

Online on-demand Project Support



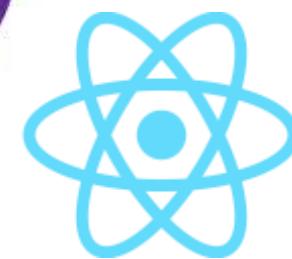
# EF Core 6.0



Thorsten Kansy ([tkansy@dotnetconsulting.eu](mailto:tkansy@dotnetconsulting.eu))

# Meine Person- Thorsten Kansy

Freier Consultant, Software Architekt,  
Entwickler, Trainer & Fachautor



Azure Cosmos DB



# Mein Service- Ihr Benefit

- Individuelle Inhouse Trainings
- (Online on-demand) Projektbegleitung
- Beratung
  - Problemanalyse und Lösungen
  - Technologieentscheidungen



# Motivation



# Motivation

- Übersicht
- Grundlagen
- Wichtiges
- Spaß

# Agenda

- Grundlagen
- Entity Model
- Beziehungen
- Arbeiten mit Entitäten
- Eigenschaft oder Feld?
- Ladestrategien
- Änderungen verfolgen
- Direct SQL
- Graph Update
- Concurrency
- Sequenzen
- Logging
- Database Handling
- Transaktionen
- Value Generator
- Migration
- Vererbungs-/ Mappingstrategie
- Indizes
- Berechnete Spalten
- Database Functions
- Self-contained Type configuration
- Shadow Property
- Global Query Filter
- Uvm.

# Grundlagen



# Unterstützte Database-Provider

Relationale Datenbanken	NoSQL Datenbanken
Microsoft SQL Server	Azure Cosmos DB >=3.0
SQLite	
IBM Data Server (DB2)	
Oracle	
MySQL	
SQL Compact	=2.2
PostgreSQL	
...	
InMemory	

# Installation

- **Nuget**

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

```
Install-Package Microsoft.EntityFrameworkCore.SqlLite
```

```
Install-Package Microsoft.EntityFrameworkCore.PostgreSQL
```

```
Install-Package Microsoft.EntityFrameworkCore.<DbProvider>
```

- **Zusätzliche Funktionalitäten (z.B. Migration)**

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

# DbContext



# DbContext

- Zentrales Element
- Ausführen von Abfragen
- Hinzufügen/ Entfernen von Entitäten
- Registrierung von Änderungen an Entitäten

# Konfiguration

- Verhalten des Kontextes
- Datenbank Provider festlegt
- Logging

# Standalone Konfiguration

## Via DbContextOptionsBuilder

```
// Optionen erzeugen
var optionsBuilder = new DbContextOptionsBuilder<MyDbContext>();
optionsBuilder.UseSqlite(@"Data Source=c:\temp\Storage.db");
optionsBuilder.ConfigureWarnings(s =>
{
    // s.Ignore(RelationalEventId.QueryPossibleExceptionWithAggregateOperatorWarning);
    // ODER
    // s.Log(RelationalEventId.QueryPossibleExceptionWithAggregateOperatorWarning);
    // ODER
    s.Throw(RelationalEventId.QueryPossibleExceptionWithAggregateOperatorWarning);
});

using (MyDbContext context = new MyDbContext(optionsBuilder.Options)) { ... }
```



**Demo**

# IoC Konfiguration

## via IConfigurationRoot

```
private static IServiceProvider createDependencyInjectionContainer(IConfigurationRoot config)
{
    // EF Context konfigurieren
    .AddDbContext<SamplesContext1>(
        o => o.UseSqlServer(config["ConnectionStrings:EFConnectionString"])
            .ConfigureWarnings(w =>
                w.Throw
```



**Demo**

# Entity Types (Model)



# Entitäten (Entity Types)

## Klassen (einfache POCOs)

Konstruktor ohne Parameter

Konstruktor mit Parameter

2.1

```
public class Speaker
{
    public readonly int Id;
    public string Name { get; set; }

    private Speaker(int id)
    {
        this.Id = id;
    }
    ...
}
```

### Ctor Injecting

DbContext

ILazyLoader

Action<object, string>

IEntityType

# Entity Modell erzeugen/ verfeinern

- Via Attribute
- Fluent API

```
protected override void DbContext:OnModelCreating (ModelBuilder modelBuilder)  
{...}
```

# Entity Modell erzeugen/ verfeinern

Wozu eigentlich?

- Anreichern von .NET unbekannten Details
  - z.B. varchar (n) oder nvarchar (n)
  - date oder time
- Features der Datenbank nutzen
  - Indizes
  - Einschränkungen, Fremdschlüssel
  - Sequenzen, etc.
- Modell so gestalten wie es benötigt wird

# HasDefaultNamespace (Fluent API)

- Legt das Standardschema für alle Objekte fest
- Standard: Provider abhängig, beim SQL Server: dbo

```
modelBuilder.HasDefaultSchema("dnc");
```

# UseCollation (Fluent API)

- Legt die Standardcollation für die Datenbank/ Spalte fest
- Standard: Serverstandard  
(vermutlich SQL\_Latin1\_General\_CI\_AS)

```
modelBuilder.UseCollation("German_PhoneBook_CI_AS");
```

```
modelBuilder
    .Entity<User>()
    .Property(e => e.Name)
    .UseCollation("German_PhoneBook_CI_AS");
```

# Ignore (Fluent API)

- Ignoriert einen Typen komplett
- Standard: -

```
modelBuilder.Ignore<Session>();
```

# Ignore (Fluent API)

- Ignoriert die Eigenschaft einer Entität
- Standard: –

```
modelBuilder.Entity<Session>()  
    .Ignore(p => p.End);
```

# ToTable/ ToView (Fluent API)

- Name der Schema/ Tabelle o. Sicht einer Entität
- Standard: Name der DbSet<>-Eigenschaft im DbContext(!)

```
modelBuilder.Entity<Session>()
    .ToTable("Sessions");

modelBuilder.Entity<Session>()
    .ToTable("Session", "dbo");

modelBuilder.Entity<SpeakerStatistics>()
    .HasNoKey()
    .ToView("vwSpeakerStats", "dbo");
```

# IsMemoryOptimized (Fluent API)

- Memory optimized table (SQL Server)
- Standard: True

```
modelBuilder.Entity<Session>()
    .ToTable("Sessions")
    .IsMemoryOptimized();
```

# IsTemporal (Fluent API)

- Temporal table (SQL Server)
- Standard: -

```
modelBuilder.Entity<MyTemporalTable>()
    .ToTable(t => t.IsTemporal(hist =>
    {
        hist.UseHistoryTable("SessionHistory");
    });
}
```

```
dbContext.MyTemporalTable
    .TemporalAsOf(<Date>).Where(w => ...);
```

# Temporal AsOf & Co

TemporalAll()	Alle Daten aus der Tabelle und der historischen Tabelle.
TemporalAsOf(...)	Daten zu einem bestimmten Zeitpunkt.
TemporalFromTo(...)	Daten aus der Tabelle plus der historischen Tabelle exk. Einschluss der oberen Grenze.
TemporalBetween(...)	Daten aus der Tabelle plus der historischen Tabelle inkl. Einschluss der oberen Grenze.
TemporalContainedIn(...)	Nur Daten aus den der historischen Tabelle aus dem Zeitraum.

# HasKey (Fluent API)

- Eindeutiger Schlüssel der Entität (Primary Key)
- Standard: Id oder { Klasse } Id

```
modelBuilder.Entity<Session>()
    .HasKey(k => k.Id)
    .HasName("MyPkName");

modelBuilder.Entity<Session>()
    .HasKey(k => new { k.Id, k.SpeakerId });
```

# HasAlternateKey (Fluent API)

- Alternativer, eindeutiger Schlüssel der Entität
- Standard: –

```
modelBuilder.Entity<Session>()
    .HasAlternateKey(a => a.Title);

modelBuilder.Entity<Session>()
    .HasAlternateKey(a => new { a.Title, a.Abstract });
```

# HasColumnName (Fluent API)

- Name der Spalte in der Datenbank
- Standard: Name des Attributes

```
modelBuilder.Entity<Session>()
    .Property(p=> p.Abstract)
    .HasColumnName("ContentDescription");
```

# HasColumnType/ HasPrecision (Fluent API)

- Datenbanktyp/ Präzision
- Standard: Abhängig vom CLR-Datentyp

```
modelBuilder.Entity<Session>()
    .Property(p => p.Begin)
    .HasColumnType("time");
```

```
modelBuilder.Entity<Session>()
    .Property(p => p.Rating)
    .HasPrecision(18, 2); 5.0
```

# IsRequired (Fluent API)

- Gibt an, ob es sich um ein Pflichtfeld handelt
- Standard: Nein, außer EntityKey

```
modelBuilder.Entity<Session>()
    .Property(p => p.Abstract)
    .IsRequired();
```

