

Einführung in Entity Framework Core

Generierung des Modells per Reverse Engineering

Dozent: Dr. Thomas Hager, Berlin,
im Auftrag der Firma GFU Cyrus AG
20.10.2025 – 22.10.2025

Auch in diesem Abschnitt wird ein neues Projekt eingerichtet. Es besteht wieder aus einer Konsolen-Applikation und einer Klassenbibliothek (EFC03.ConsApp und EFC03.Lib).

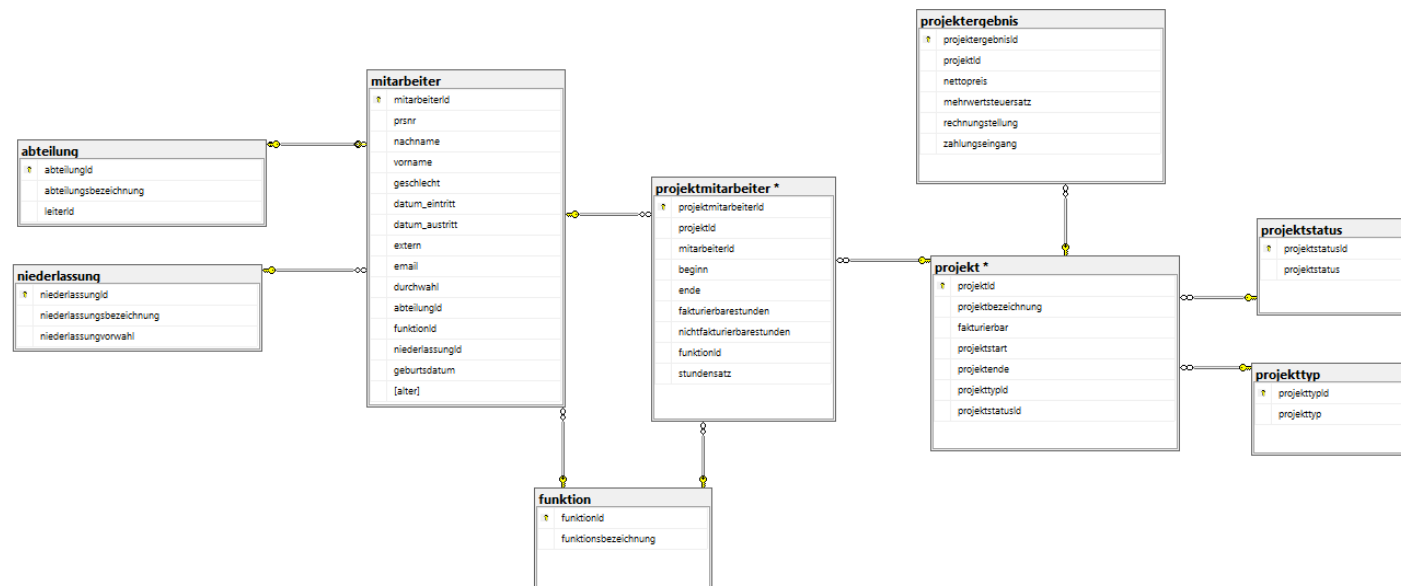
Diesmal wird jedoch von einer vorhandenen Datenbank ausgegangen, und mit Hilfe des Entity Developers (Devart) ein Reverse Engineering (DB First) durchgeführt.

Nach der Migration erfolgt eine Sichtung und Diskussion der Konfiguration nach dem Klassenentwurf per Fluent-API.



Einrichtung des erweiterten Projekts EFC03

- Wir kopieren die Anwendung EFC02 in eine neue Anwendung EFC03, die wieder aus einer ConsApp und einer Lib besteht. Diese Erweiterung ist notwendig, weil wir z.B. auch andere Relationen als die in EFC02 entwickelten 1:n – Relationen untersuchen wollen.
- Die Erweiterung bezieht sich
 - auf die Tabelle Projekt mit zwei Nachschlagetabellen (Projektstatus und Projekttyp) sowie einer mit Projekt per 1:1-Beziehung verbundenen Tabelle Projektergebnis
 - Die Tabelle Projektmitarbeiter, die eine wechselseitige Zuordnung von Mitarbeitern und Projekten ermöglicht. Zwischen Mitarbeitern und Projekten besteht also eine m:n-Beziehung.

