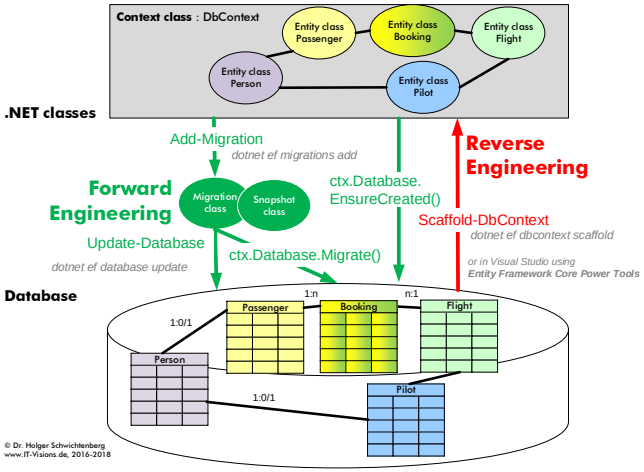
Spickzettel („Cheat Sheet"): Entity Framework Core: Modellierung

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

www.IT-Visions.de
Dr. Holger Schwichtenberg

Vorgehensmodelle und Artefakte



Reverse Engineering (Database First)

Package Manager Console (PMC) in Visual Studio:
Install-package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Design
Code für alle Datenbanktabellen generieren
Scaffold-DbContext -Connection "Server=Server123;Database=WWWings;Trusted_Connection=True;MultipleActiveResultSets=True;" **-Provider** Microsoft.EntityFrameworkCore.SqlServer
Code für Tabellen aus zwei Schemata generieren, mit Überschreiben vorhandener Dateien
Scaffold-DbContext -Connection "Server=Server123;Database=WWWings;Trusted_Connection=True;MultipleActiveResultSets=True;" **-Provider** Microsoft.EntityFrameworkCore.SqlServer **-Schema** Operation,Person **-fForce**
Code für ausgewählte Tabellen generieren, ausführliche Ausgabe, Überschreiben vorhandener Dateien, verwenden von DataAnnotations (wo möglich)
Scaffold-DbContext -Connection "Server=Server123;Database=WWWings;Trusted_Connection=True;MultipleActiveResultSets=True;" **-Provider** Microsoft.EntityFrameworkCore.SqlServer **-DataAnnotations -Tables** Flight,Passenger,Booking **-Verbose -Force**

Alternative als Visual Studio-Erweiterung: Entity Framework Core Power Tools
https://marketplace.visualstudio.com/items/ErikEJ.EFCorePowerTools
Generierung von Code für Datenbankviews bislang mit Entity Developer (kostenpflichtig) möglich: https://www.devart.com/entitydeveloper

Forward Engineering (Object Model First)

1. Erstellen der Entitätsklassen
2. Erstellen der Kontextklasse
3. PMC: Install-Package Microsoft.EntityFrameworkCore.Tools
4. PMC: Add-Migration sinnvollername (mehrfach möglich)
5. PMC: Update-Database
6. oder: Script-Migration mit Parameterangabe -from und -to

Datenbankschemamigrationen (PMC-Befehle)

```
Add-Migration v1 # Erste Schemaversion
Add-Migration v2 # Zweite Version usw.
Update-Database # Datenbank auf aktuellen Stand bringen
Update-Database -migration v1 # Datenbank auf alten Stand zurück
Script-Migration -from v2 -to v3 # SQL-Skript für die Änderungen erzeugen
Remove-Migration # letzten Migrationsschritt löschen
```

Erweiterung der erzeugten Migrationsklassen

```
public partial class v4 : Migration
 {
  protected override void Up(MigrationBuilder migrationBuilder)
  {  // Execute SQL script that is an embedded Ressource
migrationBuilder.Sql(Properties.Resources.CreateSP);  }
  protected override void Down(MigrationBuilder migrationBuilder)
  {  // Execute SQL command
   migrationBuilder.Sql("Drop Procedure dbo.[GetFlightsFromSP]");
  }
 }
```

