



Databanken

ADO.NET Entity Framework



Entity Framework

MSDN: [Get Started with EF](#)

Tutorial: [Entity Framework Tutorial](#)

ORM

Object-Relational Mapper

ORM

- Doel
 - Relationeel model versus OO-model
 - Mappen van het DB-schema naar objecten
 - Genereert elk CRUD-statement
 - Win tijd tijdens coderen en onderhoud
- Minder werk en verlaagde foutmarge m.b.t.:
 - Interactie met data storage
 - Vertaalslag resultsets uit DB naar de eigenlijke objecten
- Meest gekende en gebruikte in .NET
 - Microsoft ADO.NET Entity Framework
 - NHibernate

Klassieke ADO.NET

- Databases aanspreken met “klassieke ADO.NET” doe je als volgt:
 1. Maak connectieobject aan
 2. Creëer commando object
 - i. SQL-Statement
 - ii. Parameters
 3. Open de connectie
 4. Voer commando uit
 - i. vertaal eventueel de resultset naar objecten met behulp van een DataReader
 5. Sluit de connectie

Entity Framework

- Connectie, commando's (en zijn parameters, ...) worden allemaal beheerd door het Entity Framework
- `System.Data.Entity.DbContext` is het aanspreekpunt vanuit je applicatie

Installatie

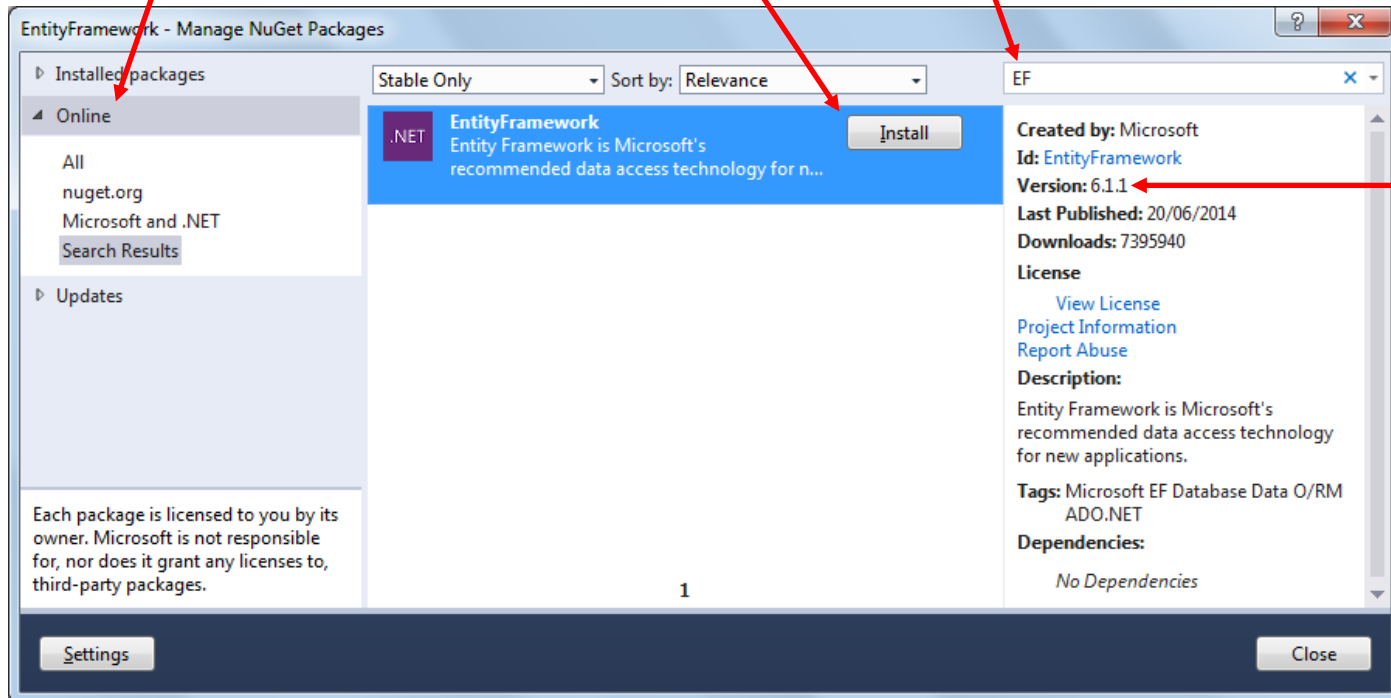
NuGet Package

- NuGet (<http://www.nuget.org/>)
 - Tool/AddIn binnen Visual Studio
 - Packages via internet/intranet toevoegen aan je project
 - Package kan meerdere references bevatten
 - Bevat standaard de laatste nieuwe versie
 - Vorige versies zijn ook mogelijk via PM Console
 - Solution Explorer
 - Project > RMK > Manage NuGet Packages...

NuGet Package

– Manage NuGet Packages

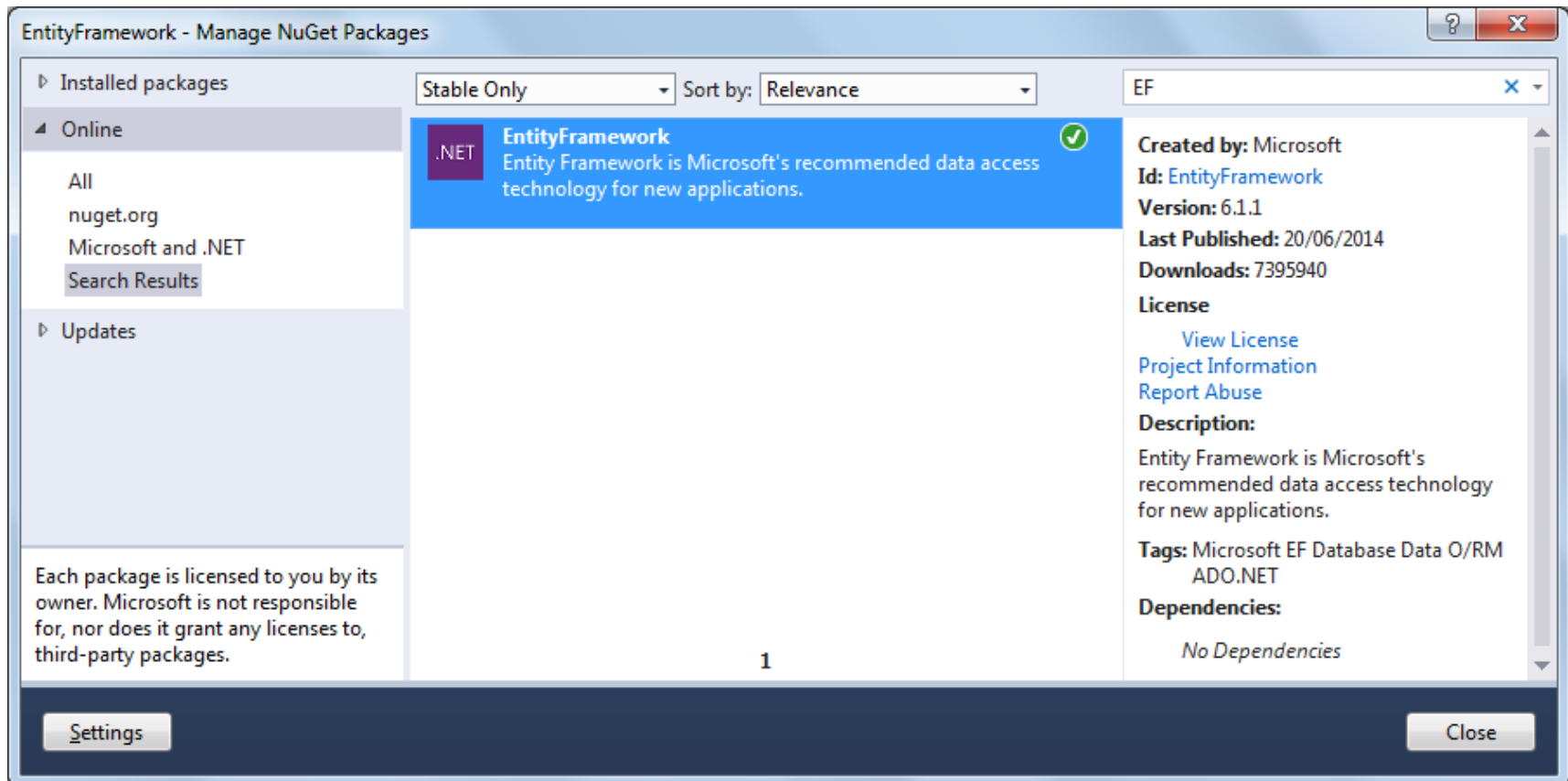
- Kies 'Online'
- Type in de search 'EF'
- Klik op 'Install'