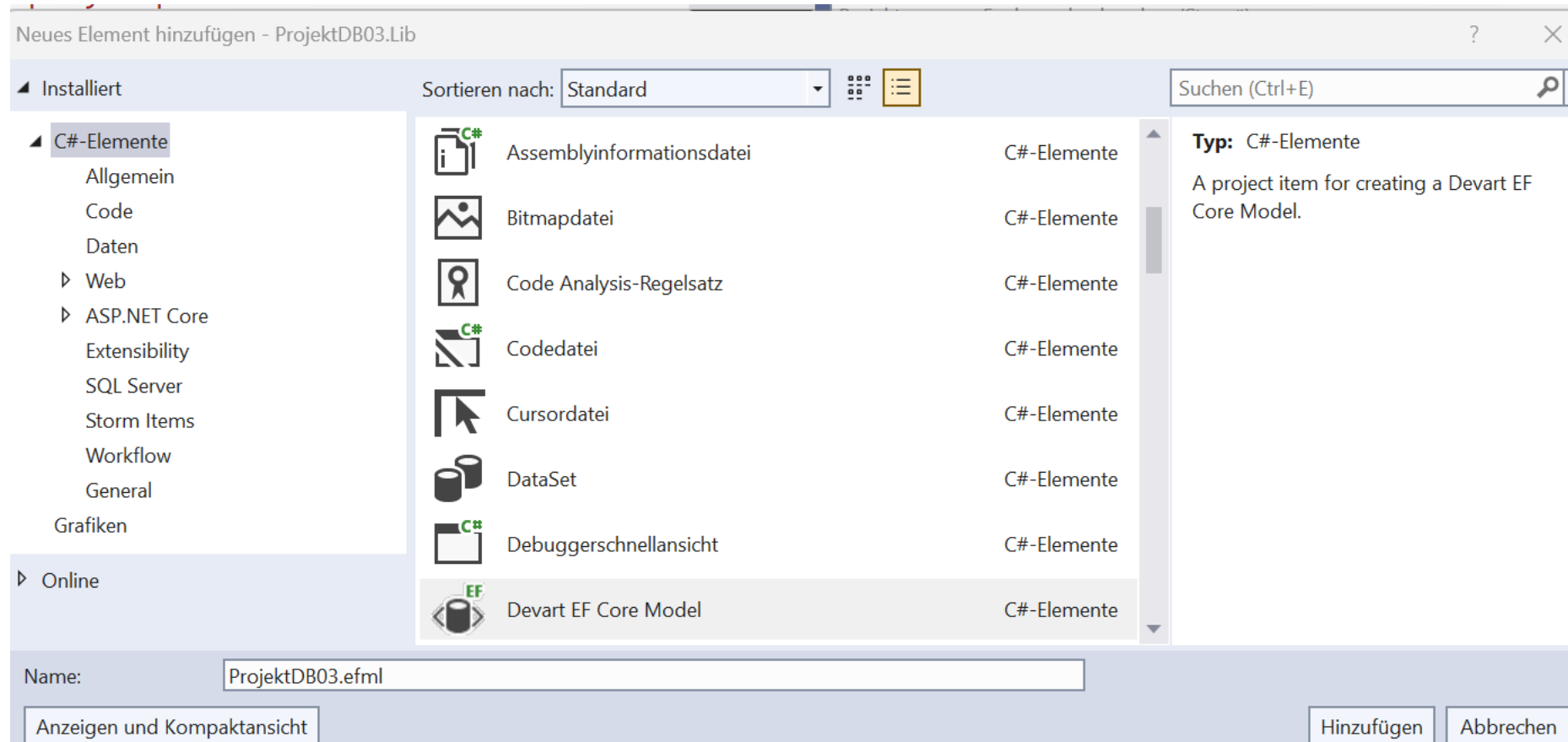


Reverse Engineering

- Methoden
 - Per Paket-Manager-Console: Scaffold-DbContext
Scaffold-DbContext "Server=XXX;Database=EFCxy; Integrated Security=true;TrustServerCertificate=Yes;Encrypt=False;"
Microsoft.EntityFrameworkCore.SqlServer [-Tables funktion, mitarbeiter]
-OutputDir Models
-ContextDir Models
-Context EFCxyDbContext
 - Per integrierten Devart Entity Developer
 - Extern mit Devart Entity Developer


Reverse Engineering aus EFC zum Einrichten von EFC03

- Nach dem Einrichten der EFC03.Lib fügen wir ein neues Element hinzu, das Devart EF Core Model:



Der Assistent zur Generierung der Modells aus der Datenbank EFC

Entity Developer: Create Model Wizard



Welcome to Entity Developer Create Model Wizard

Select how your model should be created.