EF Core erzeugt beim Umbenennen von Entitätsklassen ein DropTable() und CreateTable(). Dies muss man ersetzen durch migrationBuilder.RenameTable()

Datenbankerstellung zur Laufzeit

```
using (var ctx = new WWWingsContext())
{  ctx.Database.Migrate();  }
```

Namensräume für Entitätsklassen

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

Datenannotationen

[Table("name", "schema")] class xy { }	Vergabe Tabellen- bzw. Viewname und Datenbankschemaname
[Column("name", Order = 1, TypeName="...")]	Spaltenname, Spaltenreihenfolge und Spaltendatentyp
[Key]	Der Primärschlüssel der Klasse
[Required]	Pflichtfeld (Non-Nullable)
[DatabaseGenerated (option)]	Identity (Autowert), Computed (berechnete Spalte), None (normale Datenspalte)
[StringLength(Zahl)] oder [MaxLength(Zahl)]	Festlegung der Maximallänge der Zeichenkette
[ForeignKey(Name)]	Festlegung des zu einer Navigationseigenschaft gehörenden Fremdschlüssels
[InverseProperty(Name)]	Festlegung des zu einer Navigationseigenschaft gehörenden Navigationseigenschaft im anderen Objekt
[Timestamp]	Rowversion-Spalte zur Konflikterkennung
[ConcurrencyCheck]	Spalte wird zur Feststellung von Änderungskonflikten in die Where-Bedingung bei einem UPDATE oder DELETE aufgenommen.
[NotMapped]	Property wird nicht auf eine Datenbankspalte abgebildet.

Abstrakte Entitätsbasisklasse "Person"

PMC: install-package System.ComponentModel.Annotations

```
public abstract class Person
{
 #region Primitive properties
 // --- Primary key by convention if named ID or classnameID
 public int PersonID { get; set; }
 // --- Additional properties
 [StringLength(100)] // or [MaxLength(100)]
 public string Name { get; set; }
 // Use different database column name and datatype, default is datetime2(7)
 [Column("DayOfBirth", TypeName = "date")]
 public DateTime? Birthday { get; set; } // or: Nullable<DateTime>
 #endregion

 #region Members not mapped to the database
 // Calculated property (no setter, i.e. in RAM only)
 public string FullName => this.GivenName + " " + this.Surname;
 [NotMapped]
 public Guid Guid { get; set; } = Guid.NewGuid();
 // Methods are not mapped to the database!
 public override string ToString()
 {  return "#" + this.PersonID + ": " + this.FullName;  }
 #endregion
}
```

Erbende Entitätsklassen "Passenger" und "Pilot"

Virtual bei Navigation Properties nur notwendig, wenn Lazy Loading verwendet werden soll!

```
public partial class Passenger : Person
 {
  public Passenger()
  {
 // Collection creation possible, but not nessesary
  this.BookingSet = new List<Booking>();  // or HashSet<T> etc.
  }
  // Primary key is inherited!
  #region Primitive Properties
  public DateTime? CustomerSince { get; set; } //or: Nullable<DateTime>
  public string FrequentFlyer { get; set; }
  [StringLength(1)] public char Status { get; set; }
  #endregion
  #region Related Objects (Navigation Properties)
  public virtual ICollection<Booking> BookingSet { get; set; }
  #endregion
 }

 public partial class Pilot : Person
 {
  public virtual PilotLicenseType? PilotLicenseType { get; set; }
  #region Related Objects (Navigation Properties)
  public virtual ICollection<Flight> FlightAsPilotSet { get; set; }
  public virtual ICollection<Flight> FlightAsCopilotSet { get; set; }
  #endregion
 }
```

Spickzettel („Cheat Sheet“): Entity Framework Core: Modellierung

Autor: Dr. Holger Schwichtenberg (www.IT-Visions.de)

Enumeration "PilotLicenceType"

```
public enum PilotLicenseType
 {
  Student, Sport, Recreational, Private, Commercial, FlightInstructor, ATP
 }
```