# IsUnicode (Fluent API)

- Gibt an, ob es sich um eine Unicode-Zeichenkette handelt
- Standard: Ja

```
modelBuilder.Entity<Session>()
    .Property(p => p.Abstract)
    .IsUnicode();
```

# IsSparse (Fluent API)

- Gibt an, Sparse Column handelt
- Standard: False

```
modelBuilder.Entity<Session>()
    .Property(p => p.Abstract)
    .IsSparse();
```

# HasMaxLength (Fluent API)

- Gibt die maximale Länge von Zeichenketten an
- Standard: 4000

```
modelBuilder.Entity<Session>()
    .Property(p => p.Abstract)
    .HasMaxLength(300);
```

# HasDefaultValue (Fluent API)

- Legt einen (konstanten) typensicheren Standardwert fest
- Standard: -

```
modelBuilder.Entity<Session>()
    .Property(s => s.Title)
    .HasDefaultValue("<New Session>");
```

# HasDefaultValueSql (Fluent API)

- Legt einen (berechneten) Standardwert fest
- Standard: -

```
modelBuilder.Entity<Session>()
    .Property(s => s.Created)
    .HasDefaultValueSql("getdate()");
```

# HasIndex (Fluent API)

- (Datenbank)-Index
- Standard: -

```
modelBuilder.Entity<Session>()
    .HasIndex(i => i.Title);
```

# Useldentity (Fluent API)

- Identitätsspalte
- Standard: Seed = 1 & Increment = 1

```
modelBuilder.Entity<Session>()
    .Property(s => s.Created).UseIdentity();
```

# HasConversion (Fluent API)

- Konvertierung Speicher <-> Store
- Null wird niemals übergeben

```
Modelbuilder.Entity<Session>().Property(p => p.Homepage)
    .HasConversion(
        c => c.ToLower(), // Vor schreiben in Store
        c => c           // Nach lesen aus Store
    );
```

# Eingebaute Konvertierungen

## Microsoft.EntityFrameworkCore.Storage.Convertisers

- `BoolToZeroOneConverter` - Bool to zero and one
- `BoolToStringConverter` - Bool to strings such as "Y" and "N"
- `BoolToTwoValuesConverter` - Bool to any two values ← 
- `BytesToStringConverter` - Byte array to Base64-encoded string
- `CastingConverter` - Conversions that require only a Csharp cast
- `CharToStringConverter` - Char to single character string
- `DateTimeOffsetToBinaryConverter` - DateTimeOffset to binary-encoded 64-bit value
- `DateTimeOffsetToBytesConverter` - DateTimeOffset to byte array
- `DateTimeOffsetToStringConverter` - DateTimeOffset to string
- `DateTimeToBinaryConverter` - DateTime to 64-bit value including DateTimeKind
- `DateTimeToStringConverter` - DateTime to string
- `DateTimeToTicksConverter` - DateTime to ticks
- `EnumToNumberConverter` - Enum to underlying number
- `EnumToStringConverter` - Enum to string
- `GuidToBytesConverter` - Guid to byte array
- `GuidToStringConverter` - Guid to string
- `NumberToBytesConverter` - Any numerical value to byte array
- `NumberToStringConverter` - Any numerical value to string
- `StringToBytesConverter` - String to UTF8 bytes
- `TimeSpanToStringConverter` - TimeSpan to string
- `TimeSpanToTicksConverter` - TimeSpan to ticks



**Demo**



# Pre-convention model configuration

# Pre-convention model configuration

## Standard Konvention

```
protected override void ConfigureConventions(ModelConfigurationBuilder configurationBuilder)
{
    // Standard Konvention festlegen
    configurationBuilder.Properties<string>()
        .HaveMaxLength(100);
    configurationBuilder.Properties<decimal>()
        .HavePrecision(18, 3);

    // Auch Interfaces sind erlaubt
    configurationBuilder.IgnoreAny<INonPersisted>();

    base.ConfigureConventions(configurationBuilder);
}
```



**Demo**



# Beziehungen

# 1:n-Beziehungen

```
modelBuilder.Entity<Session>()
    .HasOne(p => p.TechEvent)
    .WithMany(p => p.Sessions)
    .HasConstraintName("MyFKConstraint")
    .HasForeignKey(p => p.TechEventId)
    .OnDelete(DeleteBehavior.Cascade);

modelBuilder.Entity<TechEvent>()
    .HasMany(p => p.Sessions)
    .WithOne(p => p.TechEvent)
    .IsRequired(true)
    .HasConstraintName("MyFKConstraint")
    .HasForeignKey(p => p.TechEventId)
    .OnDelete(DeleteBehavior.Cascade);
```

# 1:0..1-Beziehungen

```
modelBuilder.Entity<TechEvent>()
    .HasOne(p => p.VenueSetup)
    .WithOne(p => p.TechEvent)
    .HasForeignKey<VenueSetup>(f => f.TechEventId)
    .IsRequired(false)
    .OnDelete(DeleteBehavior.Cascade);
```

# 1:1-Beziehungen (Table Splitting)

```
modelBuilder.Entity<TechEvent>()
    .HasOne(p => p.VenueSetup)
    .WithOne(p => p.TechEvent)
    .IsRequired(true)
    .OnDelete(DeleteBehavior.Cascade);
```

# n:m-Beziehung

```
modelBuilder.Entity<Product>()
    .HasMany(e => e.Details)
    .WithMany(e => e.Product)
    .UsingEntity<ProductDetails>(
        c=> c.HasOne(b => b.Detail).WithMany().HasForeignKey(b => b.ProductId),
        c=> c.HasOne(b => b.Product).WithMany().HasForeignKey(b => b.DetailId)
            .ToTable("ProductDetails")
            .HasKey(b => new {b.DetailId, b.ProductId});
```



**Demo**

# Arbeiten mit Entitäten



Keine Validierung!

# Einfügen

- Add() / AddRange()
- An Kontext anfügen (Attach)

```
// An den Kontext anfügen  
_efContext.Speakers.Add(speaker);  
// oder  
_efContext.Entry(speaker).State = EntityState.Added;  
// oder  
_efContext.Add(speaker);  
// Änderungen speichern (Keine Validierung!)  
_efContext.SaveChanges();
```

# Einfügen

## Eine Entität

```
SET NOCOUNT ON;
INSERT INTO [dnc].[Speakers] ([Homepage], [Name])
VALUES (@p0, @p1);
SELECT [Id], [Created], [Infos]
FROM [dnc].[Speakers]
WHERE @@ROWCOUNT = 1 AND [Id] = scope_identity();
```

## Mehrere Entitäten (Batch-Update)

```
SET NOCOUNT ON;
DECLARE @inserted0 TABLE ([Id] int, [_Position] [int]);
MERGE [dnc].[Speakers] USING (
VALUES (@p0, @p1, 0),
(@p2, @p3, 1),
(@p4, @p5, 2),
(@p6, @p7, 3),
(@p8, @p9, 4),
(@p10, @p11, 5),
(@p12, @p13, 6),
(@p14, @p15, 7),
(@p16, @p17, 8),
(@p18, @p19, 9)) AS i ([Homepage], [Name], _Position) ON 1=0
WHEN NOT MATCHED THEN
INSERT ([Homepage], [Name])
VALUES (i.[Homepage], i.[Name])
OUTPUT INSERTED.[Id], i._Position
INTO @inserted0;

SELECT [t].[Id], [t].[Created], [t].[Infos] FROM [dnc].[Speakers] t
INNER JOIN @inserted0 i ON ([t].[Id] = [i].[Id])
ORDER BY [i].[_Position];
```



**Demo**

# Abfragen

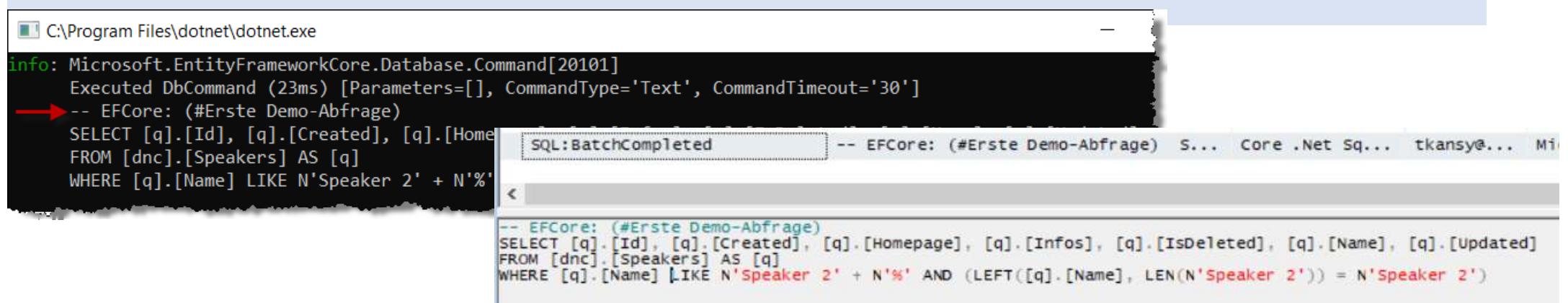
- LINQ to Entities
  - LINQ-Syntax
  - Methoden-Syntax
- Verzögerte Abfrage

```
var query1 = from q in _efContext.Speakers  
             where q.Name.StartsWith("Speaker 2")  
             select q;  
  
var query2 = _efContext.Speakers  
             .Where(q => q.Name == "Speaker 2");
```