Merk op: huidige versie 6.1.1

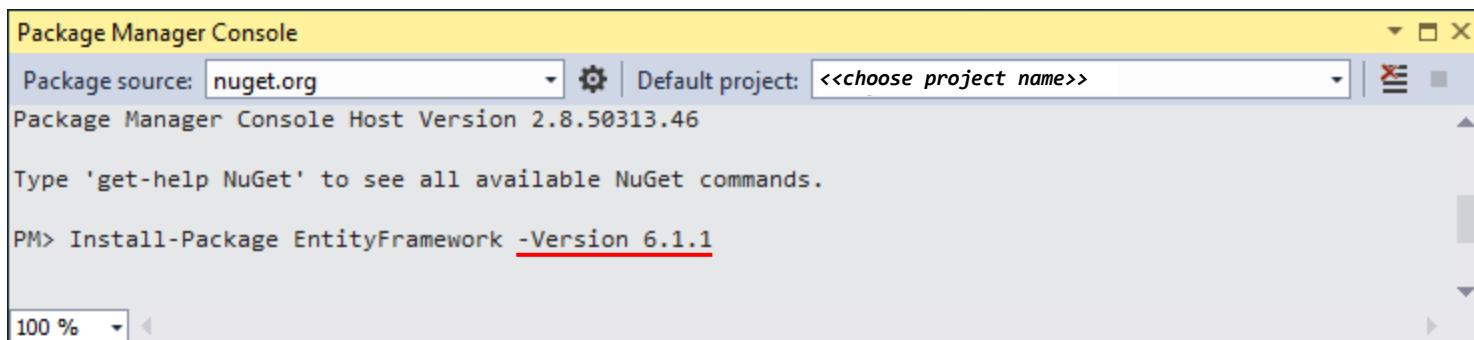
NuGet Package

- Na installatie:
 - Druk op Close



NuGet Package

- Package Manager Console
 - geeft mogelijk om versie mee te geven
- Tools > NuGet Package Manager
 - > Package Manager Console



NuGet Package

- Wat is er aangemaakt?
 - Reference naar
 - EntityFramework.dll
 - EntityFramework.SqlServer.dll
 - System.ComponentModel.DataAnnotations.dll (in GAC)
 - System.Data.DataSetExtensions.dll (in GAC)
 - App.config/Web.config
 - Nodige voorbereidingen om bijkomende configuratie te kunnen doen en om met SQL Server als achterliggende databank te kunnen werken
 - package.config
 - Overzicht van welke packages er voor het betreffende project nodig zijn (om gemakkelijk te kunnen herstellen)
 - File System
 - {SolutionPath}\packages\EntityFramework.6.1.1
 - Bevat dll's van het geïnstalleerde Entity Framework

Application config-file

- App.config / Web.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  ...
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="EntityFramework"
      type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
        Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
      requirePermission="false" />
  </configSections>
  ...
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="v11.0" />
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient"
        type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
  </entityFramework>
  ...
</configuration>
```



Application config-file

- defaultConnectionFactory
 - zorgt ervoor dat als er geen connectionstring wordt gedefinieert voor EF Code First, er een standaard database wordt aangemaakt
 - System.Data.Entity.Infrastructure.LocalDbConnectionFactory
 - maakt gebruik van data source '(localdb)'
 - naam van de DbContext-klasse wordt gebruikt als naam voor de databank
- provider
 - geeft aan welke data provider door EF gebruikt moet worden
 - invariantName: verwijst naar de 'ADO.NET Data Provider' die in een connectionstring aangegeven wordt in 'providerName'
 - type: assembly qualified name van het provider-type dat door EF gebruikt moet worden
 - dit is nodig indien de voorziene 'ADO.NET Data Provider' niet compatibel is met EF!

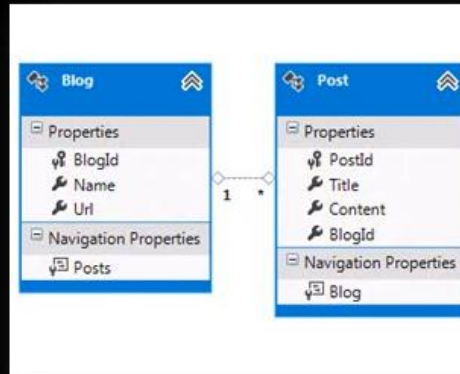
package.config

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  ...
  <package id="EntityFramework" version="6.1.1" targetFramework="net45" />
  ...
</packages>
```



EF Code First

Entity Framework Workflows



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



Model First

- Create model in designer
- Database created from model
- Classes auto-generated from model

Code First (New Database)

- Define classes & mapping in code
- Database created from model
- Use Migrations to evolve database



Database First

- Reverse engineer model in designer
- Classes auto-generated from model

Code First (Existing Database)

- Define classes & mapping in code
- Reverse engineer tools available

Code First (New Database)

1. Definieer models
2. Definieer een database context
 - voorzie een initializer (optioneel)

⇒ DB-schema wordt hier van afgeleid
en de databank wordt gecreëerd!!

Models

Models

- Beschrijven datastructuren die staan voor 'entiteiten' binnen een applicatie
- Bij het Entity Framework (EF) zijn dit objecten die direct gerelateerd zijn aan de opslag van data voor de applicatie

Models

- O/R Mapping restricties
 - Enkel **classes**, géén structs!
 - Enkel **public properties** worden in acht genomen voor het afleiden van het DB-schema
 - Properties, onderscheid tussen:
 - **scalar properties**
 - bevatten effectieve waarden die bewaard moet worden (datatype: primitive- en enum-types)
 - **navigation properties**
 - beschrijven relaties met andere models (datatype: andere model-types)
 - Elk model moet een '**unique identifier**' hebben!
- zie onderdeel [Object-Relational Mapping](#)

Voorbeeld

```
...
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace SC.BL.Domain
{
    [Table("tblTickets")]
    public class Ticket
    {
        [Key]
        public int TicketNumber { get; set; }
        public int AccountId { get; set; }
        [Required]
        [MaxLength(100, ErrorMessage = "Er zijn maximaal 100 tekens toegestaan")]
        public string Text { get; set; }
        public DateTime DateOpened { get; set; }
        [Index]
        public TicketState State { get; set; }

        public ICollection<TicketResponse> Responses { get; set; } // navigation-property
    }

    public class HardwareTicket : Ticket
    {
        public string DeviceName { get; set; }
    }
}
```



Voorbeeld



```
[Table("tblTicketResponses")]
public class TicketResponse
{
    public int Id { get; set; }
    [Required]
    public string Text { get; set; }
    public DateTime Date { get; set; }
    public bool IsClientResponse { get; set; }

    [Required]
    public Ticket Ticket { get; set; } // navigation-property
}
```

DbContext

DbContext

- `System.Data.Entity.DbContext`
- Base-class voor de entity container van de applicatie die dient als aanspreekpunt naar de databank
- **SaveChanges**-methode
 - persisteert de wijzigingen gemaakt aan de entiteiten in de context naar de databank

Container klasse

- Is een klasse die overerft van `System.Data.Entity.DbContext`
- Inhoud:
 - constructor: indien een connectionstring gespecificeerd wordt uit de config file
 - properties: voor elk model een property van het type `System.Data.Entity.DbSet<'modelType'>`
 - OnModelCreating-methode: om het standaard gedrag voor de creatie het DB-schema te beïnvloeden (zie [Fluent API](#))

ConnectionString

- Bepalen van connectionstring
 - Geef naam connectionstring door aan de constructor van de base class

```
public class TicketEFDbContext : System.Data.Entity.DbContext
{
    ...
    public TicketEFDbContext() : base('SupportCenterDB_EFCodeFirst')
    { }
    ...
}
```

