasse opgeven hoe de klasse gemapt dient te worden naar een tabel in de database
  - Maak in de Data folder een folder Mapping
  - Per klasse die gemapt wordt naar een tabel maak je een klassen aan
    - Implementeer de interface IEntityTypeConfiguration<T>
    - Configure methode : definieert de mapping adhv **Fluent API**

```
namespace BeerhallEF.Data.Mapping
{
    class BrewerConfiguration : IEntityTypeConfiguration<Brewer>
    {
        public void Configure(EntityTypeBuilder<Brewer> builder)
        {

        }
    }
}
```

Fluent API

- In de Klasse **ApplicationDbContext**, methode **OnModelCreating** geef je deze klasse op
  - Wordt aangeroepen bij aanmaken van de eerste instantie van de context. Dit wordt dan gecached, dus alle andere instances maken hier gebruik van
  - **ModelBuilder:** definieert het model. Dit wordt gebruikt om de CLR classes te mappen met database schema.

ApplicationDbContext.cs

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.ApplyConfiguration(new BrewerConfiguration());
}
```

Fluent API

- Mappen van een klasse naar een tabel in de database
  - Mappen van tabelnaam
  - Mappen van primary key
  - Mappen van de properties ⇔ kolommen
  - Enkel de zaken die afwijken van de conventies dien je in de mapping op te nemen.
  - Je kan ook opgeven dat een klasse niet gemapt dient te worden naar een tabel

Fluent API

- Mappen van tabelnaam/Uitsluiten van properties

ToTable(String)	Configures the table name to be mapped to.
Ignore<TProperty>	Excludes a property from the model so that it will not be mapped to the database.

  - Voorbeeld

```
class BrewerConfiguration : IEntityTypeConfiguration<Brewer>
{
    public void Configure(EntityTypeBuilder<Brewer> builder)
    {
        builder.ToTable("Brewer");
        builder.Ignore(t => t.ContactEmail);
    }
}
```

Fluent API: Mappen klasse

- Definiëren van de sleutel

HasKey<TKey>	Configures the primary key property(s) for this entity type.

- Enkelvoudige sleutel : bestaande uit 1 property

```
class BrewerConfiguration : IEntityTypeConfiguration<Brewer>
{
    public void Configure(EntityTypeBuilder<Brewer> builder)
    {
        builder.ToTable("Brewer");
        //Mappen primary key
        builder.HasKey(t => t.BrewerId);
    }
}
```

- Samengestelde sleutel : bestaande uit meerdere properties.
  Bvb een OrderLijn heeft een samengestelde sleutel bestaande uit OrderId en ProductId

```
b.HasKey(t => new {t.OrderId, t.ProductId});
```

Fluent API: Mappen klasse

- Mappen van properties naar kolommen
  - HasColumnName(*naam*): opgeven van de kolomnamen
  - IsRequired(*true/false*): Optioneel of verplicht (NOT NULL)
  - HasMaxLength(*maxlengte*): de maximale lengte van de kolom => *nvarchar(maxlength*)
  - HasColumnType(*type*): SQL type van de kolom opgeven
  - HasDefaultValue(*waarde*): defaultwaarde kolom bij creatie record
  - HasDefaultValueSQL(*sql fragment*): sql fragment voor berekenen van de defaultwaarde
  - HasComputedColumn(*sql-expressie*): voor een berekende kolomwaarde. Hier geef je SQL server expressie op
  - HasField(*field*): EF zal de waarde automatisch aan field toekennen ipv via de setter van de property

Fluent API: Mappen klasse

- Mappen van properties naar kolommen
  - Voorbeeld

```
class BrewerConfiguration : IEntityTypeConfiguration<Brewer>
{
    public void Configure(EntityTypeBuilder<Brewer> builder)
    {
        builder.ToTable("Brewer");

        //Mappen primary key
        builder.HasKey(t => t.BrewerId);

        //properties
        builder.Property(t => t.Name)
            .HasColumnName("BrewerName")
            .IsRequired()
            .HasMaxLength(100);

        builder.Property(t => t.ContactEmail)
            .HasMaxLength(100);

        builder.Property(t => t.Street)
            .HasMaxLength(100);
    }
}
```

Fluent API: Mappen klasse

- Mappen van properties naar kolommen
  - Generated Properties
    - Conventie: Voor primary keys van type int of Guid genereert de database een waarde bij toevoegen (autonummering of identity)
    - Fluent API
      - ValueGeneratedNever(): database genereert geen waarde
      - ValueGeneratedOnAdd(): De database genereert een waarde bij toevoegen.
      - ValueGeneratedOnAddOrUpdate(): De database genereert een waarde bij elke opslag. (Gebruikt voor concurrency, zie verder)
    - Voorbeeld :

```
builder.Property(t => t.BrewerId)
    .ValueGeneratedOnAdd();
```

Fluent API: Mappen klasse

- Mappen van properties naar kolommen
  - Read-only properties
    - Conventie: Properties met enkel een getter worden “by convention” niet gemapped naar een kolom in de database.
    - Oplossing
      - Gebruik een private setter
      - OF igv readonly property:
        zorg voor een expliciete mapping
        en voorzie een constructor. Voor de
        argumenten geldt dat de naam en
        type argument = naam en type property)
    - Voorbeeld : zie later Location klasse.

```
builder.Property(t => t.PostalCode);
builder.Property(t => t.Name);
```

```
public class Location
{
    #region Properties
    public string PostalCode { get; }
    public string Name { get; }
    #endregion

    #region Constructor
    public Location(string postalCode, string name)
    {
        PostalCode = postalCode;
        Name = name;
    }
    #endregion
}
```

Fluent API: Mappen klasse

- Mappen van properties naar kolommen
  - Concurrency
    - In multi-user omgeving, indien meerdere gebruikers dezelfde records wijzigen
    - Last-in-wins updating :
      - Enkel gebruiken indien kans op collision heel klein

“update Brewers set name=@name, street=@street, ... where brewerId=@ brewerId";

    - Timestamp-based updating (optimistische locking)
      - In tabel hou je timestamp (of rowversion) kolom bij met tijdstip laatste wijziging. Indien deze gewijzigd is sinds het opvragen van gegevens (daar haal je timestamp ook op), dan mislukt de update

“update Brewers set name=@name, street=@street, timeStamp=@timeStamp, ...

where brouwernr=@brouwernr and timeStamp=@oldTimeStamp;

      - Ofwel eigen kolom definiëren hiervoor (IsConcurrencyToken)

Fluent API: Mappen klasse

- Concurrency
  - Default: last in wins.
  - Optimistic concurrency: geen locking tussen opvragen en update, wel controle van bepaalde properties mogelijk bij update.
    - Gebruik hiervoor best een TimeStamp kolom :
      - in C# datatype byte[].
    - De mapping

```
In klasse Brewer{
        public Byte[] Timestamp { get; set; }

In de methode Configure
                builder.Property(p => p.Timestamp)
                .ValueGeneratedOnAddOrUpdate()
                .IsConcurrencyToken();
```

Fluent API: Mappen klasse

- Klassen die niet in de database voorkomen
  - Voorbeeld: klasse XXX mag niet gemapt worden naar een tabel in de database

ApplicationDbContext.cs

```
public class ApplicationDbContext : DbContext {
. . .
protected override void OnModelCreating(DbModelBuilder modelBuilder) {
  modelBuilder.Ignore<XXX>();
} }
```

  - Zie documentatie: including/excluding Types

Fluent API: Mappen klasse

STAP 1

Maak domein model aan

```
using System;

namespace BeerhallEF.Models
{
    public class Brewer
    {
        #region Properties

        public int BrewerId { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public string ContactEmail { get; set; }

        public DateTime? DateEstablished { get; set; }

        public int? Turnover { get; set; }

        public string Street { get; set; }

        #endregion

    }
}
```

STAP 2

Definieer de mapping

```
public class ApplicationDbContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)...