# Query Tagging

- Versieht die Abfrage mit einem Tag („Bezeichnung“)
  - Vereinfacht die Suche nach der Abfrage im Log oder Profiler

```
var query1 = from q in _efContext.Speakers
              .TagWith("Erste Demo-Abfrage")
              where q.Name.StartsWith("Speaker 2")
              select q;
```



The screenshot shows a terminal window with the following output:

```
C:\Program Files\dotnet\dotnet.exe
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (23ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
→-- EFCore: (#Erste Demo-Abfrage)
SELECT [q].[Id], [q].[Created], [q].[Homepage], [q].[Infos], [q].[IsDeleted], [q].[Name], [q].[Updated]
FROM [dnc].[Speakers] AS [q]
WHERE [q].[Name] LIKE N'Speaker 2' + N'%'

SQL:BatchCompleted -- EFCore: (#Erste Demo-Abfrage) S... Core .Net Sq... tkansy@... Mi
-- EFCore: (#Erste Demo-Abfrage)
SELECT [q].[Id], [q].[Created], [q].[Homepage], [q].[Infos], [q].[IsDeleted], [q].[Name], [q].[Updated]
FROM [dnc].[speakers] AS [q]
WHERE [q].[Name] LIKE N'Speaker 2' + N'%' AND (LEFT([q].[Name], LEN(N'Speaker 2')) = N'Speaker 2')
```

# Projektion

- Es werden nur benötigte Spalten berücksichtigt

```
var query3 = (from q in _efContext.Speakers  
             where q.Name.StartsWith("Speaker 1")  
             select new { q.Id, q.Name, q.Homepage }  
           ).First();
```

```
SELECT TOP(1) [w].[Id], [w].[Created], [w].[Homepage]  
FROM [dnc].[Speakers] AS [w]  
WHERE [w].[Name] = @_speakerName_0
```

# Abfragen mit Variablen

- Abfrage leichter lesbar als EF 6.x

```
string speakerName = "Speaker 3";  
  
Speaker speakerByName =  
    _efContext.Speakers  
    .Where(w => w.Name == speakerName)  
    .First();
```

```
SELECT TOP(1) [w].[Id], [w].[Created], [w].[Homepage], [w].[Infos], [w].[Name]  
FROM [dnc].[Speakers] AS [w]  
WHERE [w].[Name] = @_speakerName_0
```

# Find()

- Zugriff via PrimaryKey
- 1st Level Cache

```
var query3 = _efContext.Speakers.Find(1);
```

# 1st Level Cache

- Serverzugriff vermeiden

```
// Nur den Cache verwenden  
... = _efContext.Speakers.Local.FirstOrDefault(w => w.Id == 25);  
  
// Cache und Server  
... = _efContext.Speakers.FirstOrDefault(w => w.Id == 25);
```



**Demo**

# Änderungen

Keine Validierung!

- Via Kontext abfragen
- An Kontext anfügen (Attach)
- Update pro Spalte

```
// Via Kontext abfragen
Speaker speaker1 = _efContext.Find<Speaker>(1);

// Änderung
speaker1.Name = "Name1";

// Speichern
_efContext.SaveChanges();
```

```
UPDATE [dnc].[Speakers] SET [Name] = @p0
WHERE [Id] = @p1;
SELECT @@ROWCOUNT;
```

# Änderungen

- Attach
- Entry<>().State festlegen

```
Speaker speaker2 = new Speaker() { Id = 0, Name = "..." };
_efContext.Attach(speaker2);
// Speichern => Insert
_efContext.SaveChanges();

Speaker speaker3 = new Speaker() { Id = 11, Name="...", Infos ="..." };
_efContext.Entry(speaker3).State = EntityState.Modified;
// Speichern => Update
_efContext.SaveChanges();
```



**Demo**

# Entität löschen

- **Remove() / RemoveRange()**
- Entry()

```
Speaker speaker1 = _efContext.Find<Speaker>(1);
// Löschen
_efContext.Remove(speaker1);
// oder
_efContext.Speakers.Remove(speaker1);
// Speichern
_efContext.SaveChanges();
```

```
SET NOCOUNT ON;
DELETE FROM [dnc].[Speakers]
WHERE [Id] = @p0;
SELECT @@ROWCOUNT;
```

# Entität löschen

- Remove() / RemoveRange()
- **Entry()**

```
Speaker speaker2 = new Speaker() { Id = 11 };
_efContext.Entry(speaker2).State = EntityState.Deleted;
// Speichern
_efContext.SaveChanges();
```

```
SET NOCOUNT ON;
DELETE FROM [dnc].[Speakers]
WHERE [Id] = @p0;
SELECT @@ROWCOUNT;
```



**Demo**

# Ausnahmen

SaveChanges() löst Ausnahmen aus wenn

Ausnahme	Grund
DbUpdateException	Werte können nichtpersistiert werden, weil z.B. Spalten zu „klein“ sind oder kein NULL zulassen
DbUpdateConcurrencyException	Die Daten wurden seit dem Laden in den Client auf Seiten der Datenbank verändert



# Graph Update

# Graph Update

Nicht lauffähig, eine Idee, siehe Code!

- Einfach nur Attach()
- ChangeTracker.TrackGraph()

```
_efContext.ChangeTracker.TrackGraph(techEvent, node =>
{
    PropertyEntry propertyEntry = node.Entry.Property("Id");

    if ((int)propertyEntry.CurrentValue == 0)
        entry.State = EntityState.Added;
    else if ((int)propertyEntry.CurrentValue < 0)
        entry.State = EntityState.Deleted;
    else
        entry.State = EntityState.Modified;
});
```



**Demo**



# Eigenschaft oder Feld?

# HasField (Fluent API)

- Verwendet ein Feld statt einer Eigenschaft
- Standard: Name des Feldes mit oder ohne `_` als Prefix

```
modelBuilder.Entity<Session>()
    .Property(p => p.Title)
    .HasField("_SessionTitle");
```

# UsePropertyAccessMode (Fluent API)

Steuert, wann eine Eigenschaft verwendet werden soll (statt des Feldes)

Field	Immer das Feld verwenden
FieldDuringConstruction	Feld während Materialisierung verwenden
Property	Immer die Eigenschaft verwenden

```
modelBuilder.Entity<Session>()
    .Property(p => p.Title)
    .HasField("_SessionTitle")
        .UsePropertyAccessMode(PropertyAccessMode.Property);
```



**Demo**

# Ladestrategien



# Ladestrategien

Strategie	
Lazy Loading (Automatic)	Bei Navigation wird die anhängende Entität geladen
Lazy Loading (Explicit)	Alle Entitäten explizit vor der Verwendung laden und im Cache halten
Preloading	Alle Entitäten vor der eigentlichen Verwendung laden
EagerLoading	Anhängige/ benötigte Entitäten per Join mit einem Zugriff laden

# Lazy Loading (Automatic)

Bei Navigation wird die anhängende Entität geladen

```
.AddDbContext<SamplesContext1>(  
    o => o.UseLazyLoadingProxies()  
        .UseSqlServer(config[ "ConnectionStrings:EFConnectionString" ]);  
  
Public virtual ICollection<SpeakerSession> SpeakerSessions { get; set; }
```

Install-package Microsoft.EntityFrameworkCore.Proxies

# Do it yourself LazyLoaderProxy

```
private Blog(ILazyLoader lazyLoader)
{
    LazyLoader = lazyLoader;
}

public ICollection<Post> Posts
{
    get => LazyLoader?.Load(this, ref _posts);
    set => _posts = value;
}
```

# Lazy Loading (Explicit)

Alle Entitäten explizit vor der Verwendung laden und im Cache halten

```
_efContext.Entry(speaker2).Collection(e => e.SpeakerSessions).Load();
```

```
.Info: Microsoft.EntityFrameworkCore.Database.Command[200101]
Executed DbCommand (21ms) [Parameters=@__get_Item_0='10', CommandType='Text', CommandTimeout='30']
SELECT TOP(1) [e].[Id], [e].[Created], [e].[Homepage], [e].[Infos], [e].[IsDeleted], [e].[Name], [e].[Updated]
FROM [dnc].[Speakers] AS [e]
WHERE [e].[Id] = @__get_Item_0
.Info: Microsoft.EntityFrameworkCore.Database.Command[200101]
Executed DbCommand (2ms) [Parameters=@__get_Item_0='10', CommandType='Text', CommandTimeout='30']
SELECT [e].[Id], [e].[Created], [e].[IsDeleted], [e].[SessionId], [e].[SpeakerId], [e].[Tag], [e].[Updated]
FROM [dnc].[SpeakerSessions] AS [e]
WHERE [e].[SpeakerId] = @__get_Item_0
```

```
_efContext.Entry(techEvent).Reference(e => e.VenueSetup).Load();
```