☒ **Database First**

Creating a model from a pre-existing database. You should be able to establish a connection to an existing database.

☐ **Model First**

Creating a model without a pre-existing database and then generating a database from the model.

Click Next to continue.

< Back Next > Cancel

Entity Developer: Create Model Wizard

Set up data connection properties

On this page, specify connection settings for your model.

Provider: .Net Framework Data Provider for SqlServer

Server: Februar\SQLEXPRESS

Login details

☒ Use Windows Authentication

☐ Use SQL Server Authentication

User name:

Password:

☐ Save my password

Database: EFCTest

Connection String

data source=Februar\SQLEXPRESS;initial catalog=EFCTest;integrated security=True;persist security info=False;trustservercertificate=True;column encryption setting=Disabled

Test Connection Advanced...

< Back Next > Cancel

Auswahl der Objekte

Entity Developer: Create Model Wizard

Choose Model Contents

On this page you can choose model content.

What should the model contain?

☒ **Generate From Database**

Generates the model from a database. This wizard also lets you specify the database objects to include in the model.

☐ **Empty Model**

Creates an empty model as a starting point for designing a model visually. You can add database objects to the model later by the help of Database Explorer or Update From Database Wizard.

< Back Next > Cancel

Entity Developer: Create Model Wizard

Select database objects

From the list below, select database objects to include into your model.

Choose source

- ☒ Tables
 - ☒ dbo.abteilung
 - ☒ dbo.arbeitszeit
 - ☒ dbo.funktion
 - ☒ dbo.mitarbeiter
 - ☒ dbo.niederlassung
 - ☒ dbo.projekt
 - ☒ dbo.projektbeschreibung
 - ☒ dbo.projektergebnis
 - ☒ dbo.projektmitarbeiter
 - ☒ dbo.projektstatus
 - ☒ dbo.projekttyp
 - ☐ dbo.skills
 - ☐ dbo.test
- ☐ Views
- ☐ Procedures
- ☐ Functions

☐ Group by schemas ☒ Show all schemas

< Back Next > Cancel

Konfiguration der Entity-Klassen

Entity Developer: Create Model Wizard

Set up naming rules
On this page, specify naming rules for entities and their

Class and Method Names	Class Properties' Names
Case: FirstLetterUppercase	Case: FirstLetterUppercase
Remove prefix(es):	Remove prefix(es):
Remove suffix(es):	Remove suffix(es):
Add prefix:	Add prefix:
Add suffix:	Add suffix:
Pluralization: Singularize	Pluralization: Unchanged
<input checked="" type="checkbox"/> Remove underscore	<input checked="" type="checkbox"/> Remove underscore
<input type="checkbox"/> Add underscore	<input type="checkbox"/> Add underscore
<input checked="" type="checkbox"/> Remove invalid characters	<input checked="" type="checkbox"/> Remove invalid characters
<input type="checkbox"/> Add schema as prefix	<input checked="" type="checkbox"/> Pluralize collection navigation properties
<input type="checkbox"/> Add constraint details to navigation properties	
EntitySet Pluralization: Unchanged	
Original: Test Object_Name	Original: Test Object_Name
Becomes: TestObjectName	Becomes: TestObjectName

< Back Next > Cancel

Database First Settings

<input type="checkbox"/> Use lazy-loading proxies	<input checked="" type="checkbox"/> Preserve schema name in storage
<input type="checkbox"/> Use shadow foreign key properties	<input checked="" type="checkbox"/> Preserve columns details
<input checked="" type="checkbox"/> Detect Many-to-Many associations	<input checked="" type="checkbox"/> Preserve columns SqlType and Default
<input checked="" type="checkbox"/> Detect Table Per Type inheritances	<input checked="" type="checkbox"/> Execute procedures for result set detection
<input checked="" type="checkbox"/> Use database comments	<input type="checkbox"/> Use NULL parameter values
<input checked="" type="checkbox"/> Save connection to Model file	<input type="checkbox"/> Add complex types to diagram
<input type="checkbox"/> Use the connection string from appsettings.json: ProjektDBModelConnectionString	<input checked="" type="checkbox"/> Detect function-based column default value
<input type="checkbox"/> Rewrite connection string during regeneration	<input type="checkbox"/> Private setter for store generated properties
	<input type="checkbox"/> Constructor parameter

