

.NET Advanced

EF Core vervolg

Topics

- Aansturen generatie database objecten via EF Core
 - Conventies voor entiteiten
 - Configuratie m.b.v. Data Annotation Attributes
 - Configuratie m.b.v. Fluent API
- Data seed

EF Core vervolg

Conventies

EF Core vervolg - Conventies



Doel

- Het concept van conventies bij database generatie via EF Core kennen
- De meest courante conventies kennen

EF Core vervolg - Conventies

Theorie

- Drie manieren waarmee generatie van het EDM kan gestuurd worden
 - Conventies
 - Data Annotation Attributes op entiteiten
 - Configurations in Fluent API

(Via migration op basis van het EDM creatie/modificatie van database objecten)

EF Core vervolg - Conventies

Theorie

- Conventies
 - Ingebakken set van voorwaarden en regels in EF Core
 - Worden toegepast bij creatie van migration
 - EF Core gaat via reflection onderzoeken wat voldoet aan ingebakken voorwaarden
 - EF Core past ingebakken regels toe op datgene wat voldoet
 - Dit gedrag kan overruled worden door Data Annotation Attributes en/of Fluent API

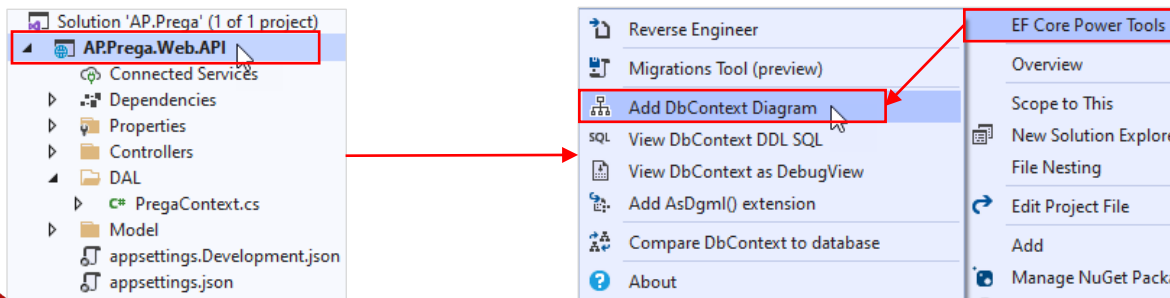
EF Core vervolg - Conventies

Theorie

- Hulp bij visualiseren van EDM m.b.v. te installeren extension 'EF Core power tools'

<https://github.com/ErikEJ/EFCorePowerTools/wiki>

- Eerst project (re)builden waarin DbContext is gedefinieerd
- Daarna right click op het project ⇒ EF Core Power Tools ⇒ Add DbContext Diagram



EF Core vervolg - Conventies

Theorie

Conventie 1 - Aanmaken van een **tabel**

- Tabel wordt aangemaakt voor elke DbSet<> property van een DbContext
- Naam van DbSet property wordt gebruikt als naam van de tabel
- Bevat de onderliggende entiteit van de DbSet een reference property waarvoor geen DbSet bestaat? \Rightarrow Ook voor die property wordt een tabel gemaakt
 - Reference property is een property met het type van het parent/child object dat fungeert als verwijzing = Navigation property
 - Navigation property kan ook uitgewerkt worden als letterlijk Foreign Key property (primitive type)

EF Core vervolg - Conventies

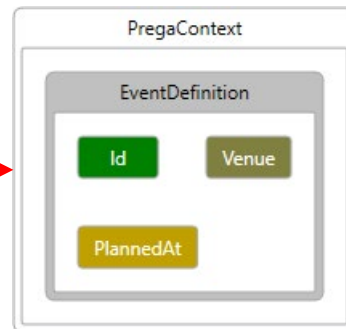
Voorbeelden

Entiteit zonder reference propert(y)(ies) en 1 DbSet

```
public class PregaContext : DbContext
{
    0 references
    public PregaContext(DbContextOptions<PregaContext> dbContextOptions) : base(dbContextOptions)
    {
    }

    3 references
    public DbSet<EventDefinition> EventDefinitions { get; set; }
}
```

```
public class EventDefinition
{
    2 references
    public int Id { get; set; }
    0 references
    public string Venue { get; set; }
    0 references
    public DateTime PlannedAt { get; set; }
}
```