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.ApplyConfiguration(new BrewerConfiguration());
    }

    public DbSet<Brewer> Brewers { get; set; }
}


    class BrewerConfiguration : IEntityTypeConfiguration<Brewer>
    {
        public void Configure(EntityTypeBuilder<Brewer> builder)
        {
            builder.ToTable("Brewer");

            //Mappen primary key
            builder.HasKey(t => t.BrewerId);

            //properties
            builder.Property(t => t.Name)
                .HasColumnName("BrewerName")
                .IsRequired()
                .HasMaxLength(100);

            builder.Property(t => t.ContactEmail)
                .HasMaxLength(100);

            builder.Property(t => t.Street)
                .HasMaxLength(100);

            builder.Property(t => t.BrewerId)
                .ValueGeneratedOnAdd();
        }
    }
```

STAP 3

Drop/create database

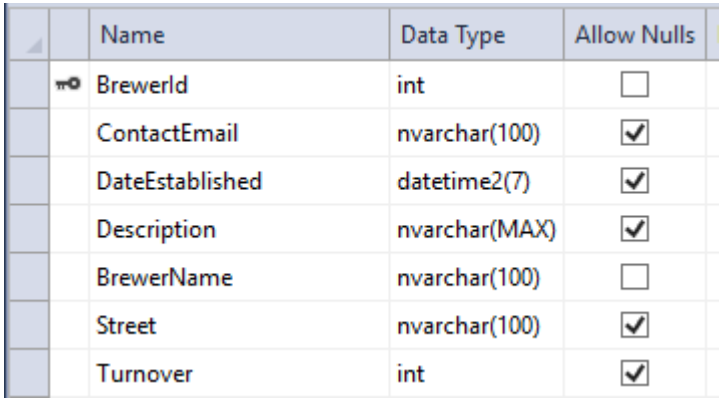
	Name	Data Type	Allow Nulls
🔑	BrewerId	int	☐
	ContactEmail	nvarchar(100)	☑
	DateEstablished	datetime2(7)	☑
	Description	nvarchar(MAX)	☑
	BrewerName	nvarchar(100)	☐
	Street	nvarchar(100)	☑
	Turnover	int	☑

Fluent API: Mappen klasse

- Fluent api
  - Voorbeeld: EF mapping Brewer klasse -> Brewer table

```
public class Brewer
{
    #region Properties

    public int BrewerId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string ContactEmail { get; set; }
    public DateTime? DateEstablished { get; set; }
    public string Street { get; set; }
    public int? Turnover { get; set; }

    #endregion
}
```

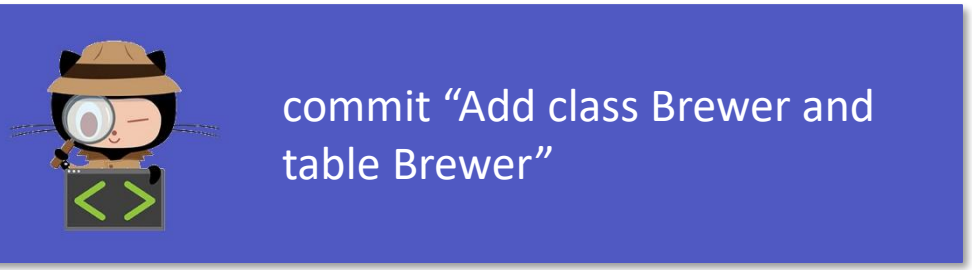
Mapped with fluent api



	Name	Data Type	Allow Nulls
🔑	BrewerId	int	☐
	ContactEmail	nvarchar(100)	☑
	DateEstablished	datetime2(7)	☑
	Description	nvarchar(MAX)	☑
	BrewerName	nvarchar(100)	☐
	Street	nvarchar(100)	☑
	Turnover	int	☑

Fluent API: Mappen klasse

- Time to Commit : Create class Brewer and table Brewer



Fluent API

- Oefening 1 : (Stappen tussen [] kan je skippen)
  - Voeg de klasse Beer toe aan het project (zie volgende slide)
  - Voeg DbSet voor Beer toe
  - [Run de applicatie]
  - [Bekijk de gegenereerde database]
  - Voeg mapping toe zodat het resultaat als volgt gemapt wordt
  - Run de applicatie

```
public class Beer
{
    #region Properties

    public int BeerId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public double? AlcoholByVolume { get; set; }
    public bool AlcoholKnown => AlcoholByVolume.HasValue;
    public decimal Price { get; set; }
```

	Name	Data Type	Allow Nulls
🔑	BeerId	int	☐
	AlcoholByVolume	float	☑
	Description	nvarchar(MAX)	☑
	Name	nvarchar(100)	☐
	Price	decimal(18,2)	☐
			☐

Fluent API

- Oefening 1 :

```
public class Beer
{
   #region Properties
   public int BeerId { get; set; }
   public string Name { get; set; }
   public string Description { get; set; }
   public double? AlcoholByVolume { get; set; }
   public bool AlcoholKnown => AlcoholByVolume.HasValue;
   public decimal Price { get; set; }
   #endregion

   #region Constructors
   protected Beer()
   {
   }
   public Beer(string name) : this()
   {
      Name = name;
   }
   #endregion
}
```

	Name	Data Type	Allow Nulls
🔑	BeerId	int	☐
	AlcoholByVolume	float	☑
	Description	nvarchar(MAX)	☑
	Name	nvarchar(100)	☐
	Price	decimal(18,2)	☐
			☐

Merk op :
Een bier moet een naam en prijs hebben bij creatie => constructor
EF gebruikt deze constructor als de parameter overeenkomt met een parameter/property conventie. Anders dien je een default constructor te voorzien (kan je protected maken). Dit geldt niet voor associaties

Entity Framework Core – To a new database (Code First workflow)

Associaties

HoGent

EF : Associaties

- Conventies
  - “By convention, a relationship will be created when there is a **navigation property** discovered on a type. A property is considered a navigation property *if the type it points to can not be mapped as a scalar type by the current database provider*.”
  - Relationships that are discovered by convention will always target the primary key of the principal entity. To target an alternate key, additional configuration must be performed using the Fluent API.

EF: Associaties

- Terminologie
  - **Principal/Primary key:** De property(s) that uniquely identifies the principal entity. This may be the primary key or an alternate key.
  - **Foreign key:** The property(s) in the dependent entity that is used to store the values of the principal key property that the entity is related to.
  - **Principal entity:** This is the entity that contains the primary/alternate key property(s). Sometimes referred to as the 'parent' of the relationship.
  - **Dependent entity:** This is the entity that contains the foreign key property(s). Sometimes referred to as the 'child' of the relationship.
  - **Navigation property:** A property defined on the principal and/or dependent entity that contains a reference(s) to the related entity(s).
  - **Collection navigation property:** A navigation property that contains references to many related entities.
  - **Reference navigation property:** A navigation property that holds a reference to a single related entity.
  - **Inverse navigation property:** When discussing a particular navigation property, this term refers to the navigation property on the other end of the relationship.

EF : Associaties

- Verschillende soorten associaties in domein
  1. Bi-directionele associaties (fully defined relationships)
  2. Geen Foreign Key property
  3. Associaties in 1 richting (single navigation property)

1

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

2

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

3

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}
```

EF Associaties : mapping conventies

- Conventies: associaties worden relaties in de db
  - Een **navigation property** moet voldoen aan volgende **voorwaarden**
    - **public**
      - de setter mag private zijn, hoeft zelf niet aanwezig te zijn.
    - een **collection** moet het type **ICollection<T>** implementeren
      - je instantieert de collection in de default constructor
  - wordt default gemapt naar
    - een **1:n** relatie.
    - De **FK** met de ***naam <principal key property name>*** wordt in de dependant tabel toegevoegd en **ALLOWS NULL**
    - **Cascading Delete** is ***None***

EF Associaties : mapping conventies

- Conventies: associaties worden relaties
  - Instantiatie collections: HashSet<T> versus List<T>:

HashSet<T>	List<T>
geen duplicaten toegestaan • override eventueel Equals en GetHashCode inklasse T • methode Add(T) retourneert een boolean	**duplicaten** toegestaan
geen volgorde op elementen bv. Add(T), Contains(T)	**volgorde** op elementen bv. Insert(index, T), IndexOf(T), …
enorm snel te bepalen of element tot de collectie behoort, element toevoegen, weghalen	**trager** te bepalen of een element tot de collectie behoort, element toevoegen, weghalen

  - https://stackoverflow.com/questions/150750/hashset-vs-list-performance

EF Associaties : mapping conventies

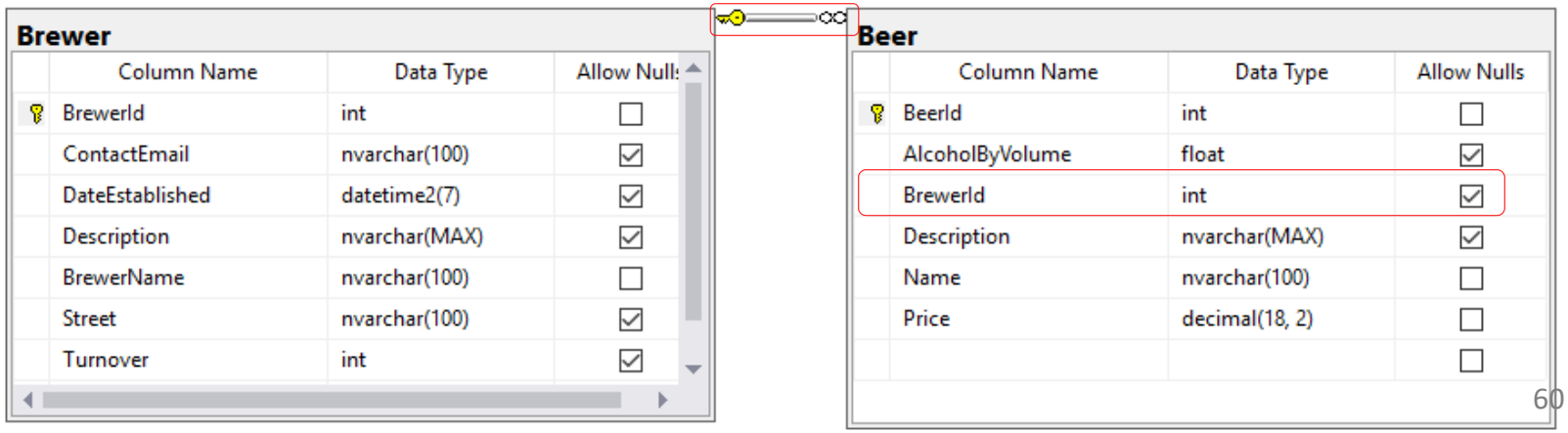
- Voorbeeld
  - Pas de klasse Brewer aan.
    - Voeg onderstaande properties toe

```
public ICollection<Beer> Beers { get; private set; }

public int NrOfBeers => Beers.Count;
```