- In App.config file, voeg connectionstring toe met de opgegeven naam

```
<connectionStrings>
  <add name='SupportCenterDB_EFCodeFirst'
        connectionString="Data Source=.\SQLSERVER2012;
                           Initial Catalog=MyDB_EFCodeFirst;
                           Integrated Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Voorbeeld

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

using EF.CodeFirst.Models;

namespace EF.CodeFirst.DAL
{
    public class TicketEFDbContext : DbContext
    {
        public TicketEFDbContext() : base("name=SupportCenterDB_EFCodeFirst")
        {
        }

        public DbSet<Ticket> Tickets { get; set; }
        public DbSet<TicketResponse> TicketResponses { get; set; }
    }
}
```

```
<connectionStrings>
    <add name="SupportCenterDB_EFCodeFirst"
        connectionString="Data Source=.\SQLSERVER2012;
            Initial Catalog=MyDB_EFCodeFirst;
            Integrated Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

DbInitializer

- Zelf impact hebben op welke strategie gebruikt wordt
 - Wanneer moet de DB aangepast (drop + create) worden?

DbInitializers - Built-In

- Namespace: System.Data.Entity
- CreateDatabaseIfNotExists
 - De databank wordt enkel alleen aangemaakt indien deze nog niet bestaat
 - Indien door omstandigheden de models en DB-schema niet meer overeenkomen kunnen er exceptions optreden
- DropCreateDatabaseIfModelChanges
 - De databank wordt verwijderd en terug aangemaakt indien de container klasse of een model gewijzigd is
- DropCreateDatabaseAlways
 - De databank wordt elke keer dat er een initialisatie plaatsvindt opnieuw verwijderd terug aangemaakt

DbInitializer - Custom

- Mogelijkheid om 'data' te voorzien in de databank bij initialisatie (default-, dummy-data)
 - Seed-methode
- Op basis van een built-in initializer
 - Erft over van de bestaande DbInitializer

DbIniatilizer - Custom

```
public class TicketEFDbInitializer
    : DropCreateDatabaseAlways<TicketEFDbContext>
{
    protected override void Seed(TicketEFDbContext context)
    {
        Ticket t1 = new Ticket()
        {
            AccountId = 1,
            Text = "Ik kan mij niet aanmelden op de webmail",
            DateOpened = new DateTime(2012, 9, 9, 13, 5, 59),
            State = TicketState.Closed
        };
        context.Tickets.Add(t1);

        context.SaveChanges();
    }
}
```


DbInitializer - Instellen

- Database-klasse
 - stel de intitializer in, in de constructor van de container klasse

```
public class TicketEFDbContext : DbContext
{
    public TicketEFDbContext() : base("name=SupportCenterDB_EFCodeFirst")
    {
        System.Data.Entity.Database.SetInitializer<TicketEFDbContext>(new TicketEFDbInitializer());
    }

    ...
}
```

DbInitializer - Instellen

- Application config-file (\geq EF 4.1)

```
<appSettings>
  ...
  <add key="DatabaseInitializerForType EF.CodeFirst.DAL.TicketEFDbContext, EF.CodeFirst.DAL"
        value="EF.CodeFirst.DAL.TicketEFDbInitializer, EF.CodeFirst.DAL" />
</appSettings>
```

'appSettings'-section

> 'add'-element

- key: begint met 'DatabaseInitializerForType' gevolgt door een spatie en 'assembly qualified name' van de container klasse
- value: 'assembly qualified name' van de initializer klasse

DbInitializer - Instellen

- Application config-file (\geq EF 4.3)

```
<entityFramework>
  ...
  <contexts>
    <context type="EF.CodeFirst.DAL.TicketEFDbContext, EF.CodeFirst.DAL"
      disableDatabaseInitializer="false">
      <databaseInitializer type="EF.CodeFirst.TicketEFDbInitializer, EF.CodeFirst.DAL" />
    </context>
  </contexts>
</entityFramework>
```

'entityFramework'-section

> 'contexts'-section > 'context'-element

- type: assembly qualified name van de container klasse
- disableDatabaseInitializer: true / false

> 'databaseInitializer'-element

- type: assembly qualified name van de initializer klasse

DbInitializer - Instellen

- DbConfiguration (\geq EF 6)
 1. voorzie een configuratie klasse

```
class TicketEFDbConfiguration : DbConfiguration
{
    public TicketEFDbConfiguration()
    {
        ...
        this.SetDatabaseInitializer<TicketEFDbContext>(new TicketEFDbInitializer());
    }
}
```

2. pas de configuratie klasse toe op de container(/context) klasse

```
[DbConfigurationType(typeof(TicketEFDbConfiguration))]
class TicketEFDbContext : DbContext
{
    ...
}
```

Voorbeeld

```
...
using System.Data.Entity;
using EF.CodeFirst.Models;

namespace EF.CodeFirst.DAL
{
    public class TicketEFDbContext : DbContext
    {
        public TicketEFDbContext() : base("name=SupportCenterDB_EFCodeFirst")
        {
            Database.SetInitializer<TicketEFDbContext>(new TicketEFDbInitializer());
        }

        public DbSet<Ticket> Tickets { get; set; }
        public DbSet<TicketResponse> TicketResponses { get; set; }
    }
}
```

Voorbeeld

```
...
using System.Data.Entity;
using EF.CodeFirst.Models;

namespace EF.CodeFirst.DAL
{
    public class TicketEFDbInitializer : DropCreateDatabaseAlways<TicketEFDbContext>
    {
        protected override void Seed(TicketEFDbContext context)
        {
            Ticket t1 = new Ticket()
            {
                AccountId = 1,
                Text = "Ik kan mij niet aanmelden op de webmail",
                DateOpened = new DateTime(2012, 9, 9, 13, 5, 59),
                State = TicketState.Closed
            };
            context.Tickets.Add(t1);

            context.SaveChanges();
        }
    }
}
```



Working with data

Ophalen data

Syntax

`Container.DbSet[.Query].EntityType(s);`

- **Container** (aka context)
 - Opzetten connectie
- **DbSet**
 - Welke entiteiten(/tabel) wil je queryen?
- **Query** (optioneel)
 - Welke filter / where-clause wil je toepassen?
- **EntityType(s)**
 - Verwacht je één of meerdere records terug?

Query (optioneel)

- LINQ-extension methoden (chainable):
 - Where
 - OrderBy
 - Except
 - Select
 - SelectMany
 - Take
 - TakeWhile
 - (Distinct)
 - ...
 - return IQueryable<ModelType>
- IQueryable: bouwt een sql-query op, maar deze wordt nog niet uitgevoerd op de databank!

Query via delegate


```
int ticketId;

public IEnumerable<TicketResponse> ReadTicketResponsesOfTicket(int id)
{
    ticketId = id;

    SupportCenterEFDbContext ctx = new SupportCenterEFDbContext();

    return ctx.TicketResponses.Where(DoesTicketResponseMatch).AsEnumerable();
}

public bool DoesTicketResponseMatch(TicketResponse responseOutOfList)
{
    return (responseOutOfList.TicketNumber == ticketId);
}
```



Query via anonymous method

```
public IEnumerable<TicketResponse> ReadTicketResponsesOfTicket(int id)
{
    SupportCenterEFDbContext ctx = new SupportCenterEFDbContext();

    return ctx.TicketResponses
        .Where(
            delegate(TicketResponse responseOutOfList)
            { return responseOutOfList.TicketNumber == id; }
        )
        .AsEnumerable();
}
```

Query via lambda-expression

```
public IEnumerable<TicketResponse> ReadTicketResponsesOfTicket(int id)
{
    SupportCenterEFDbContext ctx = new SupportCenterEFDbContext();

    return ctx.TicketResponses
        .Where(responseOutOfList => responseOutOfList.TicketNumber == id)
        .AsEnumerable();
}
```

Query via LINQ-expression

```
public IEnumerable<TicketResponse> ReadTicketResponsesOfTicket(int id)
{
    SupportCenterEFDbContext ctx = new SupportCenterEFDbContext();

    return ctx.TicketResponses
        .Where(
            // voor elke record (met variabelenaam 'response') in TicketResponses
            from response in ctx.TicketResponses
            // waarvoor de response.TicketId gelijk is aan de filter
            where response.TicketNumber == id
            // geef het record terug
            select response;
        )
        .AsEnumerable();
}
```

EntityType(s)

- DbSet-methode
 - Find
- LINQ-extension methoden:
 - 1 resultaat
 - Single / SingleOrDefault
 - First / FirstOrDefault
 - Last / LastOrDefault
 - ...
 - return ModelType
 - meerdere resultaten
 - AsEnumerable
 - ToList
 - return IEnumerable<ModelType>
- Zorgt ervoor dat de sql-query wordt uitgevoerd op de databank en het resultaat in het geheugen terecht komt!

Voorbeelden