EF Core vervolg - Conventies

Voorbeelden

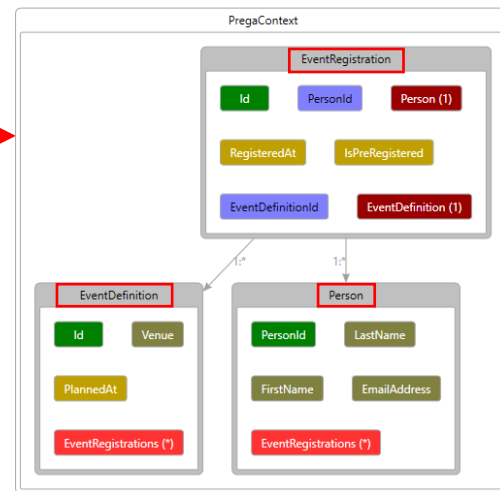
Entiteit(en) met reference properties maar geen DbSet voor gerefereerde entiteiten

```
public class PregaContext : DbContext
{
    0 references
    public PregaContext(DbContextOptions<PregaContext> dbContextOptions) : base(dbContextOptions)
    {
    }

    3 references
    public DbSet<EventDefinition> EventDefinitions { get; set; }
}
```

```
public class EventDefinition
{
    2 references
    public int Id { get; set; }
    0 references
    public string Venue { get; set; }
    0 references
    public DateTime PlannedAt { get; set; }
    0 references
    public List<EventRegistration> EventRegistrations { get; set; }
}
```

```
public class EventRegistration
{
    0 references
    public int Id { get; set; }
    0 references
    public int EventDefinitionId { get; set; }
    0 references
    public int PersonId { get; set; }
    0 references
    public DateTime RegisteredAt { get; set; }
    0 references
    public bool IsPreRegistered { get; set; }
    0 references
    public EventDefinition EventDefinition { get; set; }
    0 references
    public Person Person { get; set; }
}
```



EF Core vervolg - Conventies

Theorie

Conventie 2 - Aanmaken van een **kolom** binnen een tabel

- Gebeurt voor elke **scalaire** (single value) property van de entiteit
 - Dus geen reference properties
 - En geen properties die een collection bevatten
- Naam van de property wordt daarbij gebruikt als kolomnaam

EF Core vervolg - Conventies

Theorie

Conventie 3 - Bepalen van het **datatype** van een kolom

- .NET types <> Database types
 - Bv string vs (n)varchar
- Mappen van het .NET type naar een Database type
- Afhankelijk van Database Provider

EF Core vervolg - Conventies

Theorie

Mapping .NET ⇔ SQL Server

.NET type	SQL Server type
byte	tinyint
short	smallint
int	int
long	bigint
String	nvarchar(max) ⇔ n staat voor unicode
decimal	decimal(18,2)
float	real
byte[]	varbinary(max)
datetime	datetime
bool	bit
double	float

SQL Server datatypes zonder mapping

.NET type
char
sbyte
object

EF Core vervolg - Conventies

Theorie

Conventie 4 - Bepalen of een kolom al dan niet **Null** toelaat.

- Kolom = primary key? \Rightarrow Null **not** allowed
- .NET type = primitive type? \Rightarrow Null **not** allowed
- .NET type = nullable primitive type? \Rightarrow Null allowed (Bv. `Nullable<int>`, `int?`)
- .NET type = reference type? \Rightarrow Null allowed

EF Core vervolg - Conventies

Theorie

Conventie 5 - Bepalen van de **primary key** van een tabel

- Gebeurt op basis van naamgeving property
- Namen die in aanmerking komen:

Id
ID
iD
id

OF

<entity class name>Id
<entity class name>ID
<entity class name>iD
<entity class name>id

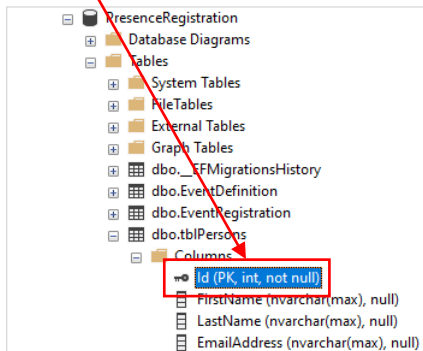
EF Core vervolg - Conventies

Voorbeelden

Id

```
public class Person
{
    // References
    public int Id { get; set; }
    // References
    public string FirstName { get; set; }
    // References
    public string LastName { get; set; }
    // References
    public string EmailAddress { get; set; }

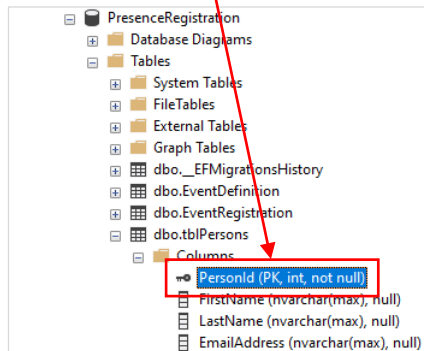
    // References
    public List<EventRegistration> EventRegistrations { get; set; }
}
```



