Online on-demand Project Support

EF Core 2.2

Grundlagen

Thorsten Kansy (tkansy@dotnetconsulting.eu)

Thorsten Kansy

Freier Consultant, Software Architekt,
Entwickler, Trainer & Fachautor

DOTNETCONSULTING
by Thorsten Kansy

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
- Vererbungsstrategie
- Indizes
- Berechnete Spalten
- Database Functions
- Self-contained Type configuration
- Shadow State
- Global Query Filter
- Uvm.

Neue Features von EF Core 2.2

Neue Features von EF Core 2.2

- Query Tagging
- Collection für Owned Entities
- Spatial Extentions
- Comos DB Provider (SQL-API)

```
https://blogs.msdn.microsoft.com/dotnet/2018/10/17/announcing-entity-framework-core-2-2-preview-3/
```

Grundlagen

Unterstützte Database-Provider

Relationale Datenbanken	NoSQL Datenbanken
Microsoft SQL Server	Azure Cosmos DB >=2.2
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 `DbContextOptionBuilder`

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

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:EFConString"])
        .ConfigureWarnings(w => w.Throw(RelationalEventId.QueryClientEvaluationWarning)));

    // Rückgabe
    IServiceProvider serviceProvider = diContainer.BuildServiceProvider();
    return serviceProvider;
}
```


Demo

Entity Types (Model)

TU2

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

Slide 21

TU2 Funktioniert mit Preview 1 nicht

Thorsten Ursula, 3/14/2018

Entity Model erzeugen/ verfeinern

- Via Attribute

- Fluent API

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

Entity Model 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");
```

Ignore (Fluent API)

- Ignoriert einen Typen komplett
- Standard: -

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

ToTable (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");
```

Ignore (Fluent API)

- Ignoriert die Eigenschaft einer Entität
- Standard: –

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

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 (Fluent API)

- Datenbanktyp
- Standard: Abhängig vom CLR-Datentyp

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

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();
```

HasMaxLenth (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);
```

HasConversion (Fluent API)

2.1

- 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

2.1

`Microsoft.EntityFrameworkCore.Storage.Converters`

- `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

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

- Nur Marke „Eigenbau“
- Beide Seiten nach folgendem Schema aufbauen

```
modelBuilder.Entity<Product>()
        .HasOne(e => e.Details)
        .WithOne(e => e.Product)
        .HasForeignKey<ProductDetails>(e => e.Id);

modelBuilder.Entity<Product>().ToTable("Products");

modelBuilder.Entity<ProductDetails>().ToTable("ProductDetails");
```


Demo

Arbeiten mit Entitäten

Einfügen

Keine Validierung!

- 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

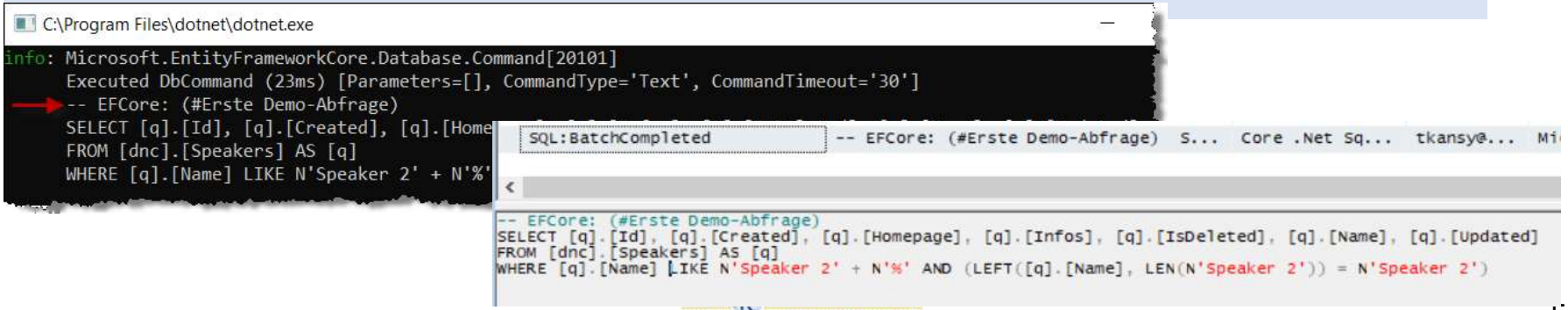
```
var query1 = from q in _efContext.Speakers
             where q.Name.StartsWith("Speaker 2")
             select q;

var query2 = _efContext.Speakers
                       .Where(q => q.Name == "Speaker 2");
```

Query Tagging

2.2

- 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;
```