    - Instantieer de associatie in de default constructor

```
Beers = new HashSet<Beer>();
```

mapped by convention



EF Code-first workflow

STAP 1

Voeg associatie toe

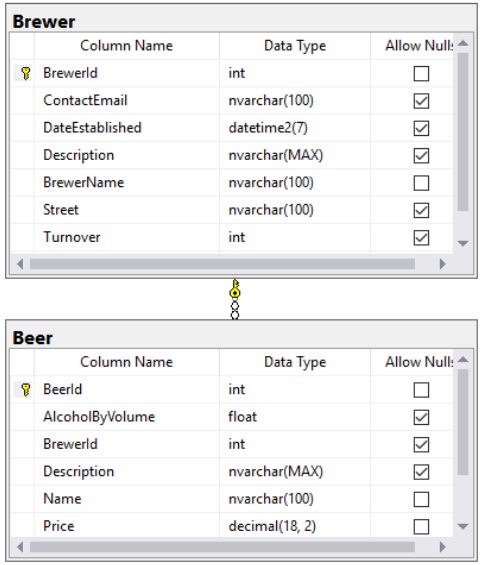
```
public ICollection<Beer> Beers { get; private set; }

public int NrOfBeers => Beers.Count;
```

STAP 2

Voeg mapping toe

STAP 3

PM>update-database



Fluent API: mappen associaties

- 1:n
  - Identificeer de **navigation property** die een associatie definieert
    - Vb. property Beers in klasse Brewer
    - Deze associatie moet volgende relatie worden in de database :
      - “Een Brouwer is gekoppeld aan meerdere Bieren, maar een Bier is gekoppeld aan exact 1 Brouwer”
      - Mappen gebeurt in 4 stappen

1. Definieer het eerste deel van de relatie, hier “een Brouwer is gekoppeld aan 0, 1 of meerdere bieren”.
   HasOne()/HasMany(): geef de navigation property waarvoor je relatie mapt op

   - Vb. `builder.HasMany(t=>t.Beers)`

Fluent API: mappen associaties

- 1:n
  2. Dan koppel je terug met **WithOne()/WithMany()** voor de omgekeerde richting.
     - Voorbeeld : Een Brouwer is gekoppeld aan meerdere Bieren, *MAAR een Bier is gekoppeld aan 1 Brouw*er
     - **Igv associatie in 1 richting**: gebruik de *parameterless* overload

```
builder.HasMany(t=>t.Beers)
       .WithOne()
```

     - **Igv bidirectionele associatie** : geef als *parameter de navigation property* op.
       - Vb. Stel dat de Beer klasse een property Brewer zou hebben

```
builder.HasMany(t=>t.Beers)
   .WithOne(t=>t.Brewer)
```

Fluent API: mappen associaties

- 1:n
  3. **IsRequired(true/false)** : een verplichte relatie. Dit bepaalt of de FK kolom al dan niet NULL mag zijn.
     - Voorbeeld :

```
builder.HasMany(t=>t.Beers)
    .WithOne()
    .IsRequired()
```

Fluent API: mappen associaties

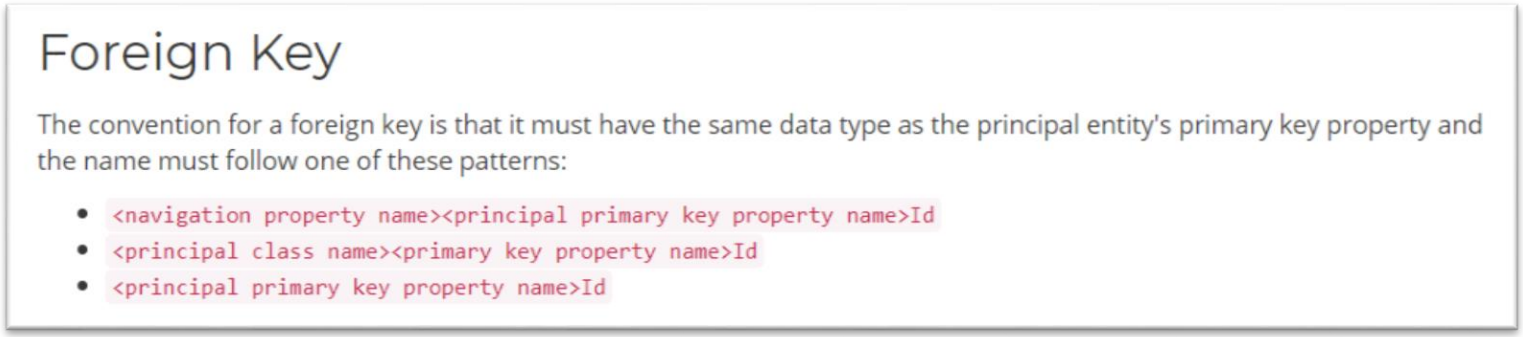
- 1:n
  4. **OnDelete**
     - **Cascade:** Gerelateerde entities worden ook verwijderd.
     - **ClientSetNull (default):** De foreign key properties in dependent entities worden op null geplaatst (enkel voor nullable FK's). Enkel voor de childs geladen in memory, niet de childs in de database
     - **SetNull** : De foreign key properties in dependent entities worden op null geplaatst (enkel voor nullable FK's). Voor de childs geladen in memory en in de database (maar niet alle db laten dit toe)
     - **Restrict:** De delete operatie wordt niet toegepast als er gerelateerde entities zijn
     - Voorbeeld

```
//Mapping Associations
builder.HasMany(t => t.Beers)
    .WithOne()
    .IsRequired()
    .OnDelete(DeleteBehavior.Cascade);
```

Fluent API: mappen associaties

- 1:n
  5. Mogelijks nog een stap 5: Als de FK property bestaat en de naam niet de conventions volgt dan moet je dit expliciet mappen naar de FK kolom.

  - .HasForeignKey(t=>t.*FKProperty*)



Foreign Key

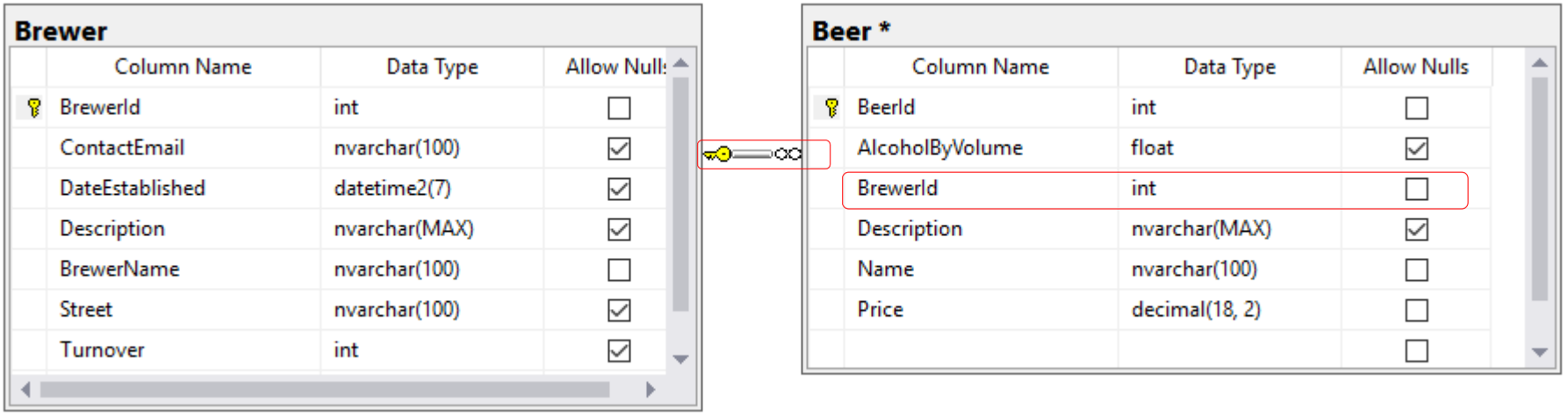
The convention for a foreign key is that it must have the same data type as the principal entity's primary key property and the name must follow one of these patterns:

- `<navigation property name><principal primary key property name>Id`
- `<principal class name><primary key property name>Id`
- `<principal primary key property name>Id`

Fluent API: mappen associaties

- Voorbeeld
  - Map de property Beers in Brewer. De relatie: 1..n, verplicht, FK BrouwerId, Cascading delete.
  - Run de applicatie

mapped by fluent api



EF Code-first workflow

STAP 1

Voeg associatie toe

```
public ICollection<Beer> Beers { get; private set; }

public int NrOfBeers => Beers.Count;
```

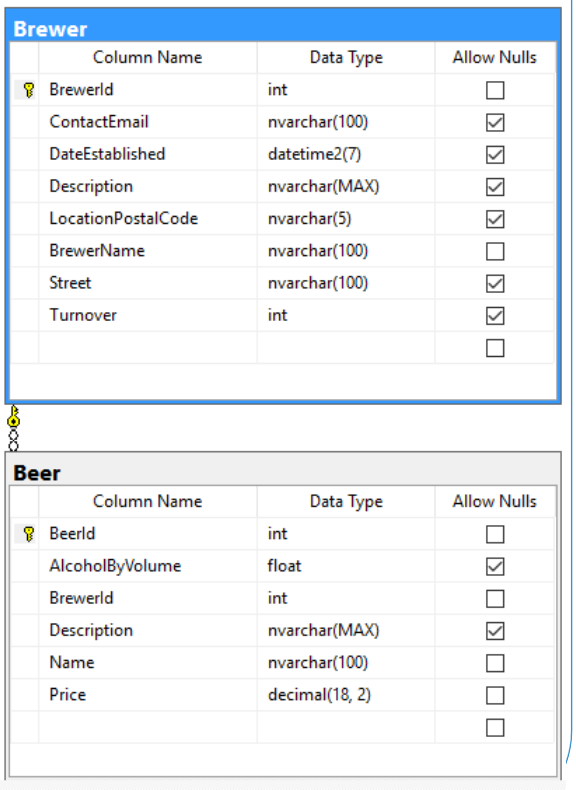
STAP 2

Voeg mapping toe

```
//Mapping Associations
builder.HasMany(t => t.Beers)
    .WithOne()
    .IsRequired()
    .OnDelete(DeleteBehavior.Cascade);
```

STAP 3

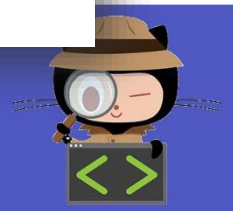
Genereer de database



Fluent API: mappen associaties

- DbSet
  - Doordat Brewer een ICollection<Beer> bevat, zal EF een table Beer aanmaken. Het volstaat een DbSet te voorzien voor het root element Brewer
  - Een DbSet laat toe om de bieren rechtstreeks op te vragen, toe te voegen, aan te passen en te verwijderen. Enkel als dit nodig is voor de applicatie voeg je de DbSet toe

```
public class ApplicationDbContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)...

    protected override void OnModelCreating(ModelBuilder modelBuilder)...

    public DbSet<Brewer> Brewers { get; set; }
    //public DbSet<Beer> Beers { get; set; }  type discovery
}
```



commit “Add class Beer and mapping”

Fluent API: mappen associaties

- Oefening

De properties bevatten geen setter
Je zal de properties expliciet moeten mappen!

  - Voeg de klasse **Location** toe.(Immutable class!)
  - Pas Brewer klasse aan. Voeg property Location toe
  - Voeg DbSet<Location> toe, daar we in de applicatie alle locaties wensen op te vragen
  - Map via de Fluent API
    - PostalCode is maximaal 5 posities en is de key
    - Naam is maximaal 100 posities en verplicht
    - Relatie: 1:N, optioneel, geen cascading delete
  - Run de applicatie en controleer de database. Merk op de naam van de FK <navigation property name><principal key property name>. Wens je de naam te veranderen voeg dan bvb .HasForeignKey("PostalCode") of .HasForeignKey(nameof(Location.PostalCode)) toe
  - Commit "Add class Location and mapping"

Fluent API: mappen associaties

- Oefening
  - Voeg de klasse **Course** en *enum* **Language** toe.
  - Pas Brewer klasse aan. Voeg property Courses (ICollection) toe, instantieer in de constructor. Merk op een bi-directionele associatie
  - Voeg DbSet toe
  - Map via Fluent API
    - Title is verplicht en maximaal 100 posities
    - relatie: 1:N, verplicht, cascading delete
  - Run
  - Commit “Add class Course and mapping”

Fluent API: mappen associaties

- 1:1
  - Zie documentatie https://docs.microsoft.com/en-us/ef/core/modeling/relationships

Fluent API: mappen associaties

- N:M
  - N:M zonder klasse die de “joined” table representeert wordt nog niet ondersteund door EF Core.
  - Je moet de klasse die de joined table voorstelt expliciteren en daarna map je de N:M als 2 1:N relaties.
  - Zie documentatie https://docs.microsoft.com/en-us/ef/core/modeling/relationships

Fluent API: mappen associaties

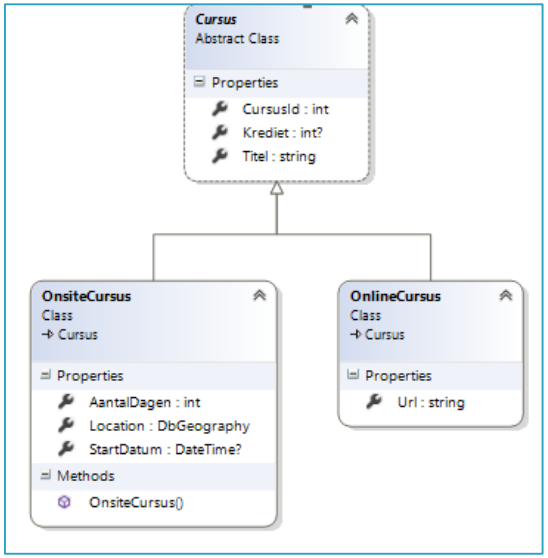
- Oefening
  - N:M
    - Neem eerst de documentatie door over N:M relaties: https://docs.efproject.net/en/latest/modeling/relationships.html#many-to-many
    - Voeg klasse Category toe. Een Category kent de brouwers, brouwers kent zijn categorieën niet. Creëer extra klassen indien nodig.
    - Voe de mapping toe.
    - Commit

Entity Framework Core – To a new database (Code First workflow)

Overerving

HoGent

EF Overerving : mapping conventies



- Conventies: overerving
  - Domein: overerving
    - Basisklasse: al dan niet abstract
    - **Voorbeeld** Course basisklasse. Online- en OnsiteCourse zijn subklassen.
  - DB: **Table per Hierarchy**
    - dit is de default voor mappen overerving
    - **1 tabel met de properties van alle klassen (basisklasse en subklassen)**
      - **Naam tabel** = naam base class (in meervoud)
      - Tabel bevat een extra kolom met de naam **Discriminator** (nvarchar(max)) , met als waarden de namen van de (afgeleide) klassen. Hierdoor kan je weten welke klasse het record voorstelt.

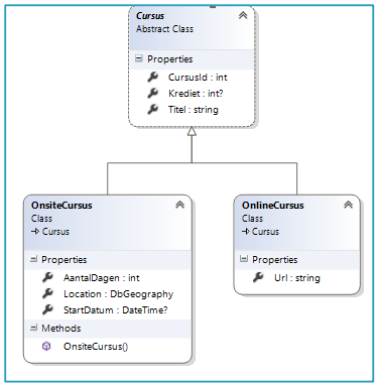
EF Overerving : mapping conventies

- Conventies: overerving
  - EF gaat niet zelf op zoek naar base klassen…
    - configurationklassen voorzien, of,
    - maak een DbSet aan voor de superklasse én de base klassen die je wens op te nemen in de hierarchie, of,

```
0 references | 0 changes | 0 authors, 0 changes
public DbSet<Course> Courses { get; set; }
0 references | 0 changes | 0 authors, 0 changes
public DbSet<OnlineCourse> OnlineCourses { get; set; }
0 references | 0 changes | 0 authors, 0 changes
public DbSet<OnsiteCourse> OnsiteCourses { get; set; }
```