# Preloading

Alle Entitäten vor der eigentlichen Verwendung laden

```
// Alle Sprecher laden  
_efContext.Speakers.ToList();  
  
// Zugriff aus dem Speicher  
Speaker speaker3 = _efContext.Speakers.Find(speakerId);
```

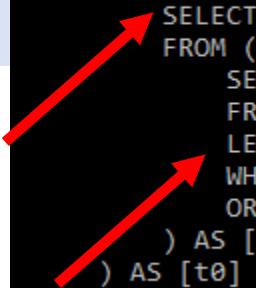
```
SELECT [s].[Id], [s].[Created], [s].[Homepage], [s].[Infos], [s].[IsDeleted], [s].[Name], [s].[Updated]  
FROM [dnc].[Speakers] AS [s]
```

# EagerLoading

Anhängende/ benötigte Entität per Join mit einem Zugriff laden

```
TechEvent techEvent4 = _efContext.TechEvents
    .Include(i => i.Sessions).ThenInclude(i => i.SpeakerSessions)
    .Include(i => i.VenueSetup)
    .SingleOrDefault(t => t.Id == techeventId);

    SELECT [s].[Id], [s].[ContentDescription], [s].[Begin], [s].[Created], [s].[Difficulty], [s].[Duration], [s].[End]
    , [s].[EventId], [s].[IsDeleted], [s].[SpeakerId], [s].[TechEventId], [s].[Title], [s].[Updated]
    FROM [dnc].[Sessions] AS [s]
    INNER JOIN (
        SELECT DISTINCT [t].*
        FROM (
            SELECT TOP(1) [i0].[Id]
            FROM [dnc].[TechEvents] AS [i0]
            LEFT JOIN [dnc].[VenueSetup] AS [i.VenueSetup0] ON [i0].[VenueSetupId] = [i.VenueSetup0].[Id]
            WHERE [i0].[Id] = @_techeventId_0
            ORDER BY [i0].[Id]
        ) AS [t]
    ) AS [t0] ON [s].[TechEventId] = [t0].[Id]
    WHERE [s].[IsDeleted] = 0
    ORDER BY [t0].[Id], [s].[Id]
```



# Split-queries

- Relationale Abfrage in einzelne Abfragen aufteilen (Keine Joins)
  - AsSplitQuery() in der Linq-Abfrage
  - Oder Context weit

```
// EF Context konfigurieren
iocContainer.AddDbContext<SamplesContext1>(o => o
    // .UseLazyLoadingProxies()
    .UseSqlServer(Configuration[".ConnectionStrings:EFConnectionString"], b =>
{
    b.UseQuerySplittingBehavior(QuerySplittingBehavior.SplitQuery);
})
.UseLoggerFactory(EFLoggerFactory.Instance)
.ConfigureWarnings(w =>
```



**Demo**



# Änderungen verfolgen



# Änderungen verfolgen

- Change Tracker
  - Änderungen verfolgen
  - HasChanges ()
  - 1th Level Cache
- AutoDetectChanges
  - Standard: true
- QueryTrackingBehavior
  - TrackAll (Standard)
  - NoTracking
- ChangeTrackingStrategy
- Zustand von Entitäten abfragen

Nicht mehr dringend notwendig

# AutoDetectChanges

- Standard: true
- Automatischer Aufruf von `ChangeTracker.DetectChanges()`

```
public override int SaveChanges()
{
    if (!ChangeTracker.AutoDetectChangesEnabled)
        ChangeTracker.DetectChanges();

    return base.SaveChanges();
}
```

# QueryTrackingBehavior

Bestimmt, ob die Entität vom Kontext überwacht werden soll

```
// Change Tracker - Alles tracken außer...
_efContext.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.TrackAll;
var queryNoTrack = from q in _efContext.Speakers.AsNoTracking()
                    where q.Name.StartsWith("Speaker")
                    select q;

// Change Tracker - Nichts tracken außer...
_efContext.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
var queryTrack = from q in _efContext.Speakers.AsTracking()
                    where q.Name.StartsWith("Speaker")
                    select q;
```

# HasChangeTrackingStrategy (Model/ Entity)

- Legt die Art der Änderungsverfolgung fest
- Standard: Snapshot

```
modelBuilder.HasChangeTrackingStrategy  
    (ChangeTrackingStrategy.ChangedNotifications);
```

```
modelBuilder.Entity<SessionRL>().HasChangeTrackingStrategy  
    (ChangeTrackingStrategy.ChangedNotifications);
```

# (Bekannter) Zustand einer Entität

```
_efContext.Entry(s).State = EntityState.Added;
```

Wert	
Detached (*)	Entität dem Kontext unbekannt
Unchanged (*)	Keine Änderung, keine Aktion für die Datenbank
Deleted (*)	Entität gelöscht => DELETE-Anweisung
Modified (*)	Entität verändert => UPDATE-Anweisung
Added (*)	Entität (neu) hinzugefügt => INSERT-Anweisung

(\*) Microsoft.EntityFrameworkCore

# Microsoft.EntityFrameworkCore.ChangeTrackingStrategy

Wert	Vorgehensweise
Snapshot	Werte werden beim Laden gespeichert und bei DetectChanges () mit aktuellen Werten verglichen
ChangedNotifications	Entität muss INotifyPropertyChanged (*) implementieren
ChangingAndChangedNotifications	Entität muss INotifyPropertyChanged (*) und INotifyPropertyChanging (*) implementieren
ChangingAndChangedNotifications-WithOriginalValues	Entität muss INotifyPropertyChanged (*) und INotifyPropertyChanging (*) implementieren

(\*) System.ComponentModel

# Änderungen analysieren

```
foreach (EntityEntry entry in _efContext.ChangeTracker.Entries())
{
    Console.WriteLine(entry.Entity);

    // Alle Eigenschaften durchlaufen
    foreach (PropertyEntry property in entry.Properties.Where(w => w.IsModified))
    {
        Console.WriteLine($"{property.Metadata.Name},
                        ClrType={property.Metadata.ClrType.Name},
                        Orginal='{property.OriginalValue}',
                        Current='{property.CurrentValue}',
                        IsModified={property.IsModified}");

        // Einige Informationen können verändert werden!
    }
}
```



**Demo**



# Direct SQL & Query Types

# Direct SQL

## Direkte SQL Statements ausführen

- DbSet<>.FromSqlRaw/FromSqlInterpolated
  - SQL in LINQ möglich
  - Nur Select
- ExecuteSqlCommand()
  - Insert, Delete, Update, jedes SQL Statement
- ADO.NET Core
  - Insert, Delete, Update, jedes SQL Statement.
  - Größte mögliche Kontrolle bei der Ausführung

# DbSet.FromSql()

Query via DbParameter-Implementierung mit Werten bestücken

```
// Parameter definieren
DbParameter searchTermPar =
    new SqlParameter("@SearchTerm", SqlDbType.VarChar, 50);
searchTermPar.Value = searchFor;

// Ausführen
List<Speaker> query1 = _efContext.Speakers
    .FromSqlRaw("EXEC dbo.usp_GetSpeaker @SearchTerm", searchTermPar)
    .ToList();
```

# DbSet.FromSqlRaw()

Query via String.Format mit Werten bestücken

```
// Ausführen  
List<Speaker> query2 = _efContext.Speakers  
    .FromSql("EXEC dbo.usp_GetSpeaker '{0}'", searchFor)  
    .ToList();
```

# DbSet.FromSqlInterpolated()

Query via String Interpolation mit Werten bestücken

```
// Ausführen
List<Speaker> query3 = _efContext.Speakers
    .FromSqlInterpolated($"EXEC dbo.usp_GetSpeaker '{searchFor}'")
    .ToList();
```

# ExecuteSqlCommand()

```
// Parameter definieren
SqlParameter id = new SqlParameter("@Id", 3);
SqlParameter infos = new SqlParameter("@Infos", "Neue Info");

// Ausführen
int rowsAffected =
    _efContext.Database.ExecuteSqlCommand(SQLSTATEMENT, id, infos);
```

# ADO.NET

```
// IDbCommand-Instanz vom Kontext geben lassen
using (IDbCommand cmd = _efContext.Database.GetDbConnection().CreateCommand())
{
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = COMMANDADONET;

    // Verbindung zur Datenbank öffnen
    _efContext.Database.OpenConnection();

    // Ausführen
    cmd.ExecuteNonQuery();
}
```