EF Core version: EF Core 9 Target Framework: .NET 9

Context Class Name: EFC03DBModel

Context Namespace: EFC03.Lib.Models

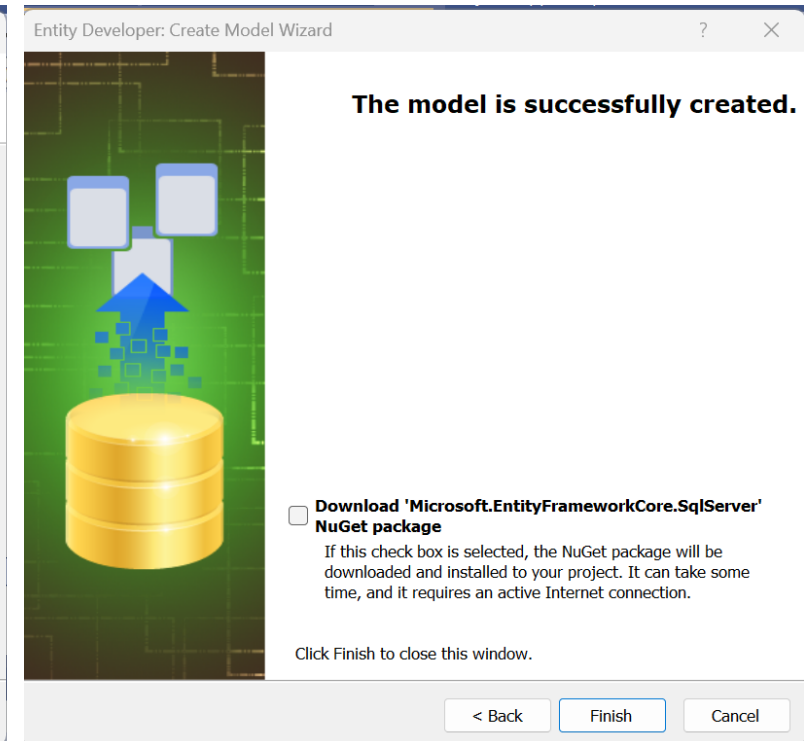
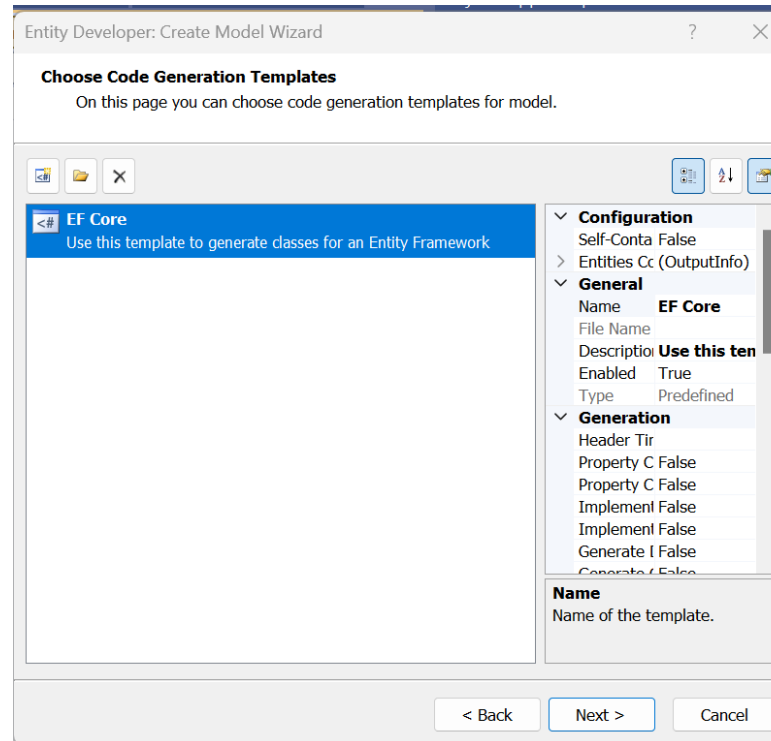
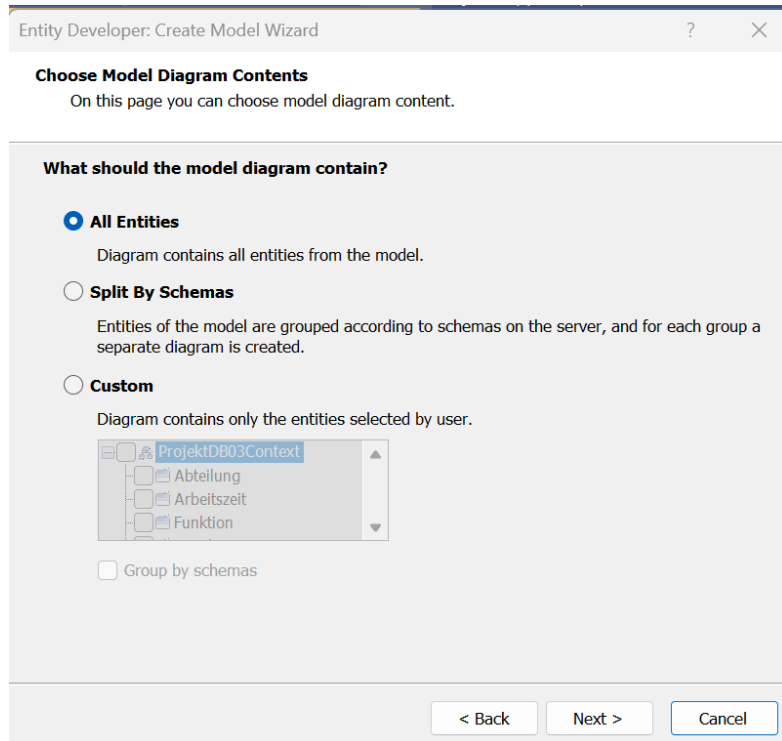
Entities Namespace: EFC03.Lib.Models