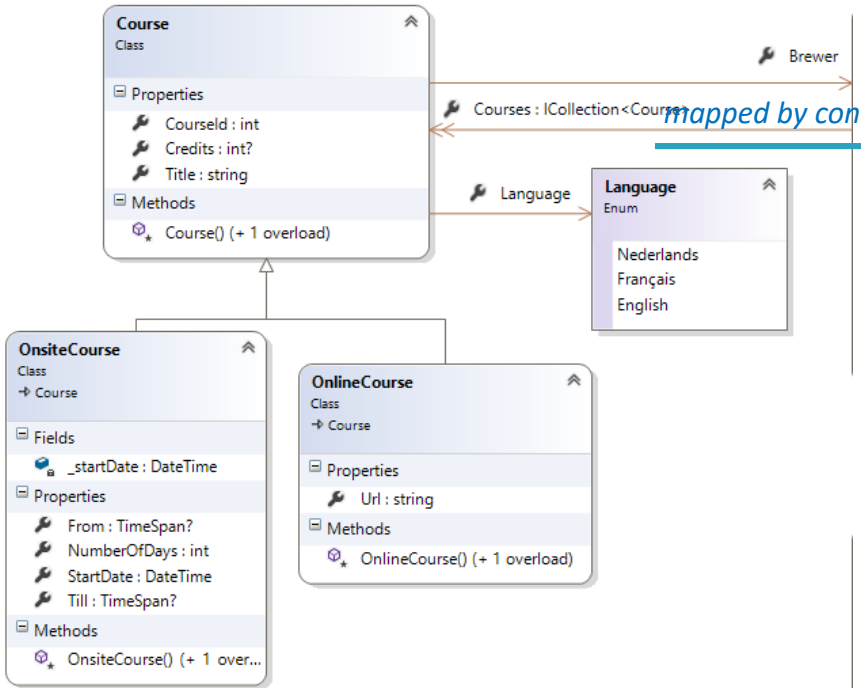
    - geef in OnModelCreating aan dat er subklassen zijn

```
modelBuilder.Entity<OnlineCourse>().HasBaseType<Course>();
modelBuilder.Entity<OnsiteCourse>().HasBaseType<Course>();
```



EF Overerving : mapping conventies

- Conventies: overerving
  - **voorbeeld**
    - Domain



mapped by convention

Database (1 tabel)

	Name	Data Type	Allow Nulls
🔑	CourseId	int	☐
	BrewerId	int	☐
	Credits	int	☑
	Discriminator	nvarchar(MAX)	☐
	Language	int	☐
	Title	nvarchar(MAX)	☑
	Url	nvarchar(MAX)	☑
	From	time(7)	☑
	NumberOfDays	int	☑
	StartDate	datetime2(7)	☑
	Till	time(7)	☑

Discriminator kolom bevat hier dus de waarde OnlineCursus of OnsiteCursus (indien Cursus een concrete klasse was dan kon de waarde ook mogelijks Cursus zijn). Kolom url zal bvb enkel een waarde hebben als Discriminator = OnlineCursus

Fluent api: mappen overerving

- De mapping https://docs.efproject.net/en/latest/modeling/relational/inheritance.html
  - CourseConfiguration

```
//Inheritance : TPH, and renaming the discriminator
builder.HasDiscriminator<string>("Type")
    .HasValue<OnlineCourse>("Online")
    .HasValue<OnsiteCourse>("Onsite");
```

```
public class OnlineCourseConfiguration : IEntityTypeConfiguration<OnlineCourse>
{
    public void Configure(EntityTypeBuilder<OnlineCourse> builder)
    {
        //Properties
        builder.Property(t => t.Url).HasMaxLength(100);
    }
}
```

```
public class OnsiteCourseConfiguration : IEntityTypeConfiguration<OnsiteCourse>
{
    public void Configure(EntityTypeBuilder<OnsiteCourse> builder)
    {
        //Properties
        builder.Property(t => t.StartDate)
            .HasField("_startDate");
    }
}
```

!! Als data uit de database wordt gelezen mag de setter niet worden uitgevoerd daar datums in het verleden kunnen liggen. Maak hiervoor gebruik van **BackingField**. EF zal gebruik maken van het attribuut _startDate ipv de setter van de property StartDate. Meer op https://docs.microsoft.com/en-us/ef/core/modeling/backing-field

EF extra's: Toevoegen van een index

- Toevoegen van indexen,….
  - Bvb unique Index op naam, voor een Bier
    Pas mapper aan

```
b.HasIndex(t => t.Name).IsUnique(true);
```

EF extra's: owned types

- 1:1
  - Stel bvb een klasse Address, zonder key. Brewer bevat een property StreetAddress van type Address.

```
public class Address
{
    public string Street { get; set; }
    public Location Location { get; set; }
}
```

```
public Address StreetAddress { get; set; }
```

  - Via owned types wordt geen nieuwe tabel aangemaakt voor address maar wordt address kolommen toegevoegd aan de tabel Brewer

```
builder.OwnsOne(t => t.StreetAddress);
```

    - Of als Address een private property is, gebruik dan de string versie
      - Builder.OwnsOne(typeOf(Address),"StreetAddress")

StreetAddress_Street	nvarchar(MAX)	☑
StreetAddress_LocationPost	nvarchar(5)	☑

EF extra's: owned types

- 1:n
  - Stel Brewer bevat een lijst van ShippingCenters

```
public ICollection<Address> ShippingCenters { get; set; }
```

  - De mapping

```
builder.OwnsMany(t => t.ShippingCenters);
```

  - Genereert een table Address met samengestelde sleutel:

	Name	Data Type	Allow Nulls
🔑	BrewerId	int	☐
🔑	Id	int	☐
	Street	nvarchar(MAX)	☑
	LocationPostalCode	nvarchar(5)	☑

  - Meer op https://docs.microsoft.com/en-us/ef/core/modeling/owned-entities

EF extra's: table splitting

- Mappen van meerdere klassen naar 1 tabel
  - Meer op https://docs.microsoft.com/en-us/ef/core/what-is-new/

Nog enkele tips

- DbContext
  - EF verzorgt de mapping van domain <-> database.
  - EF doet aan **Type Discovery**. Dit betekent dat EF alle klassen zal mappen
    - waarvoor **DbSet** gedefinieerd in klasse die erft van DbContext
    - of vermeld staan in de OnModelCreating methode
    - waarnaar verwezen wordt via navigation properties
    - Je kan een klasse uitsluiten via de fluent api : .Ignore
    - Voorbeeld : Stel DbSet voor Brewer gedefinieerd => dan wordt tabel Brewers aangemaakt/gemapt, maar door de navigation properties ook tabel Beers, Locations en Courses. Online- en OnsiteCourses, Categories wordt niet aangemaakt/gemapt tenzij je hier ook een DbSet voor voorziet of ze vermeld in OnModelCreating
  - Map enkel de **aggregate** roots
    - Aggregate = “a cluster of associated objects that we treat as a unit for the purpose of data changes”

Nog enkele tips

- Entity types met constructors
  - Wanneer EF Core instanties van een type maakt, zoals bvb bij het ophalen van een brouwer, wordt eerst de default constructor aangeroepen en wordt vervolgens elke property ingesteld via de setter op de waarde uit de database. Als EF Core echter een geparametriseerde constructor vindt met parameternamen en -typen die overeenkomen met die van de properties, roept deze in plaats daarvan de geparametriseerde constructor met waarden voor die properties aan en wordt dan elke property die niet in de constructor voorkomt expliciet ingesteld.
  - Niet alle properties vereisen constructor parameters. EF Core zal deze instellen na het aanroepen van de constructor via de setter van de property
  - De parameter types en namen in de constructor moeten matchen met de property types en namen, behalve dan dat properties Pascal-cased zijn, terwijl parameters camel-cased zijn
  - EF Core kan geen navigation properties instellen via de constructor!!! In klasse Course bvb hebben we een default constructor nodig, anders krijg je een runtime fout : System.InvalidOperationException: No suitable constructor found for entity type 'Course'. The following constructors had parameters that could not be bound to properties of the entity type: cannot bind 'brewer' in 'Course(string title, Language language, Brewer brewer)'.