<entity class name>Id

```
public class Person
{
    // References
    public int PersonId { get; set; }
    // References
    public string FirstName { get; set; }
    // References
    public string LastName { get; set; }
    // References
    public string EmailAddress { get; set; }

    // References
    public List<EventRegistration> EventRegistrations { get; set; }
}
```



EF Core vervolg - Conventies

Theorie

Conventie 6 - Instellen van **automatische nummering** voor de **primary key**

- Wordt automatisch ingesteld als de primary key
 - Een non-composite primary key is (bestaat uit slechts 1 veld)
 - Een geheel getal als datatype heeft

▼ Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Geeft aan of automatische nummering actief is

Bepaalt de waarde waarmee per rij verhoogd wordt

Bepaalt de startwaarde voor de allereerste rij

EF Core vervolg - Conventies

Theorie

Conventie 7 - Bepalen van een **foreign key** kolom van een tabel

- In principal entity (parent)
 - **Moet** een reference property bestaan naar de child entiteit(en)
 - Moet een collection van het type van de dependent entity zijn in geval van een one-to-many relatie

```
public class EventDefinition
{
    2 references
    public int Id { get; set; }
    0 references
    public string Venue { get; set; }
    0 references
    public DateTime PlannedAt { get; set; }
    0 references
    public List<EventRegistration> EventRegistrations { get; set; }
}
```

EF Core vervolg - Conventies

Theorie

Conventie 7 - Bepalen van een **foreign key** kolom van een tabel

- In dependent entity (child)
 - Moet een reference property bestaan naar de principal entity
 - Moet deze property het type van de principal entity hebben als type

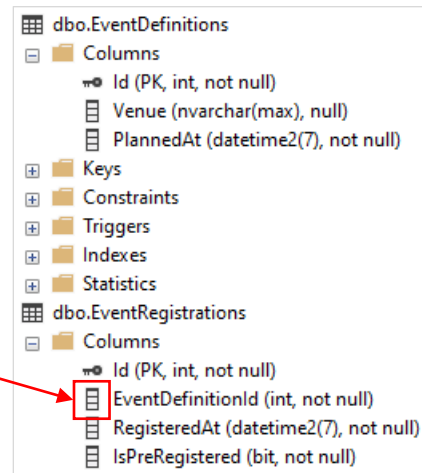
```
public class EventRegistration
{
    References
    public int Id { get; set; }
    References
    public DateTime RegisteredAt { get; set; }
    References
    public bool IsPreRegistered { get; set; }
    References
    public EventDefinition EventDefinition { get; set; }
}
```

EF Core vervolg - Conventies

Voorbeelden

```
public class EventRegistration
{
    0 references
    public int Id { get; set; }
    0 references
    public int EventDefinitionId { get; set; }
    0 references
    public DateTime RegisteredAt { get; set; }
    0 references
    public bool IsPreRegistered { get; set; }
}
```

```
public class EventDefinition
{
    2 references
    public int Id { get; set; }
    0 references
    public string Venue { get; set; }
    0 references
    public DateTime PlannedAt { get; set; }
}
```



- Foreign Key property in dependent entiteit `EventRegistration`
- Geen reference property in principal entiteit `'EventDefinition'`

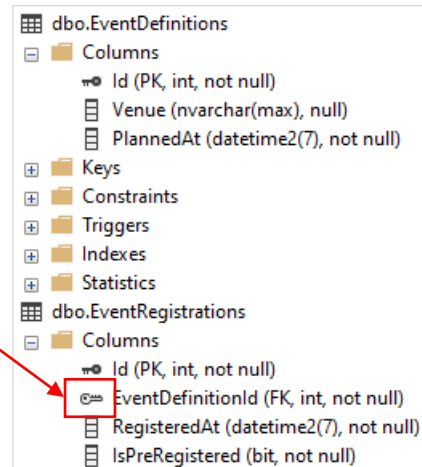
Gevolg \Rightarrow Kolom `'EventDefinitionId'` wordt niet als Foreign Key kolom aangemaakt

EF Core vervolg - Conventies

Voorbeelden

```
public class EventRegistration
{
    0 references
    public int Id { get; set; }
    0 references
    public int EventDefinitionId { get; set; }
    0 references
    public DateTime RegisteredAt { get; set; }
    0 references
    public bool IsPreRegistered { get; set; }
}
```

```
public class EventDefinition
{
    2 references
    public int Id { get; set; }
    0 references
    public string Venue { get; set; }
    0 references
    public DateTime PlannedAt { get; set; }
    0 references
    public List<EventRegistration> EventRegistrations { get; set; }
}
```