Entitätsklasse "Flight"

```
[Table("Flight", Schema = "Operation")] // set DB table name and schema
public class Flight
 {
  public Flight()
  {
   // Default Values
   this.Price = 123.45m;
   // Possible, but not nessesary
   this.BookingSet = new List<Booking>(); // or HashSet<T> etc.
  }

  #region Key, not an identity column!
  [Key] [DatabaseGenerated(DatabaseGeneratedOption.None)]
  public int FlightNo { get; set; }
  #endregion

  #region Primitive Properties
  [StringLength(3)] // or [MaxLength(50)]
  public string Departure { get; set; }
  [StringLength(3)] // or [MaxLength(50)]
  public string Destination { get; set; }
  public DateTime Date { get; set; } // default is datetime2(7), not nullable
  public bool NonSmokingFlight { get; set; } = true; // not nullable
  public short? Seats { get; set; } // nullable in DB
  public short? FreeSeats { get; set; } // nullable in DB
  public string Memo { get; set; } // unlimited string --> nvarchar(max)
  #endregion

  #region Calculated Columns (in Database!)
  public decimal? Utilization { get; private set; }   // (see Fluent API)
  [Timestamp] // rowversion for concurrency check!
  public byte[] Timestamp { get; set; }
  #endregion

  #region Related Objects
  public virtual ICollection<Booking> BookingSet { get; set; } // 1:N
  // Properties need Annotation because Flight and Pilot have two relations
  [ForeignKey(nameof(PilotId))]
  [InverseProperty(nameof(BO.Pilot.FlightAsPilotSet))]
  public virtual Pilot Pilot { get; set; } // 1:1
  [ForeignKey(nameof(CopilotId))]
  [InverseProperty(nameof(BO.Pilot.FlightAsCopilotSet))]
  public virtual Pilot Copilot { get; set; } // 1:0/1 (see CopilotId!)
  // Explicit foreign key properties for the navigation properties
  public int PilotId { get; set; } // mandatory 1:1
  public int? CopilotId { get; set; } // optional 1:0/1, because nullable
  #endregion
```

Grundaufbau der Kontextklasse

PMC: Install-package Microsoft.EntityFrameworkCore.SqlServer
oder: Install-package Microsoft.EntityFrameworkCore.Sqlite
oder z.B.: Install-package Devart.Data.Oracle.EFCore

```
using BO;
using Microsoft.EntityFrameworkCore;
public class WWWingsContext : DbContext
{
  #region Entities for DB tables
  public DbSet<Flight> FlightSet { get; set; }
  public DbSet<Pilot> PilotSet { get; set; }
  public DbSet<Passenger> PassengerSet { get; set; }
  public DbSet<Booking> BookingSet { get; set; }
  #endregion
  #region Query Views for DB views and SQL/SP/TVF resultsets
  public DbQuery<DepartureStats> DepartureStats { get; set; } // View
  public DbQuery<FlightDTO> FlightDTO { get; set; } // SQL result
  #endregion
  // Runs at very instanciation of the context class
  protected override void OnConfiguring(DbContextOptionsBuilder builder)
  {
   string connectionString = "Load from config file";
   builder.UseSqlServer(connectionString);
  }
  // Runs at first instanciation of the context class in a process
  protected override void OnModelCreating(ModelBuilder mb)
  {   // configuration via Fluent API   }
}
```

Konfiguration mit Fluent-API in OnModelCreating()

Abweichende Tabellen und Spaltennamen festlegen

```
mb.Entity<Flight>().ToTable("Flight", schema: "Operation");
mb.Entity<Person>().Property(x=>x.Birthday).HasColumnName("BDay");
```

Einfachen Primärschlüssel festlegen

```
mb.Entity<Flight>().HasKey(f => f.FlightNo);
```

Zusammengesetzten Primärschlüssel festlegen

```
mb.Entity<Booking>().HasKey(b => new { b.FlightNo, b.PassengerID });
```