```
TicketEFDbContext ctx = new TicketEFDbContext();

// Load all tickets
IEnumerable<Ticket> tickets = ctx.Tickets.AsEnumerable();

// Load ticket with a given 'ticketNumber' (unique identifier)
Ticket ticket = ctx.Tickets.Find(ticketNumber);

// Load ticket for a given 'accountId'
Ticket ticket = ctx.Tickets.Single(t => t.AccountId == accountId);

// Load ticket with state 'Open'
IEnumerable<Ticket> tickets = ctx.Tickets
    .Where(t => t.State == TicketState.Open)
    .AsEnumerable();

// Load responses of ticket by a given 'ticketNumber'
IEnumerable<TicketResponse> responses = ctx.TicketResponses
    .Where(r => r.TicketNumber == ticketNumber)
    .AsEnumerable();
```


Geassocieerde data

- Tijdens het ophalen van data kan geassocieerde data (navigation-properties) mee opgehaald worden op 2 manieren
 - Lazy loading (standaard)
 - Eager loading
- Voor een gekend model/entity in de container/context, zonder geassocieerde data, kan geassocieerde data nadien nog opgehaald worden
 - Explicit loading

Geassocieerde data - Lazy loading

- De geassocieerde data wordt pas opgehaald uit de databank als de navigation-property van een entiteit wordt aangeroepen
- Voorwaarden
 - Lazy loading moet ingeschakeld zijn op de container klasse (standaard)
 - Uitschakelen:
`Container.Configuration.LazyLoadingEnabled = false;`
 - Navigation-property moet 'virtual' zijn

Geassocieerde data - Lazy loading

```
public class Ticket
{
    ...
    public virtual ICollection<TicketResponse> Responses { get; set; }
}

public class TicketEFDbContext : System.Data.Entity.DbContext
{
    ...
    public System.Data.Entity.DbSet<Ticket> Tickets { get; set; }
}
```

```
TicketEFDbContext ctx = new TicketEFDbContext();

// Load ticket with state 'Open'
List<Ticket> tickets = ctx.Tickets
    .Where(t => t.State == TicketState.Open)
    .ToList();

// Load associated responses of third and fifth ticket
List<TicketResponse> responsesOfThirdTicket = tickets[2].Responses;
List<TicketResponse> responsesOfFifthTicket = tickets[4].Responses;
```

Geassocieerde data - Eager loading

- Op het moment dat je entiteiten queried geef je aan dat bepaalde specifieke geassocieerde entiteiten mee moeten opgehaald worden
- **DbSet.Include**-methode

Geassocieerde data - Eager loading

```
public class Ticket
{
    ...
    public virtual ICollection<TicketResponse> Responses { get; set; }
}

public class TicketEFDbContext : System.Data.Entity.DbContext
{
    ...
    public System.Data.Entity.DbSet<Ticket> Tickets { get; set; }
}
```

niet noodzakelijk bij Eager loading!

```
TicketEFDbContext ctx = new TicketEFDbContext();

// Load ticket with state 'Open'
List<Ticket> tickets = ctx.Tickets.Include(t => t.Responses)
    .Where(t => t.State == TicketState.Open)
    .ToList();
```

Geassocieerde data - Explicit loading

- Op het moment dat je entiteiten queried geef je aan dat bepaalde specifieke geassocieerde entiteiten mee moeten opgehaald worden
- **Container.Entry.Reference.Load()**
 - single-navigation-property
- **Container.Entry.Collection.Load()**
 - collection-navigation-property

Geassocieerde data - Explicit loading

```
public class Ticket
{
    ...
    public ICollection<TicketResponse> Responses { get; set; } // mag 'virtual' zijn voor Lazy-loading
}

public class TicketResponse
{
    ...
    public Ticket Ticket { get; set; } // mag 'virtual' zijn voor Lazy-loading
}

public class TicketEFDbContext : System.Data.Entity.DbContext
{
    ...
    public System.Data.Entity.DbSet<Ticket> Tickets { get; set; }
    public System.Data.Entity.DbSet<TicketResponse> TicketResponses { get; set; }
}
```

```
TicketEFDbContext ctx = new TicketEFDbContext();

// Load ticket of a TicketResponse-entity 'response'
Ticket ticket = ctx.Entry<TicketResponse>(response).Reference(r => r.Ticket).Load();

// Load ticketresponses of a Ticket-entity 'ticket'
List<TicketResponse> responses = ctx.Entry<Ticket>(ticket).Collection(t => t.Responses).Load();
```

Toevoegen data

Toevoegen van entities

Stappenplan

1. Nieuwe instantie maken van de entiteit
2. Nieuwe entiteit toevoegen
 - via **DbSet.Add**-methode
 - via navigation-property van bestaande entiteit in de container
3. Wijziging persisteren naar de databank
 - **Container.SaveChanges**

Toevoegen van entities

- OPGELET

- PK wordt niet ingevuld indien dit op de databank een auto-generated veld is
- Bij associaties moet ofwel de FK-property, ofwel de navigatie property een waarde hebben

Voorbeelden

```
TicketEFDbContext ctx = new TicketEFDbContext();

// Add new ticket (via DbSet.Add-methode)
Ticket ticketWithPrimaryKey = ctx.Tickets.Add(new Ticket() {
    AccountId = 1,
    Text = "New ticket",
    DateOpened = DateTime.Now,
    State = TicketState.Open
});
ctx.SaveChanges();

// Add new ticketresponse
// (1) via DbSet.Add-methode
//     Indien het ticket-object niet ingeladen is, maar wel de PK gekend is
//     en het geassocieerde model een FK-property heeft
TicketResponse responseWithPrimaryKey = ctx.TicketResponses.Add(new TicketResponse() {
    TicketNumber = 1, //FK-property: PK-key moet gekend zijn om als FK-key waarde te kunnen meegeven
    Text = "New response",
    Date = System.DateTime.Now,
    IsClientResponse = false
});
ctx.SaveChanges();

// (2) via navigation-property: 'ticket' is een gekende entiteit in de container!
Ticket ticket = ctx.Tickets.Find(1);
ticket.Responses.Add(new TicketResponse() {
    Text = "New response",
    Date = System.DateTime.Now,
    IsClientResponse = false
});
ctx.SaveChanges();
```

Wijzigen data

Wijzigen van entities

Stappenplan

1. Laad de entiteit die je wenst te wijzigen in de container
2. Wijzig de properties van de entiteit
 - Bewerk de properties zoals je altijd doet
3. Wijziging persisteren naar de databank
 - **Container**.SaveChanges()

Voorbeelden

```
TicketEFDbContext ctx = new TicketEFDbContext();  
  
// Update an ticket  
Ticket ticket = ctx.Tickets.Find(1);  
ticket.State = TicketState.Closed;  
ctx.SaveChanges(); // Persist changes to database!
```

Verwijder data

Verwijderen van entiteiten

Stappenplan

1. Laad de entiteit die je wenst te verwijderen in de container
2. Verwijder (remove) de entiteit van de DbSet
 - **DbSet.Remove**-methode
3. Wijziging persisteren naar de databank
 - **Container.SaveChanges()**

Voorbeelden

```
TicketEFDbContext ctx = new TicketEFDbContext();  
  