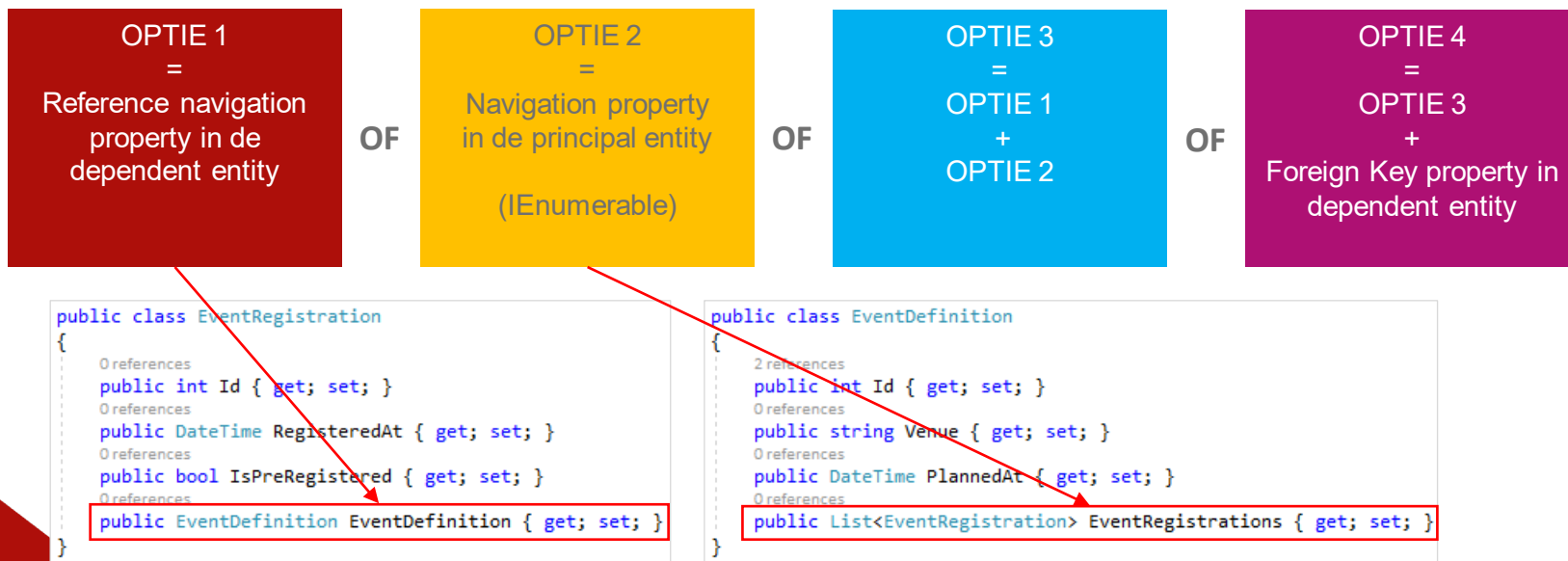
- Foreign Key property in dependent entiteit EventRegistration
- Reference property in principal entiteit 'EventDefinition'

Gevolg ⇒ Kolom 'EventDefinitionId' wordt als Foreign Key kolom aangemaakt

EF Core vervolg - Conventies

Theorie

Conventie 8 - Bepalen van een **one to many relationship** tussen 2 tabellen



EF Core vervolg

Data Annotation Attributes

EF Core vervolg - Data Annotation Attributes



Doel

- Het concept van configuratie m.b.t. het opbouwen van het EDM kennen
- Het concept van Data Annotation Attributes m.b.t. het opbouwen van het EDM kennen
- De meest courante Data Annotation Attributes kennen
- Het EDM kunnen configureren m.b.v. Data Annotation Attributes

EF Core vervolg - Data Annotation Attributes

Theorie

- Configuratie
 - Overschrijven van conventies
 - In geval dingen anders moeten dan via standaard conventies
 - Ofwel via Data Annotation Attributes in entity class (POCO?) ofwel via Fluent API