**Demo**

# Query Types

- Readonly-Abfragen-Typen
  - Kein (eindeutiger) Schlüssel wie bei Entität nötig
- Kein Caching

```
modelBuilder
    .Entity<SpeakerStatistics>().ToTable("vwSpeakerStats").HasNoKey()
    .Property(v => v.NumberOfSessions).HasColumnName("SessionCount");

public DbSet<SpeakerStatistics> SpeakerStatistics { get; set; }

var query = _efContext.SpeakerStatistics
    .OrderBy(o => o.NumerOfSessions);
```



**Demo**



# Concurrency

# (Optimistic) Concurrency

Überprüfung, ob sich zwischen dem Laden und Speichern Daten auf dem Server geändert haben

- Bei Update und Delete mitgeschickt
  - Entitätsschlüssel
  - Concurrency Tokens
  - Row Version

# IsConcurrencyToken (Fluent API)

- Weist die Eigenschaft als ConcurrencyToken aus
- Standard: false

```
modelBuilder.Entity<Session>()
    .Property(p => p.Abstract)
    .IsConcurrencyToken();
```

# IsRowVersion (Fluent API)

- Weist die Eigenschaft als RowVersion aus
  - IsConcurrencyToken + ValueGeneratedOnAddOrUpdate
- Standard: false

```
modelBuilder.Entity<Session>()
    .Property(p => p.Abstract)
    .IsRowVersion();
```

# DbUpdateConcurrencyException

- Concurrency Situation
- Lösungsmöglichkeiten
  - ClientWins
    - `OriginalValues.SetValues(e.GetDatabaseValues())`
  - StoreWins
    - `Reload()`
  - Custom
    - Eine Mischung aus den vorherigen Ansätzen



**Demo**

# Sequenzen



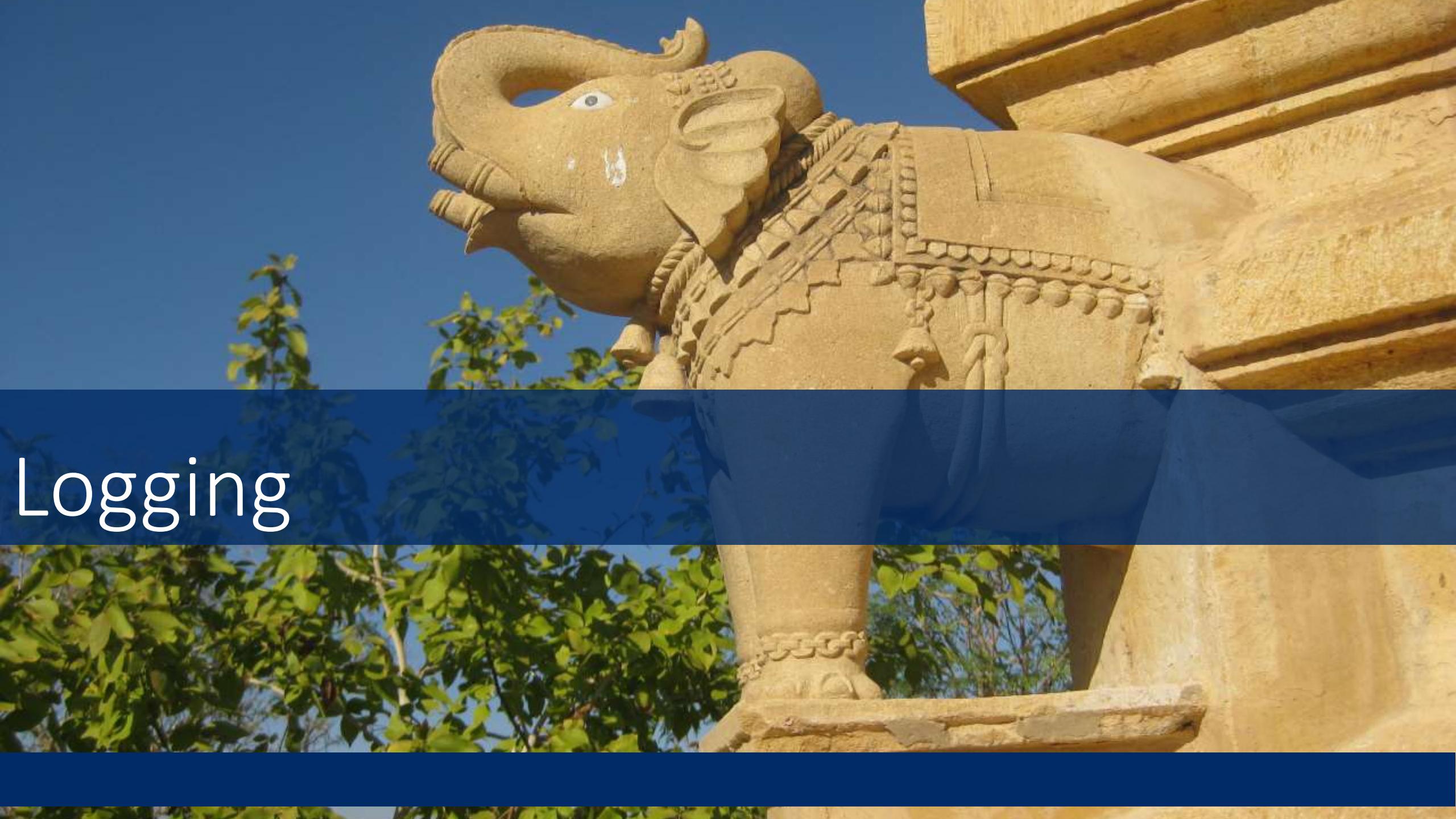
# Sequenzen (Fluent API)

- Verwendung von Datenbank Sequenzen
  - z.B. für Tabellen übergreifende, eindeutige Schlüssel
  - z.B. als Standardwert

```
modelBuilder.HasSequence<int>("Bestellung")  
    .StartsAt(1000)  
    .IncrementsBy(5)  
    .HasMin(1000)  
    .HasMax(100000)  
    .IsCyclic();
```



**Demo**



# Logging

# Logging

- Interne Events
  - QueryPossibleExceptionWithAggregateOperatorWarning
  - ConnectionError
  - ...
- Sensitive Daten
  - Parameter für Abfragen
  - Eigenschaften der gesamten Entität

# Interne Events

## Interne Ereignisse reagieren

```
optionsBuilder.ConfigureWarnings(s =>
{
    s.Throw(RelationalEventId.QueryPossibleExceptionWithAggregateOperatorWarning);
    // Oder
    s.Log(RelationalEventId.QueryPossibleExceptionWithAggregateOperatorWarning);
    // Oder
    s.Ignore(RelationalEventId.QueryPossibleExceptionWithAggregateOperatorWarning);
});
```

# EventIds

Microsoft.EntityFrameworkCore.Diagnostics.RelationalEventId

```
...public static class RelationalEventId
{
    ...public static readonly EventId ConnectionOpening;
    ...public static readonly EventId QueryPossibleExceptionWithAggregateOperator;
    ...public static readonly EventId QueryPossibleUnintendedUseOfEqualsWarning;
    ...public static readonly EventId QueryClientEvaluationWarning;
    ...public static readonly EventId MigrationsNotFound;
    ...public static readonly EventId MigrationsNotApplied;
    ...public static readonly EventId MigrationGeneratingUpScript;
    ...public static readonly EventId MigrationGeneratingDownScript;
    ...public static readonly EventId MigrationApplying;
    ...public static readonly EventId MigrationReverting;
    ...public static readonly EventId MigrateUsingConnection;
    ...public static readonly EventId DataReaderDisposing;
    ...public static readonly EventId AmbientTransactionWarning;
    ...public static readonly EventId TransactionError;
    ...public static readonly EventId TransactionDisposed;
    ...public static readonly EventId TransactionRolledBack;
    ...public static readonly EventId TransactionCommitted;
    ...public static readonly EventId TransactionUsed;
    ...public static readonly EventId TransactionStarted;
    ...public static readonly EventId CommandError;
    ...public static readonly EventId CommandExecuted;
    ...public static readonly EventId CommandExecuting;
    ...public static readonly EventId ConnectionError;
    ...public static readonly EventId ConnectionClosed;
    ...public static readonly EventId ConnectionClosing;
    ...public static readonly EventId ConnectionOpened;
    ...public static readonly EventId ModelValidationKeyDefaultValueWarning;
    ...public static readonly EventId BoolWithDefaultWarning;
```