Seeding van de database

Branch : Seeding-Querying-saving-data

HoGent

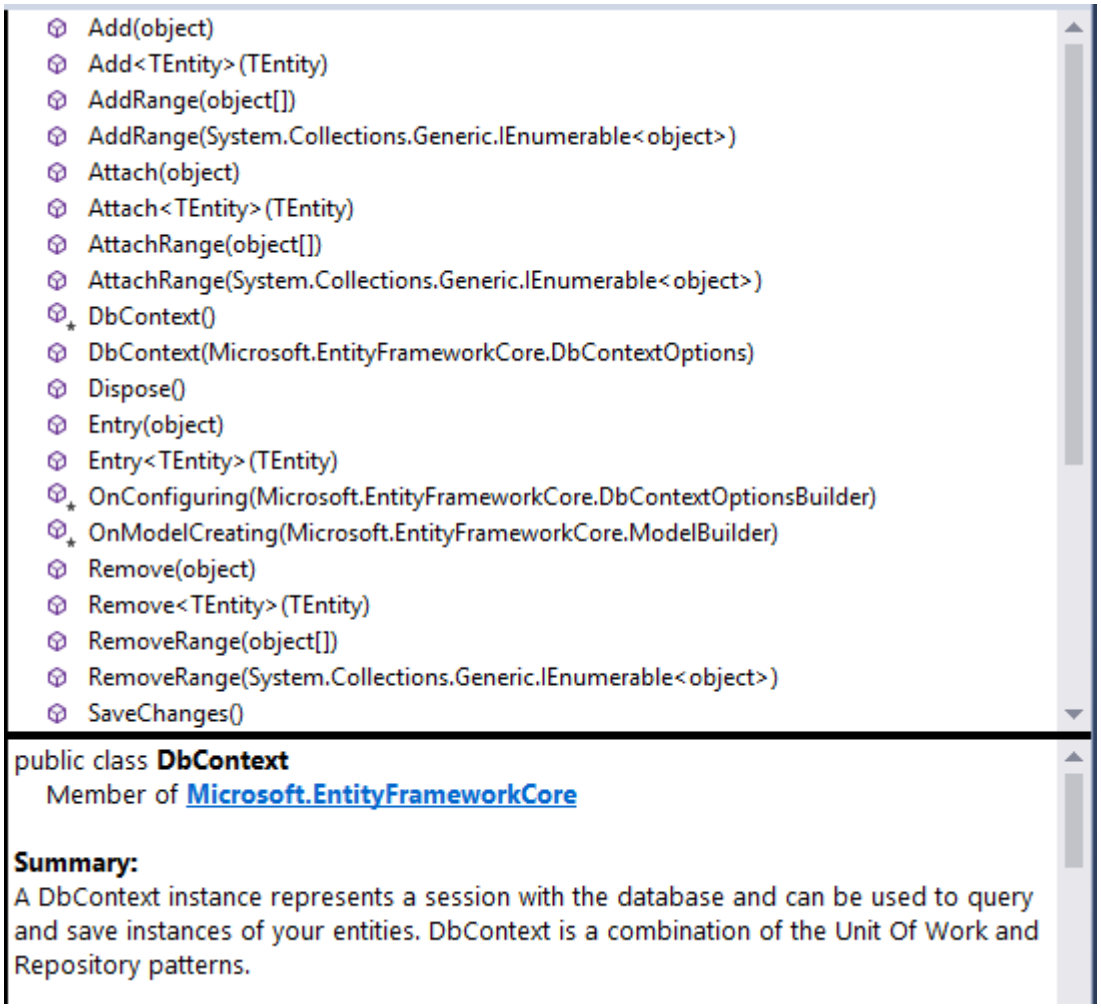
Seeden van de database

- Ga eerst naar de branch “Seeding-Querying-saving-data”

- Seeden = Vullen van de database met data.
  - De klasse Brewer werd aangepast en bevat nu extra methodes
    - Constructors
    - AddBeer
    - AddCourse
    - DeleteBeer

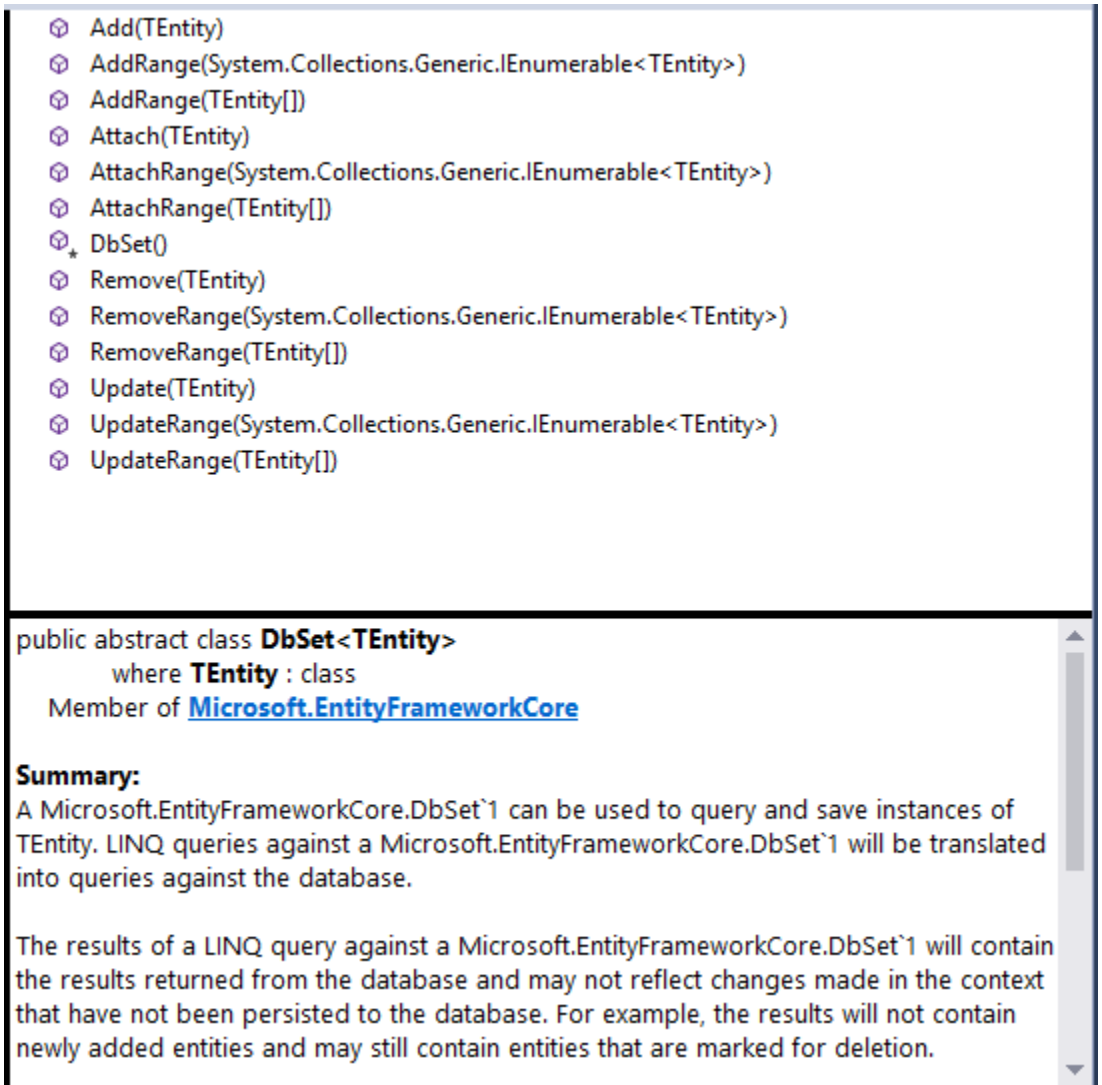
Seeden van de database

- **DbContext**



Seeden van de database

- **DbSet**



Seeden van de database

- De klasse BeerhallDataInitializer in de Data folder

```
public class BeerhallDataInitializer
{
    private readonly ApplicationDbContext _context;

    public BeerhallDataInitializer(ApplicationDbContext context)
    {
        _context = context;
    }

    public void InitializeData()
    {
        if (!_context.Locations.Any())
        {
            Location bavikhove = new Location { Name = "Bavikhove", PostalCode = "8531" };
            Location roeselare = new Location { Name = "Roeselare", PostalCode = "8800" };
            Location puurs = new Location { Name = "Puurs", PostalCode = "2870" };
            Location leuven = new Location { Name = "Leuven", PostalCode = "3000" };
            Location oudenaarde = new Location { Name = "Oudenaarde", PostalCode = "9700" };
            Location affligem = new Location { Name = "Affligem", PostalCode = "1790" };
            Location gent = new Location { Name = "Gent", PostalCode = "9000" };
            Location[] gemeenten =
                { bavikhove, roeselare, puurs, leuven, oudenaarde, affligem,gent };
            _context.Locations.AddRange(gemeenten);
            _context.SaveChanges();

            Brewer bavik = new Brewer("Bavik", bavikhove, "Rijksweg 33");
            _context.Brewers.Add(bavik);
            bavik.AddBeer("Bavik Pils", 0.4m);
            bavik.AddBeer("Wittekerke", 1.0m);
            bavik.AddBeer("Wittekerke Speciale", 1.8m);
            bavik.AddBeer("Wittekerke Rosé", 1.3m);
```

Bekijk de code.

Als de locaties reeds bestaan, worden ze niet opnieuw gecreëerd

Aan een DbSet kan je nieuwe objecten toevoegen

SaveChanges voegt de Locaties toe aan de database

Context doet aan ChangeTracking, houdt alle wijzigingen bij tot je SaveChanges aanroept. Zal dan voor alle wijzigingen INSERT, UPDATE of DELETE instructie creëren=> **Transactie**

Seeden van de database

- Aanroepen van de BeerhallDataInitializer
  - Program.cs

```
public static void Main(string[] args)
{

   using (ApplicationDbContext context = new ApplicationDbContext())
   {
      //context.Database.EnsureDeleted();
      //context.Database.EnsureCreated();
      new BeerhallDataInitializer(context).InitializeData();
      Console.WriteLine("Database geinstantieerd");
   }
}
```

Als je telkens met dezelfde database (met dezelfde data) wenst te starten, kan je de database eerst verwijderen en daarna terug creëren.