EF Core vervolg - Data Annotation Attributes

Theorie

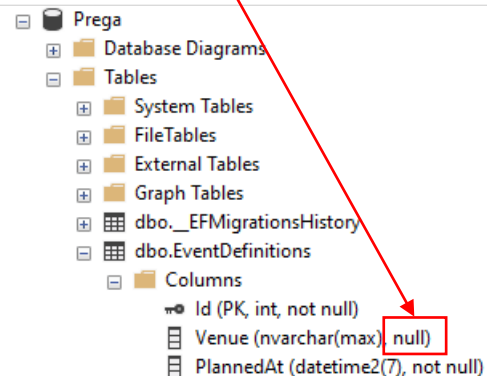
- Data Annotation Attributes
 - Attributen die we toepassen op een entity class op niveau van
 - Class
 - Property
 - Terug te vinden onder namespaces
 - `System.ComponentModel.DataAnnotations`
 - `System.ComponentModel.DataAnnotations.Schema`

EF Core vervolg - Data Annotation Attributes

Voorbeelden

- Venue is een reference type
- Reference types zijn by default nullable

```
public class EventDefinition
{
    2 references
    public int Id { get; set; }
    0 references
    public string Venue { get; set; }
    0 references
    public DateTime PlannedAt { get; set; }
    0 references
    public List<EventRegistration> EventRegistrations { get; set; }
}
```



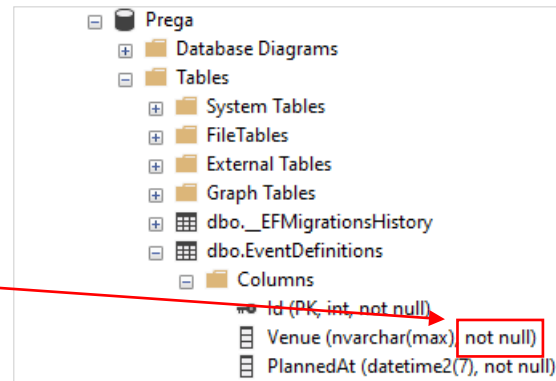
EF Core vervolg - Data Annotation Attributes

Voorbeelden

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

```
namespace AP.Prega.Web.API.Model
```

```
{  
    6 references  
    public class EventDefinition  
    {  
        2 references  
        public int Id { get; set; }  
        [Required]  
        0 references  
        public string Venue { get; set; }  
        0 references  
        public DateTime PlannedAt { get; set; }  
        0 references  
        public List<EventRegistration> EventRegistrations { get; set; }  
    }  
}
```



M.b.v. het [Required] attribuut zorgen we ervoor dat een veld vereist wordt in de database

EF Core vervolg - Data Annotation Attributes

Theorie

Courante attributen in namespace `System.ComponentModel.DataAnnotations`

Attribuut	Omschrijving
Key	Maakt van de property de primary key van de tabel. Niet bruikbaar in geval van een composite key!
Required	Voegt aan de kolom in de tabel de integriteitsregel NOT NULL toe
MinLength	Voegt aan de kolom in de tabel de integriteitsregel minimumlengte toe (*)
MaxLength	Voegt aan de kolom in de tabel de integriteitsregel maximumlengte toe (*)

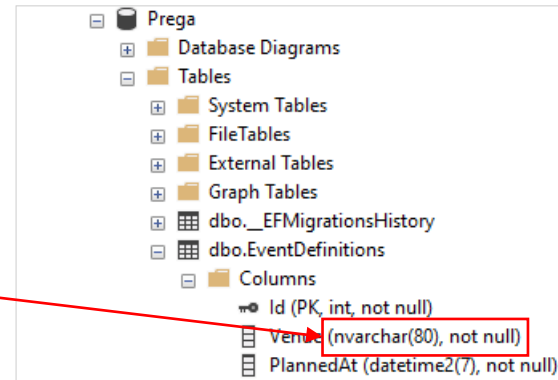
(*) De bijhorende waarde wordt tussen haakjes geplaatst bv. `MaxLength(80)`

EF Core vervolg - Data Annotation Attributes

Voorbeelden

```
using System.ComponentModel.DataAnnotations;

namespace AP.Prega.Web.API.Model
{
    6 references
    public class EventDefinition
    {
        2 references
        public int Id { get; set; }
        [Required, MaxLength(80)]
        0 references
        public string Venue { get; set; }
        0 references
        public DateTime PlannedAt { get; set; }
        0 references
        public List<EventRegistration> EventRegistrations { get; set; }
    }
}
```



Meerdere attributen kunnen gecombineerd worden door ze te scheiden met een komma

EF Core vervolg - Data Annotation Attributes

Theorie

Courante attributen in namespace `System.ComponentModel.DataAnnotations.Schema`

Attribuut	Omschrijving
Table	Toe te passen op niveau van de entity class. Maakt de tabel aan in de database met de opgegeven naam (tussen haakjes).
Column	Toe te passen op niveau van de property. Maakt de kolom aan in de tabel met de opgegeven naam (tussen haakjes).
ForeignKey	Toe te passen op niveau van de property. Maakt een Foreign Key van de kolom. Bekijk hier voorbeelden
NotMapped	Toe te passen op niveau van entity class of property. Neemt de entiteit/property niet mee op in het EDM (en bijgevolg database).
DatabaseGenerated	Toe te passen op niveau van de property. Geeft aan hoe de waarde te genereren.