Projektion

- Es werden nur benötigte Spalten berücksichtigt

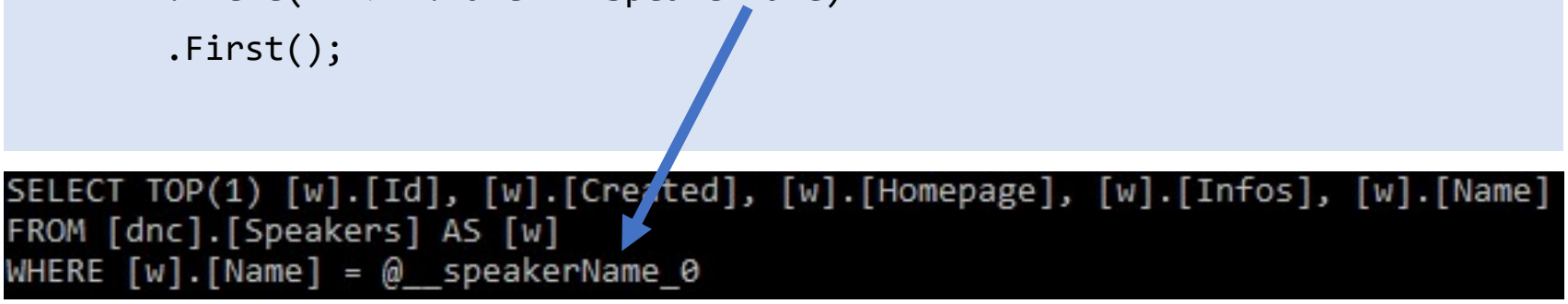
```
var query3 = (from q in _efContext.Speakers
              where q.Name.StartsWith("Speaker 1")
              select new { q.Id, q.Name, q.Homepage }
             ).First();
```

```
SELECT TOP(1) [w].[Id], [w].[Created], [w].[Homepage], [w].[Infos], [w].[Name]
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
_efContext.SaveChanges();

Speaker speaker3 = new Speaker() { Id = 11, Name="...", Infos ="..." };
_efContext.Entry(speaker3).State = EntityState.Modified;
// Speichern
_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 nicht persistiert 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)

2.1

Bei Navigation wird die anhängende Entität geladen

```
.AddDbContext<SamplesContext1>(
    o => o.UseLazyLoadingProxies()
          .UseSqlServer(config["ConnectionStrings:EFConString"]);

