Festlegungen und Einschränkungen zur Konfiguration des Datenbankschemas

- Die Konfiguration des Datenbankschemas erfolgt bei Entity Framework Core auf drei Wegen:
  - mit Konventionen (Beispiel: Erkennung des Primärschlüssel auf der Basis <Klassenname>Id
  - mit Datenannotationen in den Entitätsklassen ([Table], [Index], [StringLength]) oder
  - dem Fluent-API in der Methode OnModelCreating() in der Kontextklasse).HasKey(„ <Klassenname>Id“).
  - Es gelten dabei drei Grundregeln:
    - Konfiguration per Datenannotationen oder Fluent-API wiegt schwerer als die Konventionen, d.h. Konfiguration setzt Konventionen für einzelne Fälle außer Kraft. Microsoft spricht bei Entity Framework Core von "Konvention vor Konfiguration". Damit ist jedoch gemeint, dass es Ziel ist, durch Konventionen die explizite Konfiguration soweit es geht überflüssig zu machen.
    - Wenn es sich widersprechende Datenannotationen und Fluent-API-Aufrufe gibt, wiegt immer der Fluent-API-Aufruf schwerer.
    - Man kann alle Konfigurationsmöglichkeiten per Fluent-API ausdrücken. Eine Teilmenge davon ist auch per Datenannotation möglich

Methoden der Fluent-Api

- Festlegung der Tabellennamen und Sichtnamen in der Datenbank, wenn diese abweichen oder einen Schemanamen abweichend von "dbo" haben: ToTable() bzw. ToView()
- Festlegung der Namen für Primärschlüsselspalten: HasKey() bzw. HasNoKey() für Datenbanksichten. Entity Framework Core geht seit Version 3.0 bei allen Sichten davon aus, dass diese nicht aktualisierbar sind und betrachtet sie als schlüssellose Entitätsklassen ("Keyless Entity").
- Festlegung der Spaltentypen und Spalteneigenschaften, wenn die .NET-Typnamen nicht eindeutig einem Datentyp im Datenbankmanagementsystem zuzuordnen sind: HasColumnType(), IsRequired(), HasMaxLength()
- Festlegung der Standardwerte für Spalten: HasDefaultValueSql() oder HasDefaultValue()
- Festlegung der Kardinalitäten zwischen Tabellen und der zugehörigen Fremdschlüssel: HasOne(), HasMany(), WithOne(), WithMany(), HasForeignKey() und HasConstraintName()
- Festlegung der Indizes: HasIndex()
- Festlegung, ob ein Spalteninhalt nach dem Einfügen oder Ändern eines Datensatzes von Entity Framework Core neu gelesen werden muss, weil er vom Datenbankmanagementsystem erzeugt wird: ValueGeneratedOnAddOrUpdate() und ValueGeneratedOnAdd() und ValueGeneratedNever()
- Festlegung der Einstellungen für kaskadierendes Löschen: OnDelete()

Festlegung der Namen von Schema, Tabellen und/oder der Namen und weiterer Merkmale von Spalten