EF Core vervolg - Data Annotation Attributes

Voorbeelden

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

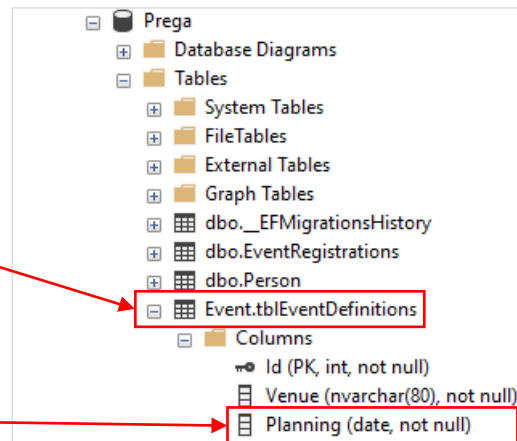
namespace AP.Prega.Web.API.Model
{
    [Table("tblEventDefinitions", Schema="Event")]
    public class EventDefinition
    {
        public int Id { get; set; }

        [Required, MaxLength(80)]
        public string Venue { get; set; }

        [Column("Planning", TypeName = "date")]
        public DateTime PlannedAt { get; set; }

        [NotMapped]
        public string PlannedAtVenue
        {
            get { return $"{Venue} ({PlannedAt})"; }
        }

        public List<EventRegistration> EventRegistrations { get; set; }
    }
}
```



EF Core vervolg

Fluent API

EF Core vervolg - Fluent API



Doel

- Het concept van Fluent API m.b.t. het opbouwen van het EDM kennen
- Het concept van Configuration classes m.b.t. het opbouwen van het EDM kennen
- Het EDM kunnen configureren m.b.v. Fluent API
- Configuration classes kunnen opbouwen m.b.v. Fluent API
- Configuration classes kunnen gebruiken om het EDM op te bouwen

EF Core vervolg - Fluent API

Theorie

- POCO = Plain Old CLR Object (definitie [hier](#))
- Entity class = POCO

⇒ Entity class met attributen = POCO?

a simple object created in the .NET Common Language Runtime (CLR) that is unencumbered by inheritance or attributes.

- Conventies niet overschrijven in entity classes
- Gebruik in plaats daarvan Fluent API via een modelBuilder object
- Fluent API biedt meer mogelijkheden dan Data Annotation Attributes

EF Core vervolg - Fluent API

Theorie

- Uit te werken in OnModelCreating() method van DbContext class
 - Overriden van base class method!
- modelBuilder object meegeven als argument
- Configuratie instelbaar op 3 niveaus
 - Model (database)
 - Entity (tabellen en relaties)
 - Property (kolom)
- Configureren m.b.v. method chaining van modelBuilder object
- Overzicht van beschikbare methods [hier](#)

EF Core vervolg - Fluent API

Voorbeelden

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.Entity<EventDefinition>()
        .ToTable("tblEventDefinitions", "Events")
        .Ignore(p => p.PlannedAtVenue)
        .HasKey(p => p.Id);

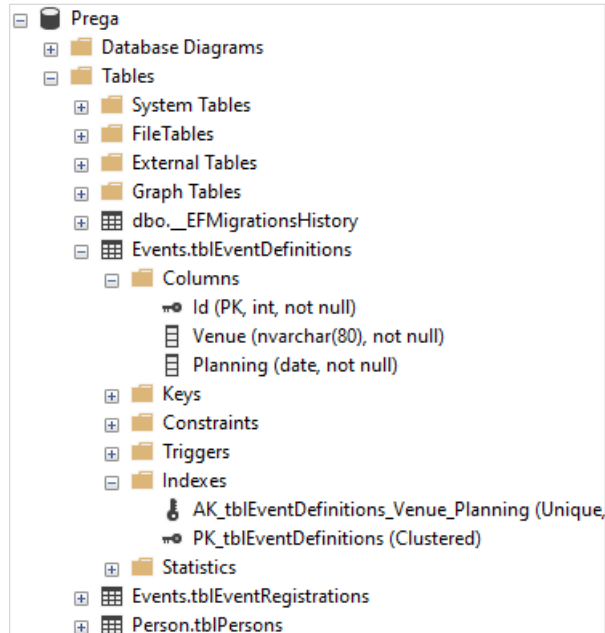
    builder.Entity<EventDefinition>().HasAlternateKey(p => new { p.Venue, p.PlannedAt });

    builder.Entity<EventDefinition>().Property(p => p.Venue)
        .IsRequired()
        .HasColumnType("nvarchar")
        .HasMaxLength(80);

    builder.Entity<EventDefinition>().Property(p => p.PlannedAt)
        .HasColumnName("Planning")
        .HasColumnType("date");

    builder.Entity<EventRegistration>()
        .ToTable("tblEventRegistrations", "Events");

    builder.Entity<Person>()
        .ToTable("tblPersons", "Person");
}
```