// Delete a ticket  
Ticket ticket = ctx.Tickets.Find(1);  
ctx.Tickets.Remove(ticket);  
ctx.SaveChanges(); // Persist changes to database!
```



Object-Relational Mapping

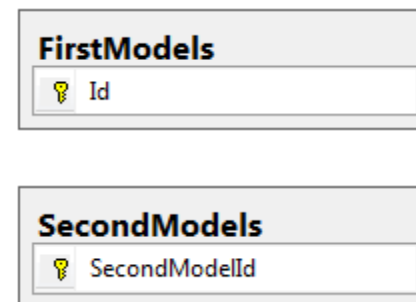
Models

- O/R Mapping restricties
 - Enkel **classes**, géén structs!
 - Enkel **public properties** worden in acht genomen voor het afleiden van het DB-schema
 - Properties, onderscheid tussen:
 - **scalar-properties**
 - bevatten effectieve waarden die bewaard moet worden (datatype: simple-, string-, DateTime- en enum-types)
 - **navigation-properties**
 - beschrijven relaties met andere models (datatype: andere model-types)
 - Elk model moet een '**unique identifier**' hebben!

Naming conventions

- Namen van klassen worden tabelnamen (in meervoud: suffix 's')
- Namen van scalar-properties worden kolomnamen(/attribuutnamen)
 - unique identifier: property met als naam 'Id' of '*ClassName*Id' wordt standaard als 'primary key' ingesteld

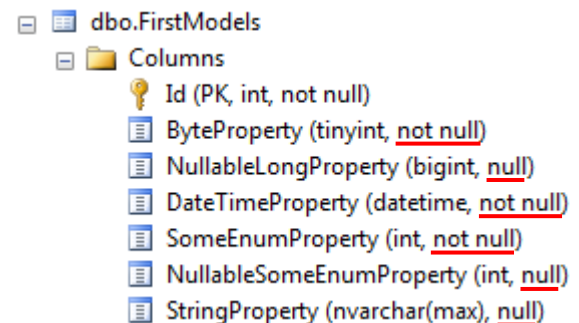
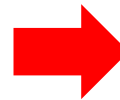
```
public class FirstModel {  
    public int Id { get; set; }  
}  
  
public class SecondModel {  
    public int SecondModelId { get; set; }  
}
```



Data-types (scalar-properties)

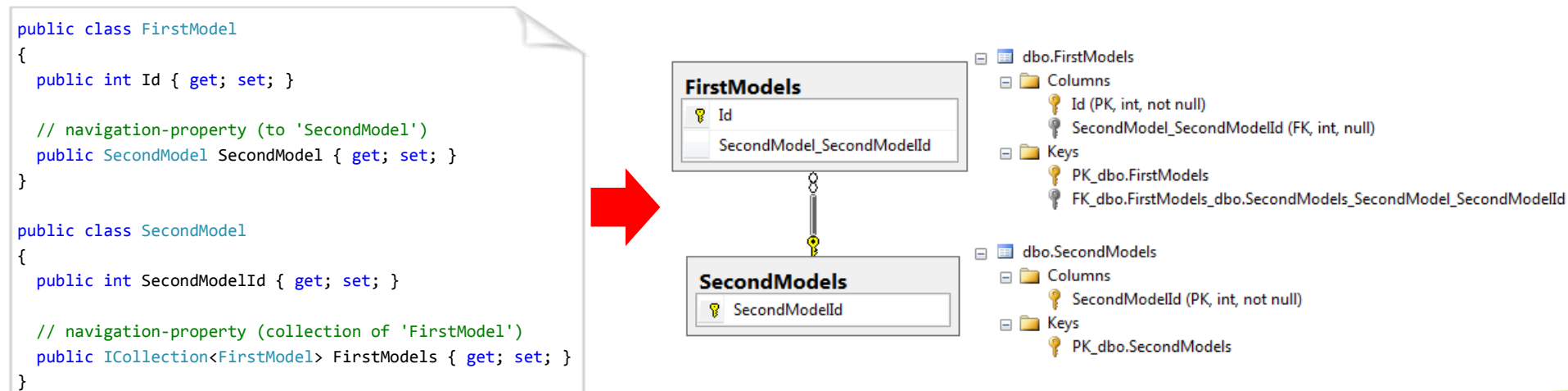
- Worden vertaald naar db-types
 - value-types: standaard 'not null' in db
 - nullable value-types: wel 'null' in db
 - reference-types: standaard 'null' in db

```
public class FirstModel {  
    public int Id { get; set; }  
  
    // value-types  
    public byte ByteProperty { get; set; }  
    public byte? NullableIntProperty { get; set; }  
    public DateTime DateTimeProperty { get; set; }  
    public SomeEnum SomeEnumProperty { get; set; }  
    public SomeEnum? NullableSomeEnumProperty { get; set; }  
  
    // reference-types  
    public string StringProperty { get; set; }  
}
```



Navigation-properties

- Properties waarvan het type een ander model is
 - Wordt vertaald naar een '**x-op-0..1**' relatie
- Properties waarvan het type een generic collection is, met als type een ander model
 - Wordt vertaald naar een '**x-op-n**' relatie



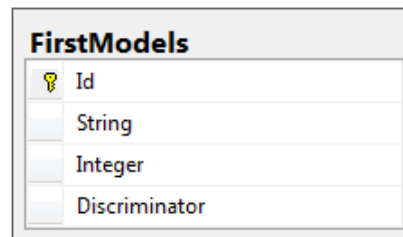
Inheritance

- Per inheritance-hiërarchie wordt er een tabel voorzien (TPH: table per hierarchy)
 - Alle properties van de verschillende derived-types worden mee opgenomen in dezelfde tabel
 - Een extra kolom 'Discriminator' wordt voorzien, waarin voor elke record wordt bijgehouden (tekstueel) met welk type die record overeenkomt

```
public class FirstModel
{
    public int Id { get; set; }
    public string String { get; set; }
}

public class DerivedModel : FirstModel
{
    public int Integer { get; set; }
}

public class MyEFDbContext : DbContext
{
    public DbSet<FirstModel> FirstModels { get; set; }
    public DbSet<DerivedModel> DerivedModels { get; set; }
}
```



dbo.FirstModels

Columns

- Id (PK, int, not null)
- String (nvarchar(max), null)
- Integer (int, null)
- Discriminator (nvarchar(128), not null)

	Id	String	Integer	Discriminator
1	1	firstmodel string	NULL	FirstModel
2	2	derivedmodel string	10	DerivedModel

O/R Mapping: overzicht

Model	DB-schema
Class	Table
- Name	- name
Scalar property	Attribute
- name	- name
» 'Id' of ' <i>ClassNameId</i> '	» primary key
- type	- db-type
Navigation property	Relationship
- name, type	- combined foreign key-name
Inheritance	TPH (table per hierarchy)

Data Annotations

Data Annotations

- 'Code First Data Annotations' (sinds EF 4.1) zijn attributes op models die de creatie van het DB-schema beïnvloeden
 - beperkte mogelijkheden
 - alternatief '[Fluent API: Type/Property Mapping](#)'
- Namespaces:
 - System.ComponentModel.DataAnnotations.Schema
 - System.ComponentModel.DataAnnotations

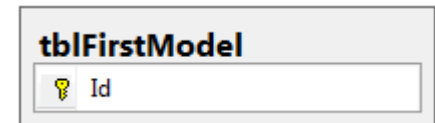
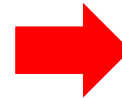
Data Annotations

- Type Mapping
 - TableAttribute
 - ComplexTypeAttribute
- Property Mapping
 - KeyAttribute
 - ForeignKeyAttribute
 - NotMappedAttribute
 - IndexAttribute
 - RequiredAttribute (-> Validation Framework)
 - MaxLengthAttribute (-> Validation Framework)

TableAttribute

- Kan gebruikt worden indien je een specifieke naam wenst te definiëren voor de tabel

```
[Table("tblFirstModel")]  
class FirstModel  
{  
    public int Id { get; set; }  
    ...  
}
```



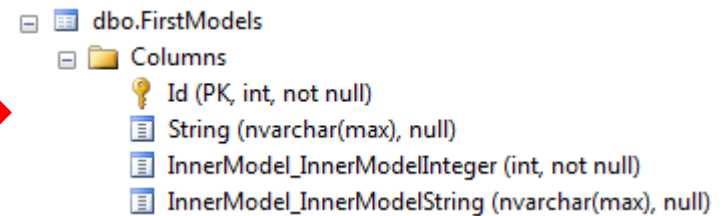
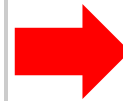
ComplexTypeAttribute

- Kan gebruikt worden bij puur 'encapsulatie' (als alternatief voor een 1-op-1-relatie)
 - geëncapsuleerde type: geen 'unique identifier' nodig

```
class FirstModel
{
    public int Id { get; set; }
    public string String { get; set; }

    public InnerModel InnerModel { get; set; }
}

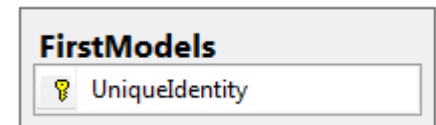
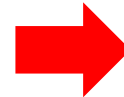
[ComplexType]
class InnerModel
{
    public int InnerModelInteger { get; set; }
    public string InnerModelString { get; set; }
}
```



KeyAttribute

- Kan gebruikt worden indien je wenst af te wijken van de standaard 'naming conventions' ('Id' of '*ClassName*Id') voor de naamgeving van de unique identifier van een model