Objekt der Festlegung	Datenannotation	Fluent-API (Methode OnModelCreating) in der DBContext-Klasse)
Tabellenname, Schemaname	Vor einer Klasse: [Table("Tabellenname")] oder unter zusätzlicher Angabe des Schemanamens [Table("Tabellenname", Schema = "Schemaname")]. Ohne die Schemanamensangabe landet die Tabelle immer im Standardschema "dbo".	modelBuilder.Entity() .ToTable("Tabellenname"); bzw. modelBuilder.Entity() .ToTable(@"Tabellenname", schema: @"Schemaname");
Spaltenname	Vor einem Property: [Column("Spaltenname ")]	modelBuilder.Entity() .Property(b => b.Property) .HasColumnName(@"Spaltenname");
Position der Spalte	[Column(Order = 5)] (Die Zählung erfolgt ab 0 aufwärts)	modelBuilder.Entity() .Property(b => b.Property) .HasColumnOrder(5);
Kommentar zur Spalte	[Comment("Das ist ein Kommentar")] public string { get; set; }	modelBuilder.Entity() .Property(b => b.Property) .HasComment("Das ist ein Kommentar");
Länge von Zeichenfolgen	[StringLength(200)] oder [MaxLength(200), MinLength(5)] oder [StringLength(200, MinimumLength = 5)]	modelBuilder.Entity() .Property(b => b.Property) .HasColumnName(@„Bezeichnung").HasColumnType(@"nvarchar(128)") .IsRequired().ValueGeneratedNever().HasMaxLength(128);
Wertebereich bei Zahlen	[Range(minWert, maxWert)	
Datentyp eines Feldes	[Column(TypeName = "nvarchar(200)")]	modelBuilder.Entity() .Property(b => b.Property) .HasColumnType(@"Datentyp");
Unicode	[Unicode(false)] (d.h. String wird nicht in nvarchar(), sondern in varchar() überführt)	modelBuilder.Entity() .Property(b => b.Property) .IsUnicode(false);
Präzision und Scale bei Dezimalzahlen	[Precision(14, 2)] public decimal Nettopreis{ get; set; }	modelBuilder.Entity() .Property(b => b.Nettopreis) .HasPrecision(14, 2);
Präzision bei Datum/Zeit-Werten	[Precision(0)] public DateTime DatumEintritt{ get; set; } (zwischen 0 und 7 – 0 bedeutet Zeitangabe ohne Millisekunden)	.Property(b => b. DatumEintritt) .HasPrecision(0);

Festlegung weiterer Merkmale von Spalten

Objekt der Festlegung	Datenannotation	Fluent-API
Obligatorisch	[Required]	modelBuilder.Entity() .Property(b => b.Property) .IsRequired()
Primärschlüssel Id	Wird automatisch erkannt, wenn z.B. der Spaltenname entsprechend lautet: public int tabellennameId {get; set;} Wenn nicht: [Key]	modelBuilder.Entity() .Property(b => b.Property) .HasColumnName(@„…Id").HasColumnType(@"int").IsRequired().ValueGeneratedOnAdd().HasPrecision(10, 0);
Primärschlüssel nicht Id-Feld, z.B. auch zusammengesetzte Schlüssel	[Key]	modelBuilder.Entity<Auto>() .HasKey(c => new { c.Land, c.Autokennzeichen });
Fremdschlüssel	[ForeignKey(„Schlüsselname“)]	
Vergabe von Namen für Schlüssel		modelBuilder.Entity<Auto>() .HasKey(c => new { c.Land, c.Autokennzeichen }).HasName("PrimaryKey_LandAutokennzeichen");
Standardwert	[DefaultValue(konkreterWert)]	modelBuilder.Entity() .Property(b => b.Property) .HasDefaultValue(konkreterWert);
Automatisch generierter Standardwert		modelBuilder.Entity() .Property(b => b.DatumErstellung) .HasDefaultValueSql(@"getdate()")
Berechnete Spalte (Computed Column)	Nicht vorhanden	modelBuilder.Entity<Mitarbeiter>() .Property(p => p.DisplayName) .HasComputedColumnSql("[NachName] + ', ' + [VorName]");

Festlegung weiterer Merkmale von Spalten: Indexe und Check-Constraints

Objekt der Festlegung	Datenannotation	Fluent-API
Index	[Index(nameof(NachName), nameof(VorName))] public class Mitarbeiter { public int MitarbeiterId { get; set; } public string NachName { get; set; } public string VorName { get; set; } }	modelBuilder.Entity<Mitarbeiter>() .HasIndex(p => new { p.VorName, p.NachName });
Unique Index	Wird vor dem Klassennamen eingetragen: [Index(nameof(Bezeichnung), IsUnique = true)] public class Funktion { public int FunktionId { get; set; } public string Bezeichnung { get; set; } }	modelBuilder.Entity<Abteilung>() .HasIndex(b => b.Bezeichnung) .IsUnique();
Name des Index	[Index(nameof(Bezeichnung), Name="Ix_Bezeichnung"]	modelBuilder.Entity<Abteilung>() .HasIndex(b => b.Bezeichnung) .HasDatabaseName("Ix_Bezeichnung");
Check		modelBuilder .Entity<Gehalt>() .ToTable(b => b.HasCheckConstraint("Ck_Gehalt", "[Gehalt] > 0"));
		entity.ToTable("mitarbeiter"); entity.ToTable(b => b.HasCheckConstraint("Ck_Geschlecht", "[Geschlecht] in ('m','w','d', '')"));
Konversionen		entity.Property(e => e.Geschlecht) .HasMaxLength(1) .IsUnicode(false) .HasConversion(v => v.ToString(), v => (EGeschlecht)Enum.Parse(typeof(EGeschlecht), v)) .HasDefaultValueSql("'");