EF Core vervolg - Fluent API

Theorie

- Bij groot aantal configuraties wordt OnModelCreating() method onverzichtelijk
- Oplossing
 - Bundelen van configuratie van een entiteit in aparte class
 - Per entiteit class voorzien
 - Hanteer naamgeving conventie “<entity class name>Configuration”
 - Plaats de classes in een aparte map in het project (bv. Configuration)

EF Core vervolg - Fluent API

Voorbeelden

```
public class EventDefinitionConfiguration : IEntityTypeConfiguration<EventDefinition>
{
    0 references
    public void Configure(EntityTypeBuilder<EventDefinition> builder)
    {
        builder.ToTable("tblEventDefinitions", "Events")
            .Ignore(p => p.PlannedAtVenue)
            .HasKey(p => p.Id);

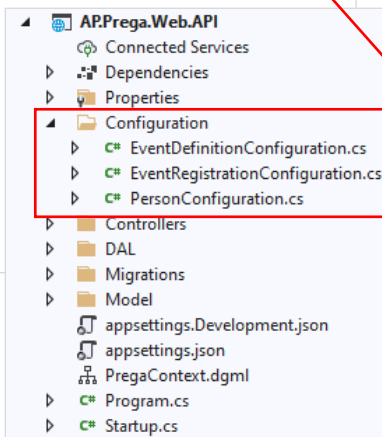
        builder.HasAlternateKey(p => new { p.Venue, p.PlannedAt });

        builder.Property(p => p.Venue)
            .IsRequired()
            .HasColumnType("nvarchar")
            .HasMaxLength(80);

        builder.Property(p => p.PlannedAt)
            .HasColumnName("Planning")
            .HasColumnType("date");
    }
}
```

Class implementeert interface `IEntityTypeConfiguration<TEntity>` uit namespace `Microsoft.EntityFrameworkCore`

Voorzie method `Configure()` met `EntityTypeBuilder<TEntity>` uit namespace (namespace `Microsoft.EntityFrameworkCore.Metadata.Builders`) als argument. In deze methode werk je de configuratie uit met Fluent API



EF Core vervolg - Fluent API

Theorie

- Toepassen van de configuration in OnModelCreating() method van de DbContext
- Voer de method ApplyConfiguration<TEntity> van de modelBuilder instance uit voor elke entiteit waarvoor een Configuration class werd voorzien
- Geef een instance van de Configuration class mee als argument

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.ApplyConfiguration(new EventDefinitionConfiguration());
    builder.ApplyConfiguration(new EventRegistrationConfiguration());
    builder.ApplyConfiguration(new PersonConfiguration());
}
```


EF Core vervolg - Fluent API

Theorie

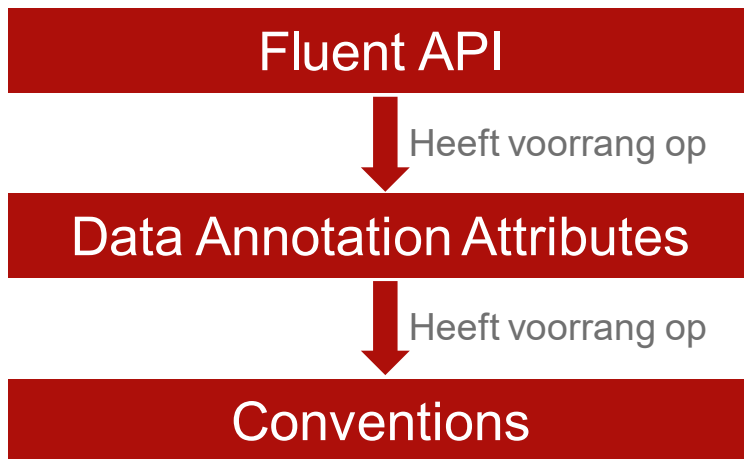
- Toepassen van de configuration in OnModelCreating() method van de DbContext
- Sinds EF Core 2.2 kan dit ook gebundeld worden via de extension method ApplyConfigurationsFromAssembly()
 - System.Reflection toevoegen aan using statements
- Scant de als argument meegegeven Assembly op classes die de interface IEntityTypeConfiguration implementeren en registreert deze

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
}
```