```
class FirstModel
{
    [Key]
    public int UniqueIdentity { get; set; }
    ...
}
```



- Kan op meerdere properties tegelijk gebruikt worden die samen de 'unique identifier' van het model vormen

ForeignKeyAttribute

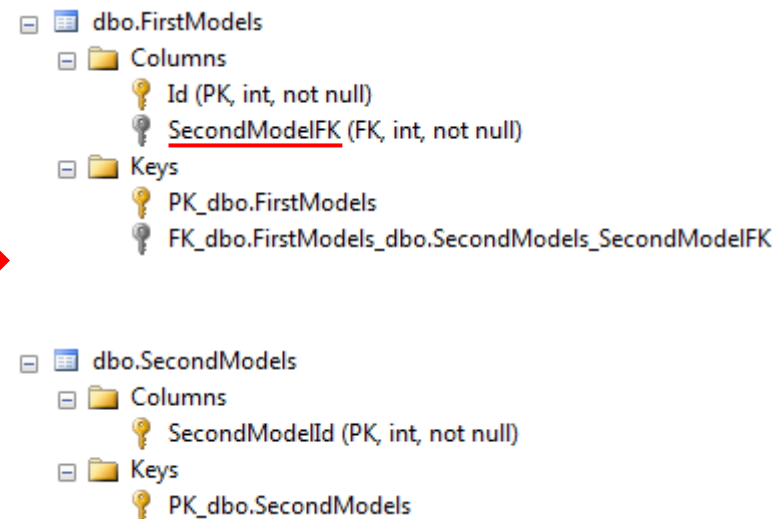
- Kan gebruikt worden indien je zelf een property voorzie waarin een foreign key-waarde moet bewaard worden, zodat je dit in je applicatie kan gebruiken
 - parameter: naam de navigation-property waarvoor deze property de FK-waarde moet bijhouden

```
class FirstModel
{
    public int Id { get; set; }
    [ForeignKey("SecondModelNavProp")]
    public int SecondModelFK { get; set; }

    // navigation-property (to 'SecondModel')
    public SecondModel SecondModelNavProp { get; set; }
}

class SecondModel
{
    public int SecondModelId { get; set; }

    // navigation-property (collection of 'FirstModel')
    public ICollection<FirstModel> FirstModels { get; set; }
}
```

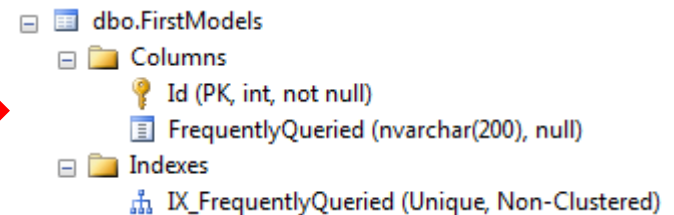


IndexAttribute

- Kan gebruikt worden bij properties waarvoor in de databank een 'index' moet voorzien worden
 - standaard naamgeving van de index is 'IX_<naam v/d property>'
- Mogelijke parameters
 - Naam
 - IsUnique (default 'false')
 - IsClustered (default 'false')
- OPGELET
Bij string-types moet er een maximum lengte voorzien worden, anders kan de index niet aangemaakt worden in de databank!

```
class FirstModel
{
    public int Id { get; set; }

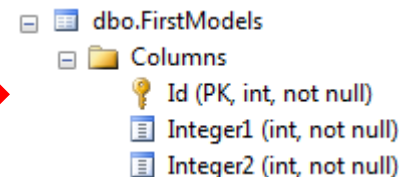
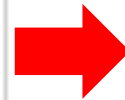
    [Index(IsUnique = true)]
    [MaxLength(200)]
    public string FrequentlyQueried { get; set; }
}
```



NotMappedAttribute

- Kan gebruikt worden bij properties waarvan de waarde niet in de databank moet bewaart worden
 - Er zal geen veld in de database aangemaakt worden voor die property
- OPGELET
Properties zonder public set-accessor worden sowieso niet voorzien in de databank!

```
class FirstModel
{
    public int Id { get; set; }
    public int Integer1 { get; set; }
    public int Integer1 { get; set; }
    public int Total {
        get { return Integer1 + Integer2; }
    }
    [NotMapped]
    public string String { get; set; }
}
```

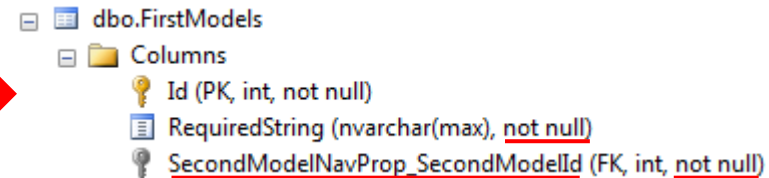


RequiredAttribute

- Zorgt ervoor dat voor die property een waarde verplicht is
 - scalar property: zorgt ervoor dat het veld/attribute/kolom 'not null' is in db
 - navigation-property: zorgt er voor dat het overeenkomstig foreign key-veld 'not null' is in db

```
class FirstModel
{
    public int Id { get; set; }
    [Required]
    public string RequiredString { get; set; }

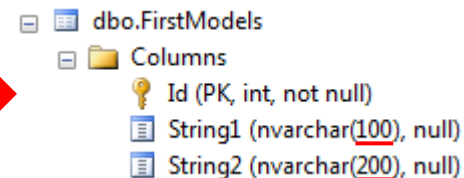
    [Required]
    public SecondModel SecondModelNavProp { get; set; }
}
```



MaxLengthAttribute

- Zorgt ervoor dat de lengte voor die property niet groter mag zijn dan de opgegeven waarde
 - enkel op type 'string' van toepassing, want array's worden niet ondersteund door EF
- Alternatief 'StringLengthAttribute'

```
class FirstModel
{
    public int Id { get; set; }
    [MaxLength(100)]
    public string String1 { get; set; }
    [StringLength(200)]
    public string String2 { get; set; }
}
```



Voorbeeld

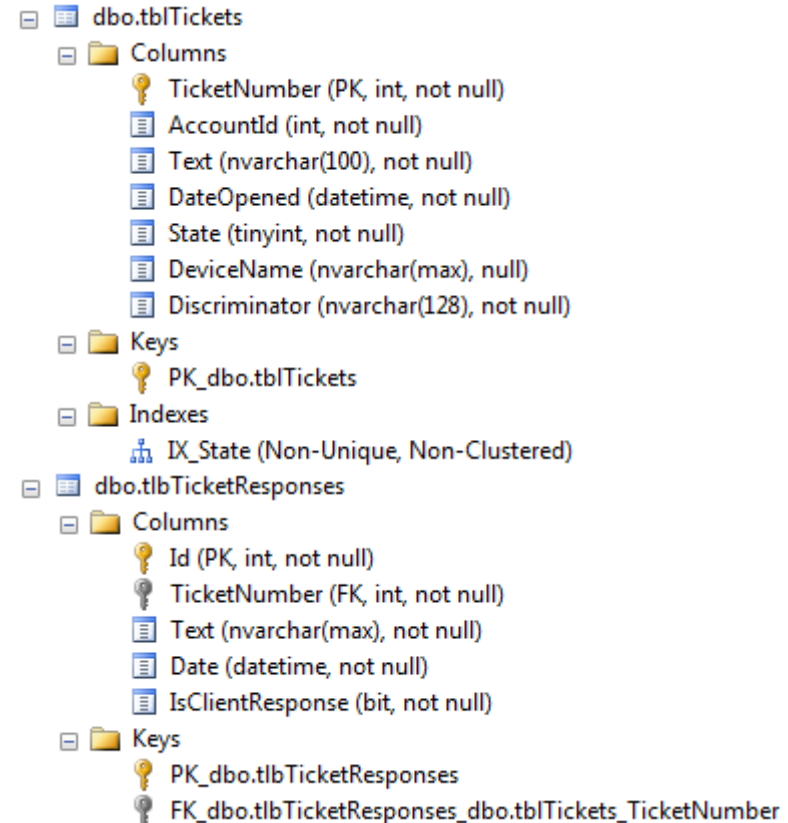
```
[Table("tblTickets")]
public class Ticket
{
    [Key]
    public int TicketNumber { get; set; }
    public int AccountId { get; set; }
    [Required]
    [MaxLength(100)]
    public string Text { get; set; }
    public DateTime DateOpened { get; set; }
    [Index]
    public TicketState State { get; set; }

    public ICollection<TicketResponse> Responses { get; set; }
}

public class HardwareTicket : Ticket
{
    public string DeviceName { get; set; }
}