# Konfiguration via IoC

```
.AddDbContext<SamplesContext1>(  
    o => o.UseSqlServer(config[".ConnectionStrings:EFConnectionString"])  
    .UseLoggerFactory(AppLoggerFactory.Instance)  
    .ConfigureWarnings(w =>  
        w.Throw(RelationalEventId.QueryPossibleExceptionWithAggregateOperatorWarning))  
    .ConfigureWarnings(w => w.Ignore(RelationalEventId.ConnectionError))  
    .EnableSensitiveDataLogging(true)  
)
```

# LoggerFactory als Singelton-Pattern

```
...
// Methode erzeugt Factory-Instanz (individuell für EF)
private static void SetupFactory(LogLevel minLevel)
{
    _loggerFactory = LoggerFactory.Create(builder =>
    {
        builder.AddConsole();
        builder.AddDebug();
        builder.SetMinimumLevel(minLevel);
    });
}
...
```

# LoggerFactory via IoC

```
...
// Methode entnimmt Factory-Instanz dem Serviceprovider
public static void SetupFactory(IServiceCollection serviceProvider)
{
    Instance = serviceProvider.GetService<ILoggerFactory>();
    ...
}
```



**Demo**



# Database Handling

# Database-API

- Transaktion
  - BeginTransaction()
  - CommitTransaction() & RollbackTransaction()
  - CreateSavepoint() & RollbackToSavepoint() 5.0
  - UseTransaction()
  - CurrentTransaction
  - AutoTransactionsEnabled
- Connections
  - OpenConnection()
  - CloseConnection()
  - GetDbConnection()

# Database-API

- Datenbank/ Migration
  - EnsureCreated()
  - EnsureDeleted()
  - Migrate()
  - GetPendingMigrations()



**Demo**

# Transaktionen



# Transaktionen

- Erreichbar über Database-API des Kontext
- SaveChanges () ist transaktionssicher
  - Wenn Database.AutoTransactionsEnabled = true
- SavePoints **5.0**
  - Database.CreateSavepoint ();
  - Database.RollbackToSavepoint ();
- Ambient Transactions **2.1**
  - Mehrere (Indirekte) Kontexte
  - ADO.NET Core Transaktionen
  - Andere Transaktionsfähige System

# Mehrere Kontexte & ADO.NET Core

## System.Transaction.TransactionScope

```
using (DbConnection con = new SqlConnection(_configuration["ConnectionStrings:EFConnectionString"]))
{
    con.Open();

    using (TransactionScope scope = new TransactionScope(TransactionScopeOption.Required)) {
        // _efContext1.Speakers.Remove(...);

        using (DbCommand cmd = con.CreateCommand()) {
            cmd.CommandType = CommandType.Text;
            cmd.CommandText = "...";
            cmd.ExecuteNonQuery();
        }

        // Gesamte Transaktion(en) freigeben
        scope.Complete();
    }
}
```



**Demo**

# Explizite Transaktion

System.Transactions.CommittableTransaction

```
using (CommittableTransaction scope = new CommittableTransaction()) {
    using (DbConnection con = new SqlConnection(_configuration["ConnectionStrings:EFConnectionString"])) {
        con.Open();

        _efContext1.Database.OpenConnection();
        _efContext1.Database.EnlistTransaction(scope);

        using (DbCommand cmd = con.CreateCommand()) {
            // Mit DbCommand arbeiten
            // Mit DbContext arbeiten
        }
    }
    scope.Commit();
}
```



**Demo**

# Value Generator

# Value Generator (Attribut)

In der Datenbank erzeugte Werte

- ValueGeneratedNever (auch für z.B. Standardwerte)
- ValueGeneratedOnAdd
- ValueGeneratedOnAddOrUpdate
- ValueGeneratedOnUpdate
- HasValueGenerator

# Value Generator (Fluent API)

- Festlegen, wann Werte aus der Datenbank nachgeladen werden müssen
- Standard: ValueGeneratedNever

```
modelBuilder.Entity<Session>()
    .Property(p => p.Duration)
    .ValueGeneratedNever()          // oder
    .ValueGeneratedOnAddOrUpdate() // oder
    .ValueGeneratedOnAdd()         // oder
    .ValueGeneratedOnUpdate();
```



**Demo**

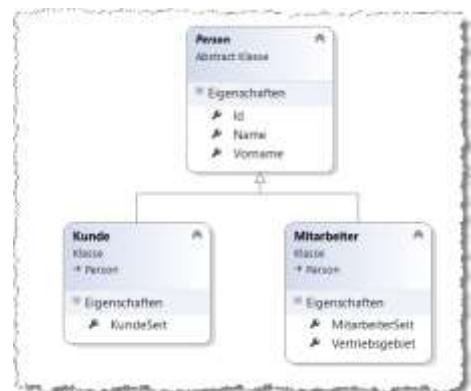


# Vererbungs-/ Mappingstrategien

# Vererbungsstrategie

- Table-per-hierarchy (TPH)

```
modelBuilder.Entity<Person>()
    .HasDiscriminator<string>("Discriminator")
    .HasValue<Kunde>("Kunde")
    .HasValue<Mitarbeiter>("Mitarbeiter");
```



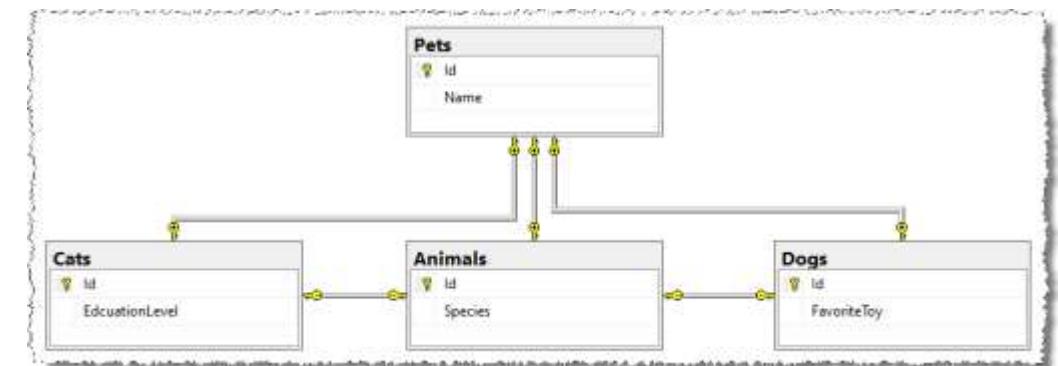
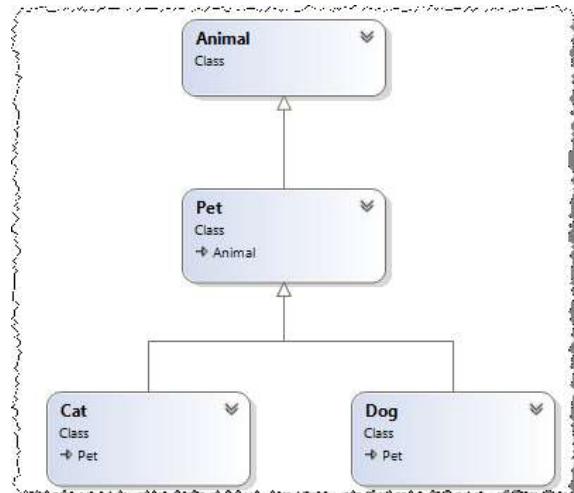


**Demo**

# Vererbungsstrategie

- Table-per-type (TPH)

```
modelBuilder.Entity<Animal>().ToTable("Animals");  
modelBuilder.Entity<Pet>().ToTable("Pets");  
modelBuilder.Entity<Cat>().ToTable("Cats");  
modelBuilder.Entity<Dog>().ToTable("Dogs");
```





**Demo**

# Indizes



# HasIndex (Fluent API)

Datenbank Indizes für schnellere (Lese-)Zugriffe

```
modelBuilder.Entity<Session>()
    .HasIndex(p => new { p.Created, p.Updated })
    .HasFilter("[Updated] IS NOT NULL")
    .IsUnique(true);
```



**Demo**



# Berechnete Spalten



# Berechnete Spalten (Fluent API)

## Datenbank berechnete Spalten

```
modelBuilder.Entity<Session>()
    .Property(s => s.Duration)
    .HasComputedColumnSql(
        "DATEDIFF(Second, [BeginTime], [EndTime]) / 60", stored: true);
```



**Demo**

The background of the slide features a wide-angle photograph of a mountainous landscape. In the foreground, there's a green grassy field with a small, low fence. Behind the fence is a dense forest of tall evergreen trees. Further back, majestic mountains rise, their peaks partially covered in snow. The sky is clear and blue. A branch with white flowers is visible in the top right corner, adding a sense of depth.