Querying Data

HoGent

Linq to Entities

- LINQ to Entities
  - Bevragen van de database
  - Communicatie met de DbContext
    - Aggregaties, projecties, filteren, sorteren, … mogelijk
  - Strongly typed queries!
  - Retourneert: Entiteiten
- Achter de schermen genereert EF de overeenkomstige queries. Kan je bekijken in
  - Een profiler
    - SQL Server Profiler: openen via Tools menu in SQL Server Management Studio
  - Een logger klasse die je aan je project toevoegt : https://docs.microsoft.com/en-us/ef/core/miscellaneous/logging

Querying data

- Linq to Entities
  - **Basic Query** (zie ook opm op volgende slides)
  - **Loading Related Data** – Eager Loading en explicit loading(zie ook opm op volgende slides)
  - **Client vs. Server Evaluation**, lees ook https://docs.microsoft.com/en-us/ef/core/what-is-new/ef-core-3.0/ Restricted client evaluation
  - **Tracking vs. No-Tracking**
  - **Raw SQL Queries**
  - **How Query Works**

- Neem de documentatie door op https://docs.microsoft.com/en-us/ef/core/querying/
- Vervolledig de overeenkomstige oefeningen in Program.cs

Querying data

- Zie program.cs,
  - plaats QueryData(context); uit commentaar
- Wens je de gegenereerde query te zien, maak dan gebruik van de ApplicationDbContextWithLogging klasse.
  - Meer op https://docs.microsoft.com/en-us/ef/core/miscellaneous/logging

```
using (ApplicationDbContext context = new ApplicationDbContextWithLogging())
```

- De queries worden gelogd in de Console

Querying data

- Het resultaat

```
---Loading all brewers, ordered by name---
Bavik
De Graal
De Leeuw
Duvel Moortgat
InBev
Palm Breweries
Roman

---Loading the brewer with id 1---
Brewer with id 1: Bavik

---Filtering the brewers:  brewers whose name starts with b---
Bavik

---Filtering the brewers:  brewers from Leuven--
InBev

---Filtering the brewers: brewers with more than 4 beers, ordered by name---
Bavik
Duvel Moortgat
InBev

----Filtering the brewers: brewers with a beer starting with the letter B. ---
Bavik
InBev
Roman

---All beers from brewer with id 1---
Bavik Pils  0,4
Wittekerke  1,0
Wittekerke Speciale  1,8
Wittekerke Rosé  1,3
Ezel Wit  1,8
Ezel Bruin  2,5
```

```
---All brewers from Leuven, print the name and the number of beers---
InBev 5

---All brewers from Leuven, print the name and the number of beers - Use projections---
InBev 5

---Loading multiple relationships: all brewers, print name, location and number of beers--
Bavik                          Bavikhove                      6
De Graal                                                      0
De Leeuw                                                      0
Duvel Moortgat                     Puurs                      6
InBev                             Leuven                      5
Palm Breweries                                                4
Roman                          Oudenaarde                     4

---Including multiple levels: All brewers from the first category---
Bavik
Palm Breweries
Roman

---Explicit loading: all english courses from bavik--
Brewing beer Advanced

---Explicit loading: all courses from bavik--
Brewing beer Advanced
Bierbrouwen basis

---Inheritance--
Brewing beer Advanced

---All brewers with NrOfBeers > 4--
InBev: 5
Bavik: 6
Duvel Moortgat: 6

---Add: Create Brewer Gentse Gruut, Rembert Dodoensdreef, 9000 Gent ---

---Add in ICollection : add Course 'Hoppe'---

---Update : Give Gentse Gruut a new address in Roeselare---
```

```
---Delete : remove Gentse Gruut---
Number of brewers before delete: 8
Number of brewers after delete: 7

---Transactions, multiple operations in 1 save, change the turnc

---Create Brewer De Koninck, Mechelsesteenweg 291, 2018 Antwerpe
Number of cities before insert:7
Number of cities after insert:8

---Removing relationships: Remove the first Beer from Bavik - De
```

Basic queries : enkele opm

- **Single(OrDefault)/First(OrDefault)**
  - Opvragen 1 brouwer (vb met id 1)

```
_brewer = context.Brewers
    .SingleOrDefault(b => b.BrewerId == 1);
```

```
SELECT TOP(2) [b].[BrewerId], [b].[ContactEmail], [b].[DateEstablished], [b].[Description], [b].[LocationPostalCode],
[b].[BrewerName], [b].[Street], [b].[Turnover]
FROM [Brewers] AS [b]
WHERE [b].[BrewerId] = 1
```

- SingleOrDefault(): retourneert null als brouwer niet bestaat. Throwt exception als er meer dan 1 brouwer aan criterium voldoet
- Single() : Exception als brouwer niet bestaat. Throwt exception als er meer dan 1 brouwer aan criterium voldoet
- FirstOrDefault(): neemt eerste brouwer die aan criterium voldoet. Null als brouwer niet bestaat. Geeft geen fout als er meerdere brouwers aan criterium voldoen. (is performanter dan SingleOrDefault)
- First(): Exception als geen brouwers gevonden
- Merk op: Select Top(2) laat toe om de nodige controles te doen

Basic queries : enkele opm

- Filteren: **Where**
  - 1 klasse

```
Console.WriteLine("\n---Filtering the brewers:  brewers whose name starts with b---");
_brewers = context.Brewers
   .Where(b => b.Name.StartsWith("b"))
   .OrderBy(b => b.Name)
   .ToList();
```

```
SELECT [b].[BrewerId], [b].[ContactEmail], [b].[DateEstablished], [b].[Description], [b].[LocationPostalCode], [b]
.[BrewerName], [b].[Street], [b].[Turnover]
FROM [Brewers] AS [b]
WHERE [b].[BrewerName] LIKE N'b' + N'%'
ORDER BY [b].[BrewerName]
```

  - Opm : als EF3.0 een where conditie die het niet kan vertalen naar een SQL Query dan throwt het een exception. Oplossing switch naar IEnumerable en gebruik dan Linq to Objects

```
var specialCustomers =
  context.Customers
    .Where(c => c.Name.StartsWith(n))
    .AsEnumerable() // switches to LINQ to Objects
    .Where(c => IsSpecialCustomer(c));
```

Basic queries : enkele opm

- Filteren: **Where**
  - via navigational props => JOIN!!

```
Console.WriteLine("\n---Filtering the brewers:  brewers from Leuven--");
_brewers = context.Brewers
   .Where(b => b.Location.Name == "Leuven")
   .OrderBy(b => b.Name)
   .ToList();
```

```
SELECT [b].[BrewerId], [b].[ContactEmail], [b].[DateEstablished], [b].[Description], [b].[LocationPostalCode], [b]
.[BrewerName], [b].[Street], [b].[Turnover], [b.Location].[PostalCode], [b.Location].[Name]
FROM [Brewers] AS [b]
LEFT JOIN [Locations] AS [b.Location] ON [b].[LocationPostalCode] = [b.Location].[PostalCode]
ORDER BY [b].[BrewerName], [b].[LocationPostalCode]
```

Distinct() : geeft enkel de verschillende terug

Basic queries : enkele opm

- Voorbeeld filteren op * associatie via **Any** (subQuery)
  - Brouwers met een bier waarvan de naam met een B begint. Gebruiken associatie *

```
_brewers = context.Brewers
   .Where(br => br.Beers.Any(b => b.Name.ToUpper().StartsWith("B")))
   .ToList();
```

```
SELECT [br].[BrewerId], [br].[ContactEmail], [br].[DateEstablished], [br].[Description], [br].[LocationPostalCode]
, [br].[BrewerName], [br].[Street], [br].[Turnover]
FROM [Brewers] AS [br]
WHERE EXISTS (
    SELECT 1
    FROM [Beers] AS [b]
    WHERE UPPER([b].[Name]) LIKE N'B' + N'%' AND ([br].[BrewerId] = [b].[BrewerId]))
```

Heel wat methodes beschikbaar:
- Contains
- Count
- Except
- Intersect
-…

Loading Related Data: opm

- **Projection: Select**: Opvragen specifieke properties van brouwers
  - Kan performantie verbeteren

```
Console.WriteLine("\n---All brewers from Leuven, print the name and the number of beers---");
_brewers = context.Brewers
   .Include(b => b.Beers)
   .Where(b => b.Location.Name == "Leuven")
   .ToList();
```

```
Console.WriteLine("\n---All brewers from Leuven, print the name and the number of beers - Use projections---");
var brewers2 = context.Brewers
   .Where(b => b.Location.Name == "Leuven")
   .Select(b => new { Name = b.Name, NumberOfBeers = b.Beers.Count })
   .ToList();
```

```
SELECT [b].[BrewerId], [b].[ContactEmail], [b].[DateEstablished], [b].[Description], [b].[LocationPostalCode], [b]
.[BrewerName], [b].[Street], [b].[Turnover], [b.Location].[PostalCode], [b.Location].[Name], (
    SELECT COUNT(*)
    FROM [Beers] AS [b1]
    WHERE [b].[BrewerId] = [b1].[BrewerId]
)
FROM [Brewers] AS [b]
LEFT JOIN [Locations] AS [b.Location] ON [b].[LocationPostalCode] = [b.Location].[PostalCode]
ORDER BY [b].[LocationPostalCode]
```