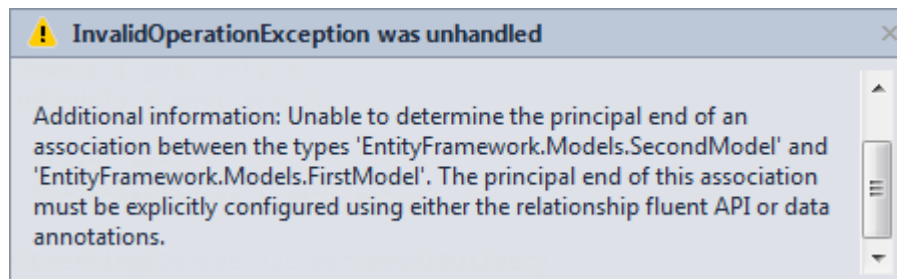
[Table("tblTicketResponses")]
public class TicketResponse
{
    public int Id { get; set; }
    [ForeignKey("Ticket")]
    public int TicketNumber { get; set; }
    [Required]
    public string Text { get; set; }
    public DateTime Date { get; set; }
    public bool IsClientResponse { get; set; }

    public Ticket Ticket { get; set; }
}
```



Relationships

- Voorzie altijd beide kanten van een relatie!
- 0..1-op-0..1 EN 1-op-1 zijn **niet mogelijk** zonder bijkomende configuratie (zie [Fluent API: Relational Mapping](#))



Relationships

- 1-op-1
kan ook vervangen worden door een
complexttype-configuratie
 - Geëncapsuleerd-model
 - ComplexTypeAttribute
 - geen 'unique identifier'

Relationships

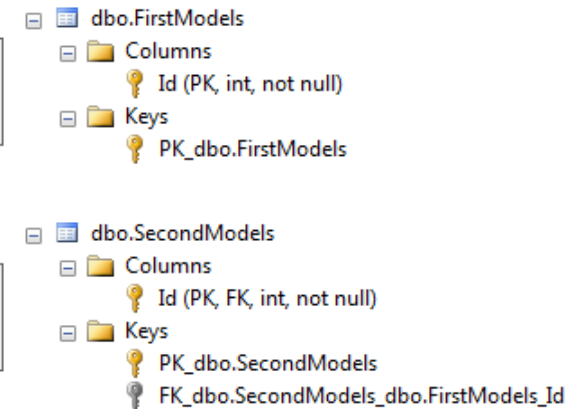
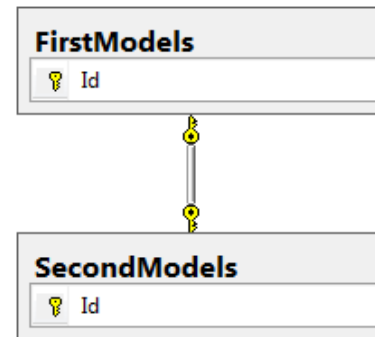
• 1-op-0..1

```
public class FirstModel
{
    public int Id { get; set; }

    public SecondModel SecondModel { get; set; }
}

public class SecondModel
{
    public int Id { get; set; }

    [Required]
    public FirstModel FirstModel { get; set; }
}
```



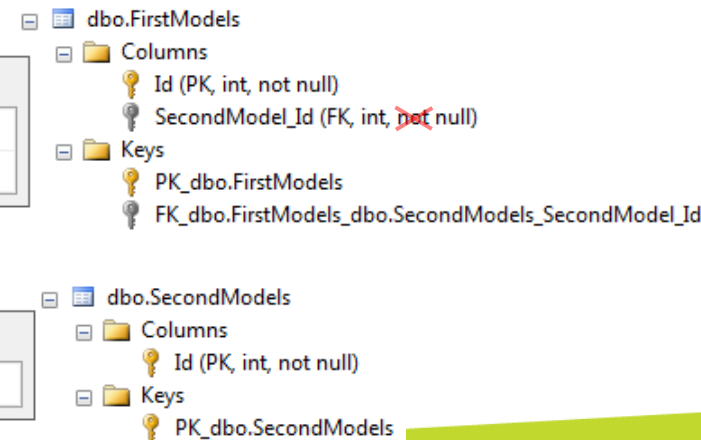
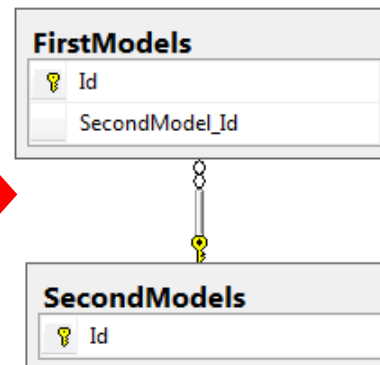
• 0..1-op-n

```
public class FirstModel
{
    public int Id { get; set; }

    [Required]
    public SecondModel SecondModel { get; set; }
}

public class SecondModel
{
    public int Id { get; set; }

    public IList<FirstModel> FirstModel { get; set; }
}
```



Relationships

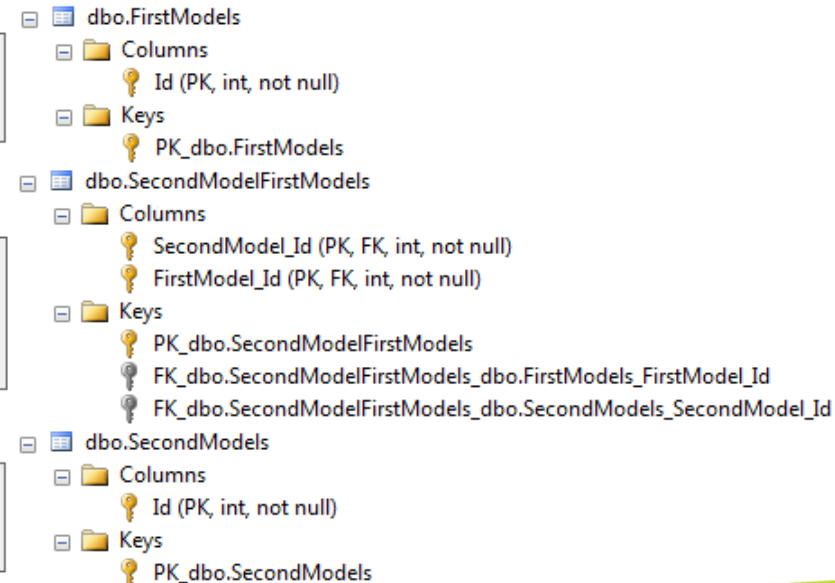
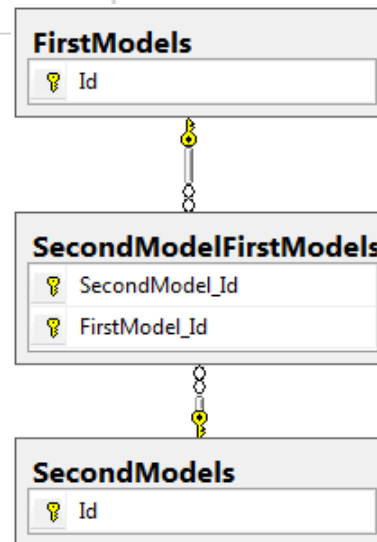
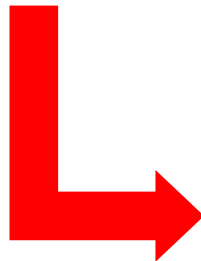
- n-op-n

```
public class FirstModel
{
    public int Id { get; set; }

    public IList<SecondModel> SecondModel { get; set; }
}

public class SecondModel
{
    public int Id { get; set; }

    public IList<FirstModel> FirstModel { get; set; }
}
```





Fluent API

Fluent API

- 'Fluent API' is een library binnen EF die kan worden gebruikt om in te spelen op de creatie van het DB-schema
- Basisklasse 'ModelBuilder'
 - via DbContext.OnModelCreating-methode
 - overschrijven in container klasse

```
class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure DB-schema creation through 'Fluent API' here!
    }
}
```

Conventions

Conventions

- Het algemeen gedrag van hoe EF het DB-schema afleid van models is vastgelegd in conventies welke gewijzigd kunnen worden
- Namespace
System.Data.Entity.ModelConfiguration.Conventions

```
using System.Data.Entity.ModelConfiguration.Conventions;

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure DB-schema creation through 'Fluent API' here!
        // Change
        modelBuilder.Conventions. ... ;
    }
}
```

Voorbeelden

```
using System.Data.Entity.ModelConfiguration.Conventions;

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure DB-schema creation through 'Fluent API' here!