EF Core vervolg - Fluent API

Theorie

- Welke technologie heeft voorrang bij configuratie van het EDM



EF Core vervolg

Data seed

EF Core vervolg - Data seed

Doel

- Het concept van data seeding kennen
- Initiële data kunnen voorzien m.b.v. EF Core

EF Core vervolg - Data seed

Theorie

- Data seeding is het voorzien van initiële data voor een app die nodig is om te kunnen starten
- Vanuit de `OnModelCreating()` method in de `DbContext`
 - ⇒ Voer de method `HasData()` uit op ...
 - ⇒ Het resultaat van de method `Entity<TEntity>()`
 - ⇒ Van het `ModelBuilder` object
 - ⇒ Geef 1 of meer entity instances mee als argument aan de `HasData()` method
- Bij creatie van een migration wordt het toevoegen van gegevens mee opgenomen in de `up()` method
- Bij het uitvoeren van update-database zullen de nodige inserts uitgevoerd worden

EF Core vervolg - Data seed

Voorbeelden

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());

    builder.Entity<Person>().HasData(
        new Person { FirstName = "John", LastName = "Doo", EmailAddress="john.doo@johndoo.com"}
    );
}
```

1. Voer de method `HasData()` uit op ...

2. Het resultaat van de method `Entity<TEntity>()`

3. Van het `ModelBuilder` object



EF Core vervolg - Data seed

Voorbeelden

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());

    builder.Entity<Person>().HasData(
        new Person { FirstName = "John", LastName = "Doo", EmailAddress="john.doo@johndoo.com"}
    );
}
```

Package Manager Console

Package source: All  Default project: AP.Prega.Web.API 

```
PM> add-migration Initial
Build started...
Build succeeded.
```

```
String contextType, String namespace)
    at Microsoft.EntityFrameworkCore.Design.OperationExecutor.AddMigration.<>c__DisplayClass0_0.<.ctor>b__0()
    at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.<>c__DisplayClass3_0`1.<Execute>b__0()
    at Microsoft.EntityFrameworkCore.Design.OperationExecutor.OperationBase.Execute(Action action)
The seed entity for entity type 'Person' cannot be added because a non-zero value is required for property 'Id'.
Consider providing a negative value to avoid collisions with non-seed data.
PM>
```

EF Core vervolg - Data seed

Theorie

- Wat loopt er mis?
 - ⇒ Er wordt geen value voorzien voor de Primary Key
 - ⇒ `The seed entity for entity type 'Person' cannot be added because a non-zero value is required for property 'Id'. Consider providing a negative value to avoid collisions with non-seed data.`
- We moeten zelf de waarde voorzien!
- Deze mag nadien niet meer veranderd worden omdat EF Core deze zal gebruiken ter vergelijking bij eventuele volgende migrations
- Gebruik ook nooit code die waarden genereert voor een migration aangezien
 - Deze zullen bij elke migration andere waarden opleveren!


EF Core vervolg - Data seed

Voorbeelden

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());

    builder.Entity<Person>().HasData(
        new Person { Id = 1, FirstName = "John", LastName = "Doo", EmailAddress="john.doo@johndoo.com"}
    );
}
```

Package Manager Console

Package source: All  Default project: AP.Prega.Web.API 

```
PM> add-migration Initial
Build started...
Build succeeded.
```

```
migrationBuilder.InsertData(
    schema: "Person",
    table: "tblPersons",
    columns: new[] { "Id", "EmailAddress", "FirstName", "LastName" },
    values: new object[] { 1, "john.doo@johndoo.com", "John", "Doo" });
```

Oefeningen

Deel 3 - Oefenbundels 1 & 2

- Surf naar Digitap
- Download de opgave
- Los de vragen op

EF Core vervolg



Bronnen

- <https://www.entityframeworktutorial.net/efcore/conventions-in-ef-core.aspx>
- <https://www.entityframeworktutorial.net/code-first/dataannotation-in-code-first.aspx>
- https://en.wikipedia.org/wiki/Plain_old_CLR_object
- <https://www.learnentityframeworkcore.com/configuration/fluent-api>
- <https://www.tektutorialshub.com/entity-framework-core/entitytype-configuration-ef-core/>
- <https://www.learnentityframeworkcore.com/migrations/seeding>
- <https://www.tektutorialshub.com/entity-framework-core/ef-core-data-seeding/>