Public virtual ICollection<SpeakerSession> SpeakerSessions { get; set; }
```

```
Install-package Microsoft.Azure.DocumentDB
```

```
Install-package Microsoft.Azure.DocumentDB.Core
```

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

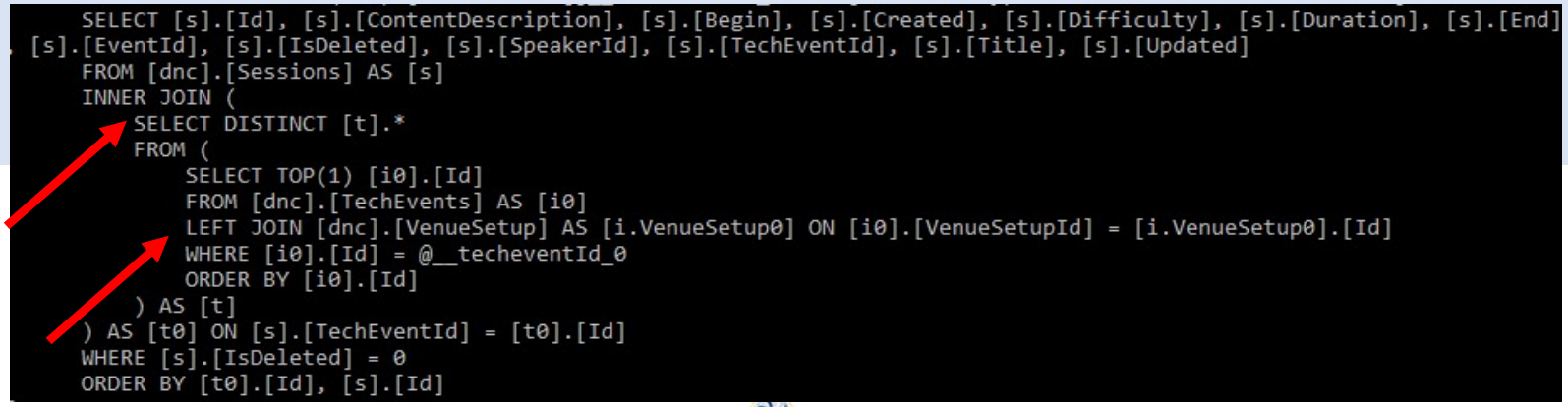
```
// Alle Sprecher laden
_efContext.Speakers.ToList();

// 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]
```


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

AutoDetectChanges

Nicht mehr dringend notwendig

- 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;
```

(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

HasChangeTrackingStrategy (Model/ Entity)

- Legt die Art der Änderungsverfolgung fest
- Standard: Snapshot

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


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

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<>.FromSql`
  - 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()

```
// Parameter definieren
SqlParameter searchTerm =
        new SqlParameter("@SearchTerm", SqlDbType.VarChar, 50);
searchTerm.Value = "*dolo*";

// Ausführen
List<Speaker> query1 = _efContext.Speakers
    .FromSql("EXEC dbo.usp_GetSpeaker @SearchTerm", searchTerm)
    .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

2.1

- Readonly-Abfragen-Typen
  - Kein Eindeutiger Schlüssel wie bei Entität nötig
- Kein Caching

```
modelBuilder
    .Query<SpeakerStatistics>().ToView("vwSpeakerStats")
    .Property(v => v.NumberOfSessions).HasColumnName("SessionCount");


public DbQuery<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
  - `QueryClientEvaluationWarning`
  - `ConnectionError`
  - …
- Sensitive Daten
  - Parameter für Abfragen
  - Eigenschaften der gesamten Entität

Interne Events

- Interne Ereignisse reagieren

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

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:EFConString"])
    .ConfigureWarnings(w => w.Throw(RelationalEventId.QueryClientEvaluationWarning))
    .ConfigureWarnings(w => w.Ignore(RelationalEventId.ConnectionError))
    .EnableSensitiveDataLogging(true)
)
```


Demo

Database Handling

Database Fassade

- Transaktion
  - `BeginTransaction()`
  - `CommitTransaction()`
  - `RollbackTransaction()`
  - `UseTransaction()`
  - `CurrentTransaction`
  - `AutoTransactionsEnabled`
- Connections
  - `OpenConnection()`
  - `CloseConnection()`
  - `GetDbConnection()`

Database Fassade

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


Demo

Transaktionen

Transaktionen

- Erreichbar über Database-API des Kontext
- `SaveChanges()` ist transaktionssicher
  - Wenn `Database.AutoTransactionsEnabled = true`

- 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:EFConString"]))
{
    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:EFConString"])) {
        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

Migration

Migration

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

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

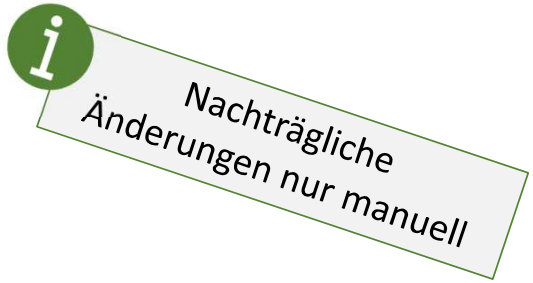
https://docs.microsoft.com/en-us/ef/core/miscellaneous/cli/powershell

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`


Demo

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=Tr
ue;MultipleActiveResultSets=True;"

-Provider Microsoft.EntityFrameworkCore.SqlServer

–project Basf.EFCoreDemo.AdventureWorks

-Folder Repository

-Froce
```