Autorwert für Primärschlüssel deaktivieren

```
mb.Entity<Flight>().Property(b => b.FlightNo).ValueGeneratedNever();
```

Zeichenkettenlänge begrenzen

```
mb.Entity<Flight>().Property(f => f.Departure).HasMaxLength(50);
```

Standardwerte (vom DBMS zu setzen)

```
mb.Entity<Flight>(f =>
   {  f.Property(x => x.Price).HasDefaultValue(123.45m);
      f.Property(x => x.Date).HasDefaultValueSql("getdate()");  });
```

Zusätzliche Indexe

```
mb.Entity<Flight>().HasIndex(x => x.FreeSeats).HasName("Index_FreeSeats");
mb.Entity<Flight>().HasIndex(f => new { f.Departure, f.Destination });
```

Berechnete Spalte

```
mb.Entity<Flight>().Property(p => p.Utilization)
.HasComputedColumnSql("100.0-(([FreeSeats]*1.0)/[Seats])*100.0");
```

Shadow Property (zusätzliche Spalte in der Tabelle)

```
mb.Entity<Flight>().Property<DateTime>("LastChange");
```

Beziehungen und Kaskadierendes Löschen

1:N-Beziehung von Pilot zu Flight ohne kaskadierendes Löschen

```
mb.Entity<Pilot>()
.HasMany(p => p.FlightAsPilotSet).WithOne(p => p.Pilot)
.HasForeignKey(f => f.PilotId).OnDelete(DeleteBehavior.Restrict);
```

1:N-Beziehung von Flight zu Booking mit kaskadierendem Löschen

```
mb.Entity<Flight>()
.HasMany(f => f.BookingSet).WithOne(b => b.Flight)
.HasForeignKey(p => p.FlightNo);
```

Value Converter

FrequentFlyer existiert als Zeichenkette ("Yes" | "No") im Objekt, aber soll in der Datenbank als Bool (true/false) abgelegt werden.

```
var cv = new ValueConverter<string, bool>(
                v => (v == "Yes" ? true : false), // from object to DB
                v => (v == true ? "Yes" : "No")); // from DB to object
mb.Entity<Passenger>().Property(x => x.FrequentFlyer).HasConversion(cv);
```

Enumeration *PilotLicenseType* soll in Datenbank nicht als Zahl, sondern Zeichenkette erscheinen.

```
var cv = new EnumToStringConverter<PilotLicenseType>();
mb.Entity<Pilot>().Property(e => e.PilotLicenseType).HasConversion(cv);
```

Massenkonfiguration im Fluent-API

Alle Datenbanktabellen sollen heißen wie die Klassen, nicht wie die Properties in der Kontextklasse (ausgenommen den Klassen, die eine [Table]-Annotation besitzen).

```
foreach (IMutableEntityType entity in mb.Model.GetEntityTypes())
  {
   var annotation = entity.ClrType?.GetCustomAttribute<TableAttribute>();
   if (annotation == null)  {
       entity.Relational().TableName = entity.DisplayName();   }
  }
```

Global Query Filter

Alle Abfragen auf "Flight" liefern nur Flüge mit freien Plätzen einer bestimmten Fluggesellschaft

```
mb.Entity<Flight>().HasQueryFilter(x => x.FreeSeats > 0 && x.AirlineCode
== "WWW");
```

Eine Abfrage kann dies mit IgnoreQueryFilters() außer Kraft setzen!

Über den Autor

Dr. Holger Schwichtenberg gehört zu den bekanntesten Experten für Webtechniken und .NET in Deutschland. Er hat zahlreiche Fachbücher veröffentlicht und spricht regelmäßig auf Fachkonferenzen. Sie können ihn und seine Kollegen für Entwicklungsarbeiten, Schulungen, Beratungen und Coaching buchen.
E-Mail: **buero@IT-Visions.de**
Website: **www.IT-Visions.de**
Weblog: **www.dotnet-doktor.de**