Zur Fluent-Api: Übersicht über DeleteBehavior

Felder3		
Cascade	3	Für Entitäten, die vom Kontext nachverfolgt werden, werden abhängige Entitäten gelöscht, wenn der zugehörige Prinzipal gelöscht wird.
ClientCascade	4	Für Entitäten, die vom Kontext nachverfolgt werden, werden abhängige Entitäten gelöscht, wenn der zugehörige Prinzipal gelöscht wird.
ClientNoAction	6	Hinweis: Es ist ungewöhnlich, diesen Wert zu verwenden. Verwenden Sie stattdessen ClientSetNull , um das Verhalten von EF6 mit deaktivierten kaskadierenden Löschvorgängen abzugleichen.
ClientSetNull	0	Für Entitäten, die vom Kontext nachverfolgt werden, werden die Werte der Fremdschlüsseleigenschaften in abhängigen Entitäten auf NULL festgelegt, wenn der zugehörige Prinzipal gelöscht wird. Dies trägt dazu bei, den Graphen von Entitäten in einem konsistenten Zustand zu halten, während sie nachverfolgt werden, sodass dann ein vollständig konsistentes Diagramm in die Datenbank geschrieben werden kann. Wenn eine Eigenschaft nicht auf NULL festgelegt werden kann, weil es sich nicht um einen Nullable-Typ handelt, wird beim SaveChanges() Aufrufen eine Ausnahme ausgelöst.
NoAction	5	Für Entitäten, die vom Kontext nachverfolgt werden, werden die Werte der Fremdschlüsseleigenschaften in abhängigen Entitäten auf NULL festgelegt, wenn der zugehörige Prinzipal gelöscht wird. Dies trägt dazu bei, den Graphen von Entitäten in einem konsistenten Zustand zu halten, während sie nachverfolgt werden, sodass dann ein vollständig konsistentes Diagramm in die Datenbank geschrieben werden kann. Wenn eine Eigenschaft nicht auf NULL festgelegt werden kann, weil es sich nicht um einen Nullable-Typ handelt, wird beim SaveChanges() Aufrufen eine Ausnahme ausgelöst.
Restrict	1	Für Entitäten, die vom Kontext nachverfolgt werden, werden die Werte der Fremdschlüsseleigenschaften in abhängigen Entitäten auf NULL festgelegt, wenn der zugehörige Prinzipal gelöscht wird. Dies trägt dazu bei, den Graphen von Entitäten in einem konsistenten Zustand zu halten, während sie nachverfolgt werden, sodass dann ein vollständig konsistentes Diagramm in die Datenbank geschrieben werden kann. Wenn eine Eigenschaft nicht auf NULL festgelegt werden kann, weil es sich nicht um einen Nullable-Typ handelt, wird beim SaveChanges() Aufrufen eine Ausnahme ausgelöst.
SetNull	2	Für Entitäten, die vom Kontext nachverfolgt werden, werden die Werte der Fremdschlüsseleigenschaften in abhängigen Entitäten auf NULL festgelegt, wenn der zugehörige Prinzipal gelöscht wird. Dies trägt dazu bei, den Graphen von Entitäten in einem konsistenten Zustand zu halten, während sie nachverfolgt werden, sodass dann ein vollständig konsistentes Diagramm in die Datenbank geschrieben werden kann. Wenn eine Eigenschaft nicht auf NULL festgelegt werden kann, weil es sich nicht um einen Nullable-Typ handelt, wird beim SaveChanges() Aufrufen eine Ausnahme ausgelöst.