# Database Functions

# Database Funktionen (Vordefiniert)

## Microsoft.EntityFrameworkCore.EF

```
var query3 = _efContext.Speakers.Where(w => EF.Functions.Like(w.Name, "Speaker [1-3]"));
```

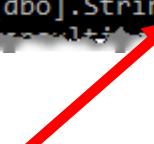
# Database Functions

- Verwendung (benutzerdefinierter) Funktionen

```
[DbFunction("StringLike", "dbo")]
public static bool StringLike(string @string, string Pattern)
{
    throw new Exception("No direct call");
}
```

```
var q = from te in _efContext.TechEvents
        where SamplesContext.StringLike(te.Name, "Ba%")
        select te;
```

```
SELECT [te].[Id], [te].[Begin], [te].[End], [te].[Image]
FROM [dnc].[TechEvents] AS [te]
WHERE [dbo].StringLike([te].[Name], N'Ba%') = 1
```



# Database Function (Fluent API)

## Datenbank-Funktion

```
modelBuilder.HasDbFunction(  
    GetType().GetMethod("dnc")),  
    o =>  
    {  
        o.HasName("StringLike");  
        o.HasSchema("dbo");  
    });
```



**Demo**



# Self-contained Type configuration

# Self-contained Type Configuration

- Trennung von Entitäten-Konfiguration und Kontext

```
public class SpeakerEFConfiguration : IEntityTypeConfiguration<Speaker>
{
    public void Configure(EntityTypeBuilder<Speaker> builder)
    {
        builder.Property(s => s.Infos)
            .IsRequired()
            .HasDefaultValue("(Keine Infos)");
        // ...
    }
}
// In OnModelCreating()
modelBuilder.ApplyConfiguration(new SpeakerEFConfiguration());
```



**Demo**



# Shadow Properties



# Shadow Properties

- Eigenschaften, die nicht in der Entität angelegt sind
  - Domain-Driven Design pattern

```
modelBuilder.Entity<TechEvent>()
    .Property<String>("Code")
    .HasColumnName("SecretCode");

Entry(techEvent1).Property("Code").CurrentValue

EF.Property<string>(w, "Code")
```



**Demo**



# Global Query Filter

# HasQueryFilter (Fluent API)

- Globale Einschränkung für alle Abfragen
  - Kann ignoriert werden (`IgnoreQueryFilters()`)
- Standard: –

```
modelBuilder.Entity<Session>()
    .HasQueryFilter(p => p.IsDeleted == false);
```

```
SELECT TOP(1) [s].[Id], [s].[ContentDescription], [s].[Begin], [s].[Created], [s].[Difficulty], [s].[Duration], [s].[End], [s].[EventId], [s].[IsDeleted], [s].[SpeakerId], [s].[TechEventId], [s].[Title], [s].[Updated]
FROM [dnc].[Sessions] AS [s]
WHERE ([s].[IsDeleted] = 0) AND ([s].[Id] = @_get_Item_0)
```





**Demo**

# Table-valued Functions (TVF)

# FromExpression (Fluent API)

- Mapping auf TVF, Type für
  - Rückgabe (Report)
  - Tabellenaufbau (Employee)

```
modelBuilder.Entity<Employee>();  
modelBuilder.Entity(typeof(Report)).HasNoKey();  
  
public IQueryable<Report> GetReports(int managerId)  
    => FromExpression(() => GetReports(managerId));
```



**Demo**

# Migration



# Migration

- Code First
  - Datenbank aus DbContext erzeugen
    - Add-Migration
    - Update-Database
- Database First (Reverse Engineering)
  - Scaffold-DbContext
- Bundle
  - Bundle-Migration

# Package Manager Console // Command line

- **Command line**
  - dotnet ef migrations add First
  - dotnet ef database update First
  - dotnet ef migrations bundle
  - dotnet ef migrations bundle
  - dotnet ef DbContext Scaffold
- **Package Manager Console**
  - Add-Migration First
  - Update-Database
  - Script-Migration
  - Bundle-Migration
  - Scaffold-DbContext

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

# Migration- Code First

- **Migration erstellen**
  - Add-Migration First -project dotnetconsulting.Samples.EFContext
- **Datenbank erstellen/ anpassen**
  - Update-Database First -project dotnetconsulting.Samples.EFContext
  - Update-Database 0 -project dotnetconsulting.Samples.EFContext
- **Migration skripten**
  - Script-Migration -project dotnetconsulting.Samples.EFContext -Idempotent

# Migration- Code First

- Migration erstellen
  - dotnet.exe ef migrations add First
- Datenbank erstellen/ anpassen
  - dotnet.exe ef database update First
- Migration skripten
  - dotnet.exe ef migrations script



**Demo**



Nachträgliche  
Änderungen nur manuell

# Migration- Database First

- DbContext erstellen

- Scaffold-DbContext **-Connection**

```
"Server=.;Database=<DbName>;Trusted_Connection=True;  
MultipleActiveResultSets=True;" -Provider  
Microsoft.EntityFrameworkCore.SqlServer -project  
dotnetconsulting.Samples.EFDatabaseFirst
```

```
PM> Scaffold-DbContext -Connection "Server=.;Database=TechEvents;Trusted_Connection=True;MultipleActiveResultSets=True;" -P  
Unable to find a table in the database matching the selected table dbo.sysdiagrams.  
Unable to find a table in the database matching the selected table dbo.sysdiagrams.  
Unable to find a table in the database matching the selected table dbo.sysdiagrams.  
Unable to find a table in the database matching the selected table dbo.sysdiagrams.  
Unable to find a table in the database matching the selected table dbo.sysdiagrams.  
For index PK_sysdiagr_C2B05B61D862A0D2. Unable to find parent table dbo.sysdiagrams. Skipping index.  
For index UK_principal_name. Unable to find parent table dbo.sysdiagrams. Skipping index.  
For index UK_principal_name. Unable to find parent table dbo.sysdiagrams. Skipping index.
```

# Scaffold-DbContext

```
Scaffold-DbContext
```

```
-Connection  
"Server=(localdb)\mssqllocaldb;Database=AdventureWorks;Trusted_Connection=True;MultipleActiveResultSets=True;"  
-Provider Microsoft.EntityFrameworkCore.SqlServer  
-Project Basf.EFCoreDemo.AdventureWorks  
-Folder Repository  
-Force
```

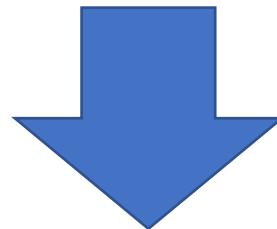


**Demo**

# Migration Bundles

## Migration via DevOps freundlicher EXE

```
dotnet ef migrations bundle
```



```
efbundle.exe
```

oder

```
efbundle.exe --connection { $ENVVARWITHCONNECTION }
```



**Demo**

# Data Seeding (Fluent API)

- Daten bei Migration mit Daten befüllen
  - Insert, Delete & Update werden erkannt

```
modelBuilder.Entity<Post>().HasData(  
    new { BlogId = 1, PostId = 1, Title = "P1", Content = "T1"},  
    new { BlogId = 1, PostId = 2, Title = "P2", Content = "T2"});
```

```
efContext.Database.EnsureCreated();
```



**Demo**

# Compiled Models



# Compiled Models

Geschwindigkeitsgewinn durch Vorkompilierung des Models zur Entwurfszeit

```
dotnet ef dbcontext optimize --output-dir  
MyCompiledModels --namespace MyCompiledModels
```

```
optimize-dbcontext -outputdir MyCompiledModels -  
namespace MyCompiledModels
```

- Keine Global Query Filter
- Kein LazyLoading (da kein Proxy Change-Tracking)