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

Data Seeding (Fluent API)

2.1

- 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();
```

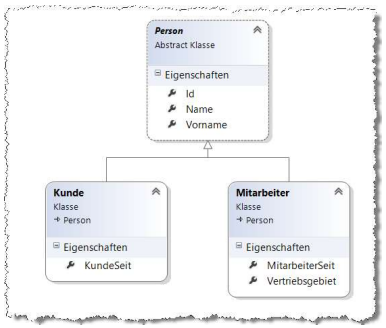

Demo

Vererbungsstrategie

Vererbungsstrategie

- Table-per-hierarchy (TPH)

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




Demo

Indizies

HasIndex (Fluent API)

Datenbank Indizies 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");
```


Demo

Database Functions

Database Funktionen (Vordefiniert)

Microsoft.EntityFrameworkCore.EF

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

Database Functions

2.0

- 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
```

DOTNETCONSULTING
by Thorsten Kansy

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

2.0

- 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 State

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

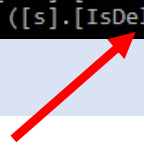
HasQueryFilter (Fluent API)

2.0

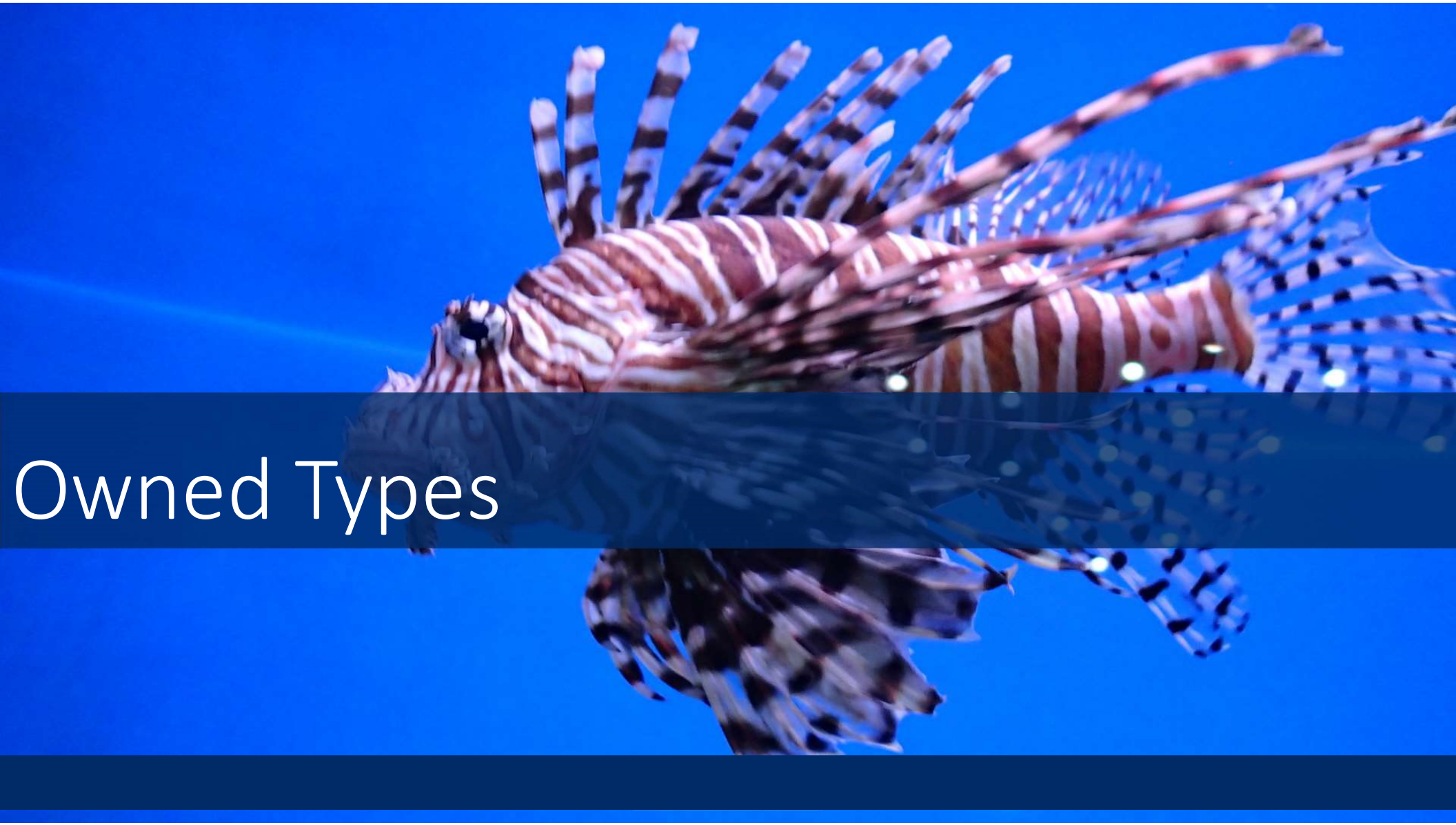
- Globale Einschränkung für alle Abfragen
  - Kann ignoriert werden (`IgnoreQueryFilters()`)
- Standard: –