        // Remove pluralizing tablename
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

        // Remove cascading delete on required-relationships
        modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>()
        modelBuilder.Conventions.Remove<ManyToManyCascadeDeleteConvention>()
    }
}
```

Type/Property Mapping

Type/Property Mapping

`ModelBuilder.Entity<ModelType>()[.Property(...)]`

- In plaats van conventies in algemeen te wijzigen kunnen conventies ook beïnvloed worden per type/model
 - op type-niveau
 - op property-niveau
- Data Annotation bieden een subset van deze functionaliteit aan via de techniek van attributes

Tabelnaam

- Data Annotation: TableAttribute
- Fluent API:

```
class FirstModel
{
    public int Id { get; set; }
}

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.Entity<FirstModel>().ToTable("tblFirstModel");
    }
}
```


Complexe types

- Data Annotation:
ComplexTypeAttribute
- Fluent API:

```
class FirstModel
{
    public int Id { get; set; }
    public string String { get; set; }

    public InnerModel InnerModel { get; set; }
}

[ComplexType]
class InnerModel
{
    public int InnerModelInteger { get; set; }
    public string InnerModelString { get; set; }
}
```

```
class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating
        (DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.ComplexType<InnerModel>();
    }
}
```

Primary Key (unique identifier)

- Data Annotation: KeyAttribute
- Fluent API:

```
class FirstModel
{
    public int UniqueIdentity { get; set; }
}

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.Entity<FirstModel>().HasKey(fm => fm.UniqueIdentity);
    }
}
```

Foreign Key

- Data Annotation: ForeignKeyAttribute
- Fluent API:

```
class FirstModel
{
    public int Id { get; set; }
    public int SecondModelFK { get; set; }

    // navigation-property (to 'SecondModel')
    public SecondModel SecondModelNavProp { get; set; }
}

class SecondModel
{
    public int SecondModelId { get; set; }

    // navigation-property (collection of 'FirstModel')
    public ICollection<FirstModel> FirstModels { get; set; }
}

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.Entity<FirstModel>().HasRequired(fm => fm.SecondModelNavProp).WithMany(sm => sm.FirstModels)
            .HasForeignKey(fm => fm.SecondModelFK);
    }
}
```

Index en/of unieke waarde

- Data Annotation: IndexAttribute
 - parameter 'IsUnique'
- Fluent API:

```
class FirstModel
{
    public int Id { get; set; }

    public string FrequentlyQueried { get; set; }
}

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.Entity<FirstModel>()
            .Property(fm => fm.FrequentlyQueried)
            .HasColumnAnnotation("Index", new IndexAnnotation(new IndexAttribute()));
    }
}
```

Not Mapped

- Data Annotation:
NotMappedAttribute
- Fluent API:

```
class FirstModel
{
    public int Id { get; set; }

    public string String { get; set; }
}

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.Entity<FirstModel>().Ignore(fm => fm.String);
    }
}
```

Verplicht

- Data Annotation: RequiredAttribute
- Fluent API:

```
class FirstModel
{
    public int Id { get; set; }

    public string RequiredString { get; set; }
}

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.Entity<FirstModel>().Property(fm => fm.RequiredString).IsRequired();
    }
}
```

Maximum lengte

- Data Annotation: MaxLengthAttribute of StringLengthAttribute
- Fluent API:

```
class FirstModel
{
    public int Id { get; set; }

    public string String { get; set; }
}

class MyEFDbContext : DbContext
{
    ...
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        ...
        modelBuilder.Entity<FirstModel>().Property(fm => fm.String).HasMaxLength(50);
    }
}
```

Relationship Mapping

Relationship Mapping

`ModelBuilder.Entity<ModelType>().Has...().With...()`

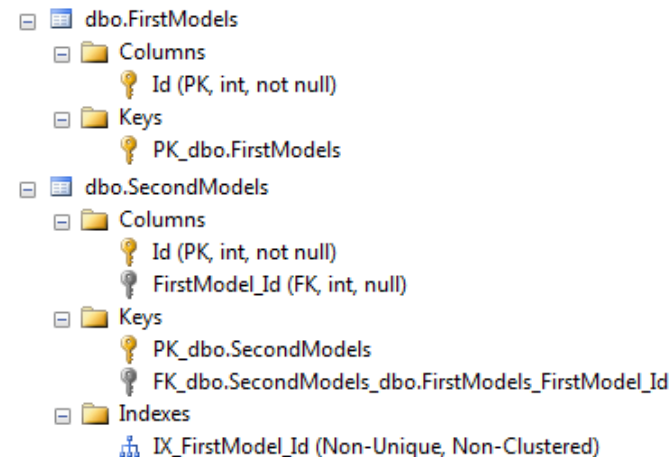
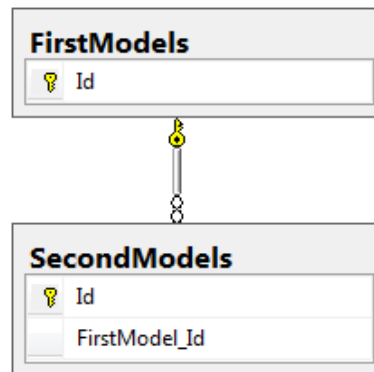
- 0..1-op-0..1
➔ in db: n-op-0..1 of 0..1-op-n
....HasOptional(...).WithOptionalPrincipal(...)
....HasOptional(...).WithOptionalDependent(...)

```
public class FirstModel
{
    public int Id { get; set; }

    public SecondModel SecondModel { get; set; }
}

public class SecondModel
{
    public int Id { get; set; }

    public FirstModel FirstModel { get; set; }
}
```



Relationship Mapping

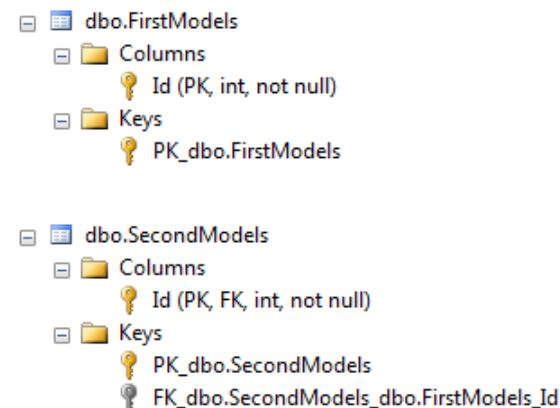
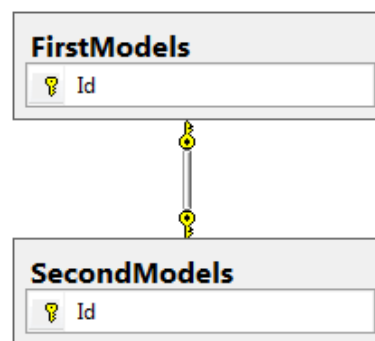
- 1-op-1
➔ in db: 1-op-0..1 of 0..1-op-1
....HasRequired(...).WithRequiredPrincipal(...)
....HasRequired(...).WithRequiredDependent(...)

```
public class FirstModel
{
    public int Id { get; set; }

    [Required]
    public SecondModel SecondModel { get; set; }
}

public class SecondModel
{
    public int Id { get; set; }

    [Required]
    public FirstModel FirstModel { get; set; }
}
```



Relationship Mapping

- 1-op-0..1
....HasOptional(...).WithRequired(...)
- 0..1-op-1
....HasRequired(...).WithOptional(...)
- 1-op-n
....HasMany(...).WithRequired(...)
- 0..1-op-n
....HasMany (...).WithOptional(...)
- n-op-n
....HasMany (...).WithMany(...)

Cascading delete

- In plaats van algemeen conventies voor 'x-op-n' of 'n-op-n' aan te passen kan je ook per relatie 'cascading delete' instellen

```
ModelBuilder.Entity<ModelType>()  
    .Has...().With...()  
    .WillCascadeOnDelete(bool)
```