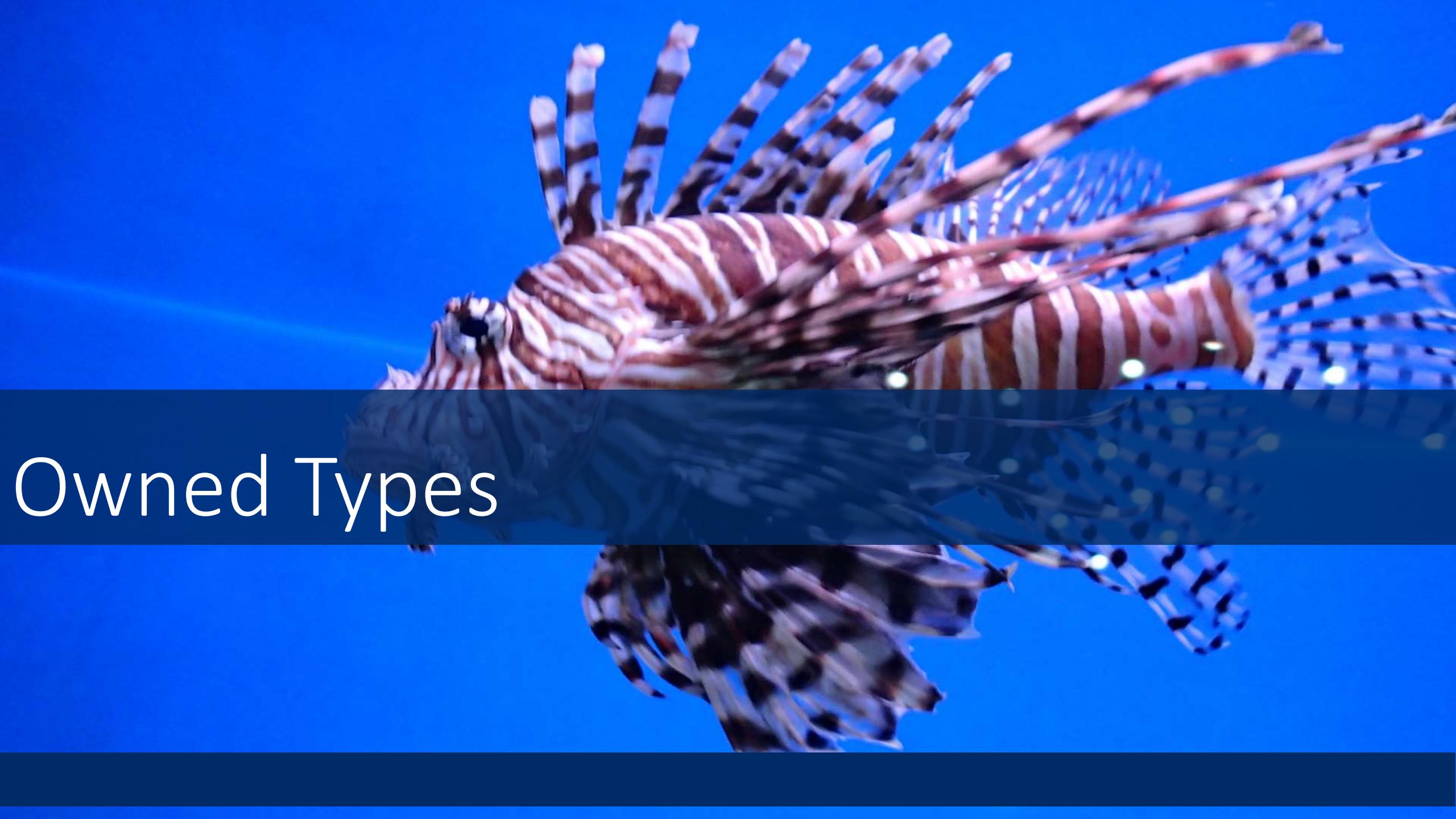
# Compiled model bootstrapping

## Model einbinden

```
...  
optionsBuilder.UseSqlServer(„<connection string>“);  
optionsBuilder.UseModel(MyCompiledModels.DemoDbContextModel.Instance);  
...
```



**Demo**



# Owned Types

# Owned Types

Aka Complex Types (Nested Documents bei NoSQL)

Owned-Attribute 2.1

```
C#  
[Owned]  
public class StreetAddress  
{  
    public string Street { get; set; }  
    public string City { get; set; }  
}  
  
public class Order  
{  
    public int Id { get; set; }  
    public StreetAddress ShippingAddress { get; set; }  
}
```

# Owned Types (n-Elemente)

```
modelBuilder.Entity<Book>()
    .OwnsMany<Chapter>("Chapters")
    .WithOwner();

modelBuilder.Entity<Book>()
    .OwnsOne<Author>("Author")
    .WithOwner();
```



A screenshot of a JSON object representing a Book entity with its Chapters. The JSON structure is as follows:

```
{
  "id": "f7949954-f258-44b1-b9fe-1e1fc2c85d17",
  "Discriminator": "Book",
  "Title": "How to EF Core",
  "Author": {
    "Name": "Thorsten Kansy",
    "id": "00000000-0000-0000-0000-000000000000"
  },
  "Chapters": [
    {
      "id": "1ffe825-e4fd-4d81-ba1f-e733d31e9fe1",
      "Content": "...",
      "Index": 1,
      "Title": "Chapter #1"
    },
    {
      "id": "563e9c0a-43cc-4697-8714-136317913980",
      "Content": "...",
      "Index": 2,
      "Title": "Chapter #2"
    }
  ]
}
```



**Demo**



# Interceptors

# Interceptors

- Interne EF Core Aktionen abfragen, protokollieren und verändern
  - DbConnectionInterceptor
  - DbTransactionInterceptor
  - DbCommandInterceptor

(Microsoft.EntityFrameworkCore.Diagnostics)

```
...
optionsBuilder.UseSqlServer(CONSTRING);
optionsBuilder.AddInterceptors(new QueryHintInterceptor());
...
```



**Demo**



# Spatial Extentions

# Spatial Extentions

## Räumliche Datentypen

### NetTopologySuite.Geometries-Suite

- Geometry (IGeometry)
- Point (IPoint)
- Polygon (IPolygon)
- SQL Server, In-Memory, SQLite, ...

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite
```

# SQL Server

- GEOMETRY / GEOGRAPHY

Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Position	geometry	<input checked="" type="checkbox"/>
Temperatur	float	<input type="checkbox"/>
Zeitpunkt	datetime	<input type="checkbox"/>

	Results	Messages		
	Id	Position	Temperatur	Zeitpunkt
1	1	0x000000000010C0000000000000000184000000000000002240	81.61	2018-09-18 15:09:55.673
2	2	0x000000000010C0000000000000000104000000000000000040	55.96	2018-09-18 15:09:56.363
3	3	0x000000000010C0000000000000000144000000000000001440	49.68	2018-09-18 15:09:56.363
4	4	0x000000000010C000000000000000018400000000000000040	85.02	2018-09-18 15:09:56.363
5	5	0x000000000010C0000000000000000F03F0000000000001840	80.32	2018-09-18 15:09:56.363
6	6	0x000000000010C000000000000000084000000000000001440	22.61	2018-09-18 15:09:56.363
7	7	0x000000000010C00000000000000001C400000000000000F03F	77.15	2018-09-18 15:09:56.363
8	8	0x000000000010C000000000000000018400000000000000040	50.70	2018-09-18 15:09:56.363
9	9	0x000000000010C00000000000000001C400000000000000F03F	31.94	2018-09-18 15:09:56.363

```
exec sp_executesql N'SELECT [m].[Id], [m].[Position], [m].[Temperatur], [m].[Zeitpunkt]
FROM [Messpunkte] AS [m]
WHERE [m].[Position].STDistance(@__mittelpunkt_0) < 4.0E0
ORDER BY [m].[Position].STDistance(@__mittelpunkt_0) DESC',N'@__mittelpunkt_0 varbinary(22)',@__mittelpunkt_0=0x000000000010C0000000000000000144000000000000001440
```



**Demo**



# Explicitly compiled Queries

# Explicitly compiled Queries

## Vorcompilierte LINQ-Abfragen

```
static Func<SamplesContext1, int, Session> sessionById =  
    EF.CompileQuery((SamplesContext1 ctx, int id) =>  
        ctx.Sessions.SingleOrDefault(s => s.Id == id));
```

```
Session session1 = sessionById(_efContext, 10);
```



**Demo**

# Compiled Models

Vorcompilierte Models => Performance Boost

Nicht unterstützt:

- Global query filters
- Lazy loading proxies
- Change tracking proxies

```
DbContextOptions<Context>.UseModel(...)
```

```
dotnet ef dbcontext optimize --context SamplesContext1
```



**Demo**



# DbContext Pooling & Factory

# DbContext Pooling

Pooling von DbContext Instanzen ähnlich Ado.NET Connection Pooling

```
public class SamplesContext3 : DbContext
{
    public SamplesContext3(DbContextOptions options):
        base(options)
    {}

    .AddDbContextPool<SamplesContext3>(
        o => o.UseSqlServer(...)
```



**Demo**

# DbContext Factory

## DbContext Factory via DI bereitstellen

```
services.AddDbContextFactory<SomeDbContext>(b => b.UseSqlServer(@"..."));

public class Controller
{
    public Controller(IDbContextFactory<SomeDbContext> factory) {}
}

using (var context = _contextFactory.CreateDbContext())
{
    // Context Instanz ist nicht gemanaged!
}
```



**Demo**



# Unit Tests

# Unit Tests

EF Core goes Memory ;-)

Achtung bei direkten Datenbank Funktionalitäten!

```
[TestMethod]  
public void TestMethod1()  
{  
    var options = new DbContextOptionsBuilder<SamplesContext1>()  
        .UseInMemoryDatabase(databaseName: "UnitTest")  
        .Options;  
  
    using (SamplesContext1 context = new SamplesContext1(options))  
    { ... }  
}
```



**Demo**

# Fragen? Jetzt oder später!



[www.dotnetconsulting.eu](http://www.dotnetconsulting.eu)



# Links



<http://dotnetconsulting.eu/blog/>



@Tkansy



tkansy@dotnetconsulting.eu



[www.dotnetconsulting.eu](http://www.dotnetconsulting.eu)