```
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



Owned Types

Owned Types

2.0

Aka Complex Types

Owned-Attribut 2.1

C# Copy

```
modelBuilder.Entity<Order>().OwnsOne(p => p.OrderDetails, cb =>
    {
        cb.OwnsOne(c => c.BillingAddress);
        cb.OwnsOne(c => c.ShippingAddress);
    });

public class Order
{
    public int Id { get; set; }
    public OrderDetails OrderDetails { get; set; }
}

public class OrderDetails
{
    public StreetAddress BillingAddress { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}

public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}
```

C# Copy

2.1

```
[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 (Collection)

2.2

- Cosmos DB Provider => nested Documents

```
modelBuilder.Entity<Session>()
    .OwnsMany(p => p.Assessment);
```


Demo

Spatial Extentions

Spatial Extentions

2.2

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	☐
▶	Position	geometry	☐
	Temperatur	float	☐
	Zeitpunkt	datetime	☐
			☐

Results | Messages

	Id	Position	Temperatur	Zeitpunkt
1	1	0x00000000010C0000000000001840000000000002240	81.61	2018-09-18 15:09:55.673
2	2	0x00000000010C0000000000001040000000000000040	55.96	2018-09-18 15:09:56.363
3	3	0x00000000010C0000000000001440000000000001440	49.68	2018-09-18 15:09:56.363
4	4	0x00000000010C0000000000001840000000000000040	85.02	2018-09-18 15:09:56.363
5	5	0x00000000010C000000000000F03F0000000000001840	80.32	2018-09-18 15:09:56.363
6	6	0x00000000010C0000000000008400000000000001440	22.61	2018-09-18 15:09:56.363
7	7	0x00000000010C000000000001C400000000000F03F	77.15	2018-09-18 15:09:56.363
8	8	0x00000000010C0000000000001840000000000000040	50.70	2018-09-18 15:09:56.363
9	9	0x00000000010C000000000001C400000000000F03F	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=0x00000000010C0000000000001440000000000001440
```


Demo

Explicitly compiled Queries

Explicitly compiled Queries

2.0

Vorcompilierte LINQ-Abfragen

```
private 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

DbContext Pooling

DbContext Pooling

2.0

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

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?

Links

http://dotnetconsulting.eu/blog/

@Tkansy

tkansy@dotnetconsulting.eu

www.dotnetconsulting.eu

DOTNETCONSULTING
by Thorsten Kansy