Loading Related Data: opm

- Overerving
  - OfType

```
_brewer = context.Brewers
   .Include(b => b.Courses)
   .SingleOrDefault(b => b.Name == "Bavik");
var courses = _brewer.Courses.OfType<OnlineCourse>().ToList();
```

Saving data

HoGent

DbContext en updates

- DbContext verzorgt de object Tracking
  - Bij opvragen van objecten (na een select query) worden deze in de Cache (Identity Map) geplaatst.
  - De DbContext houdt in de cache voor elke entiteit 2 objecten bij
    - Het object (entiteit) zelf
    - De ObjectStateEntry : nodig voor change tracking
      - Originele waarde van object
      - EntityState : unchanged, added, updated, deleted
  - SaveChanges : persisteert wijzigingen naar database
    - Alle entiteiten in cache worden overlopen. Als EntityState verschilt van unchanged wordt insert, update of delete instructie gegenereerd. Enkel de gewijzigde properties worden gepersisteerd.
    - = Unit of Work (alle wijzigingen tegelijk) en TransactieBeheer (alle wijzigingen lukken of worden gerollbackt)

Saving Data

- Meer op https://docs.microsoft.com/en-us/ef/core/saving/
  - Basic Save
  - Related Data
  - Cascade Delete
- Lees de documentatie en pas dit toe op de voorbeelden in program.cs. Plaats uit commentaar:

```
#region "DbContext en updates"
SavingData(context);
#endregion
```

- Zorg ervoor dat de database nu telkens eerst verwijderd wordt en opnieuw gecreëerd wordt

```
using (ApplicationDbContext context = new ApplicationDbContext())
{
    context.Database.EnsureDeleted();
    context.Database.EnsureCreated();
    new BeerhallDataInitializer(context).InitializeData();
    Console.WriteLine("Database created");
```

Saving data: opm

- Toevoegen van een object met een identity kolom
  - 2 queries naar de database
    - Insert
    - Select : opvragen van de gegenereerde sleutel en aanpassen van het object

```
Console.WriteLine("\n---Add: Create Brewer Gentse Gruut, Rembert Dodoensdreef, 9000 Gent ---");
Brewer gruut = new Brewer("Gentse Gruut")
{
  Street = "Rembert Dodoensdreef",
  Location = context.Locations.Single(g => g.Name == "Gent")
};
context.Brewers.Add(gruut);
context.SaveChanges();
```

```
INSERT INTO [Brewer] ([ContactEmail], [DateEstablished], [Description], [BrewerName], [PostalCode], [Street], [Turnover])
VALUES (@p0, @p1, @p2, @p3, @p4, @p5, @p6);
SELECT [BrewerId]
FROM [Brewer]
WHERE @@ROWCOUNT = 1 AND [BrewerId] = scope_identity();
```

Saving data: opm

- DbContext verzorgt ook de associaties
  - Opgelet met deletes via navigational properties

```
_brewer = context.Brewers.Single(b => b.Name == "Bavik");
Beer beer = _brewer.Beers.First();
_brewer.DeleteBeer(beer);
context.SaveChanges();
```

If a cascade delete is configured, the child/dependent entity will be deleted from the database, see Cascade Delete for more information. If no cascade delete is configured, the foreign key column in the database will be set to null (if the column does not accept nulls, an exception will be thrown).

  - Als cascade delete = no action. Beer blijft bestaan maar is niet langer gekoppeld aan brouwer. Als beer ook verwijdert moet worden:

```
_brewer = context.Brewers.Single(b => b.Name == "Bavik");
Beer beer = _brewer.Beers.FirstOrDefault();
_brewer.DeleteBeer(beer);
context.Beers.Remove(beer);
context.SaveChanges();
```

Appendix

HoGent

8. Extra's : Interessante links

- De documentatie: https://docs.microsoft.com/en-us/ef/core/
- Introductie tot EF Core: http://www.learnentityframeworkcore.com/

8. Extra's: EF Migrations

- Installeer ook de Microsoft.EntityFrameworkCore.Tools als je EF Core commands in powershell gebruikt en build het project (Zie appendix voor gebruik)

```
The following Entity Framework cmdlets are included.

    Cmdlet                      Description
    --------------------------  ---------------------------------------------------
    Add-Migration               Adds a new migration.

    Remove-Migration            Removes the last migration.

    Scaffold-DbContext          Scaffolds a DbContext and entity type classes for a specified database.

    Script-Migration            Generates a SQL script from migrations.

    Update-Database             Updates the database to a specified migration.

    Use-DbContext               Sets the default DbContext to use.
```

- Meer op https://docs.microsoft.com/en-us/ef/core/miscellaneous/cli/powershell

8. Extra's: EF Migrations

- Bouwen van domein model adhv de **Migrations**
  - Pas het domein model aan: creëer een nieuwe domeinklasse of wijzig een bestaande klasse, voeg associaties toe, …
  - Maak gebruik van **EF Migrations** om de database aan te passen
    1. Genereer schema: add-migration
    2. Update de database: update-database
  - Bekijk de gegenereerde tabellen
  - Pas, indien nodig, de mapping aan
  - Maak gebruik van EF Migrations om de database verder aan te passen
  - Vul de database met sample data
  - Commit

8. Extra's: EF Migrations

- EF Migrations
  - Migrations is een manier om aanpassingen in het domein model ook door te voeren in de database, zonder de database eerst te verwijderen.

  - add-migration: scaffolds een nieuwe migratie en maakt hier een klasse voor aan met code om de database aan te passen.
  - update-database: voert de aanpassingen door in de database.

8. Extra's: EF Migrations

STAP 1

Maak domein model aan

```
namespace BeerhallEF.Models
{
    public class Brewer
    {
        #region Properties

        public int BrewerId { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string ContactEmail { get; set; }
        public DateTime? DateEstablished { get; set; }
        public string Street { get; set; }
        public int? Turnover { get; set; }

        #endregion
    }
}
```

```
namespace BeerhallEF.Data
{
    public class ApplicationDbContext : DbContext
    {
        public DbSet<Brewer> Brewers { get; set; }
```

STAP 2

PM>add-migration CreateTableBrewers

```
namespace BeerhallEF.Migrations
{
    public partial class CreateTableBrewers : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Brewers",
                columns: table => new
                {
                    BrewerId = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy",
                        SqlServerValueGenerationStrategy.IdentityColumn),
                    ContactEmail = table.Column<string>(nullable: true),
                    DateEstablished = table.Column<DateTime>(nullable: true),
                    Description = table.Column<string>(nullable: true),
                    Name = table.Column<string>(nullable: true),
                    Street = table.Column<string>(nullable: true),
                    Turnover = table.Column<int>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Brewers", x => x.BrewerId);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Brewers");
        }
    }
}
```

STAP 3

PM>update-database

Brewers

Column Name	Data Type	Allow Nulls
BrewerId	int	☐
ContactEmail	nvarchar(MAX)	☑
DateEstablished	datetime2(7)	☑
Description	nvarchar(MAX)	☑
Name	nvarchar(MAX)	☑
Street	nvarchar(MAX)	☑
Turnover	int	☑
		☐

8. Extra's: EF Migrations

- Data Seeding
  - https://docs.microsoft.com/en-us/ef/core/modeling/data-seeding

8. Extra's : Shadow properties

- Properties die geen deel uitmaken van het domein model. De waarde en state van deze properties wordt enkel bijgehouden in de Change Tracker
  - modelBuilder.Entity<Brewer>()
    .Property<DateTime>("LastUpdated");

  - Meer op https://docs.microsoft.com/en-us/ef/core/modeling/shadow-properties

8. Extra's : Lazy loading

- De gerelateerde data wordt automatisch geladen wanneer een navigational property gebruikt wordt
  - Meer info op https://docs.microsoft.com/en-us/ef/core/querying/related-data, zie Lazy loading

8. Extra's : Value converters

- Je zou bvb de data kunnen encrypteren alvorens het wordt opgeslaan in de db
  - Meer op https://docs.microsoft.com/en-us/ef/core/modeling/value-conversions

8. Extra's : Async programmeren

- Van een aantal methods bestaat ook een asynchrone versie (bvb SaveChangesAsync). Een uitgebreide uitleg over asynchroon programmeren (async, await) kan je vinden op onderstaande link
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/