Model First Settings

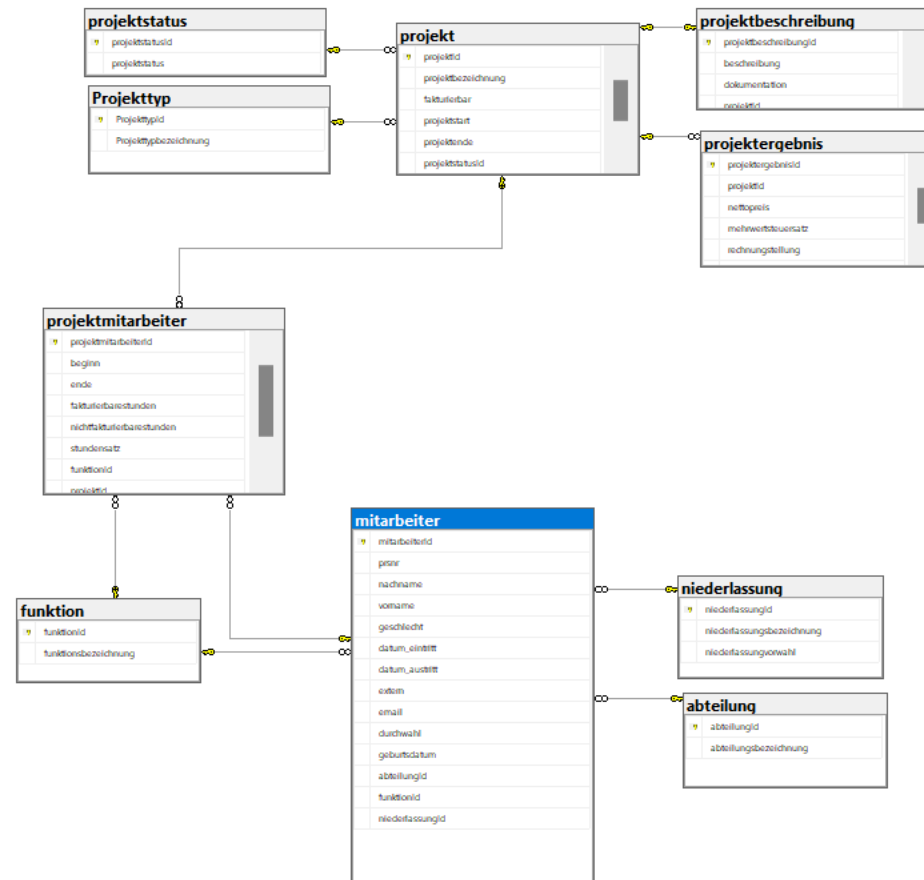
Default Schema:	Default Precision:
Default Length:	Default Scale:
<input type="checkbox"/> Default Unicode	

OK Cancel

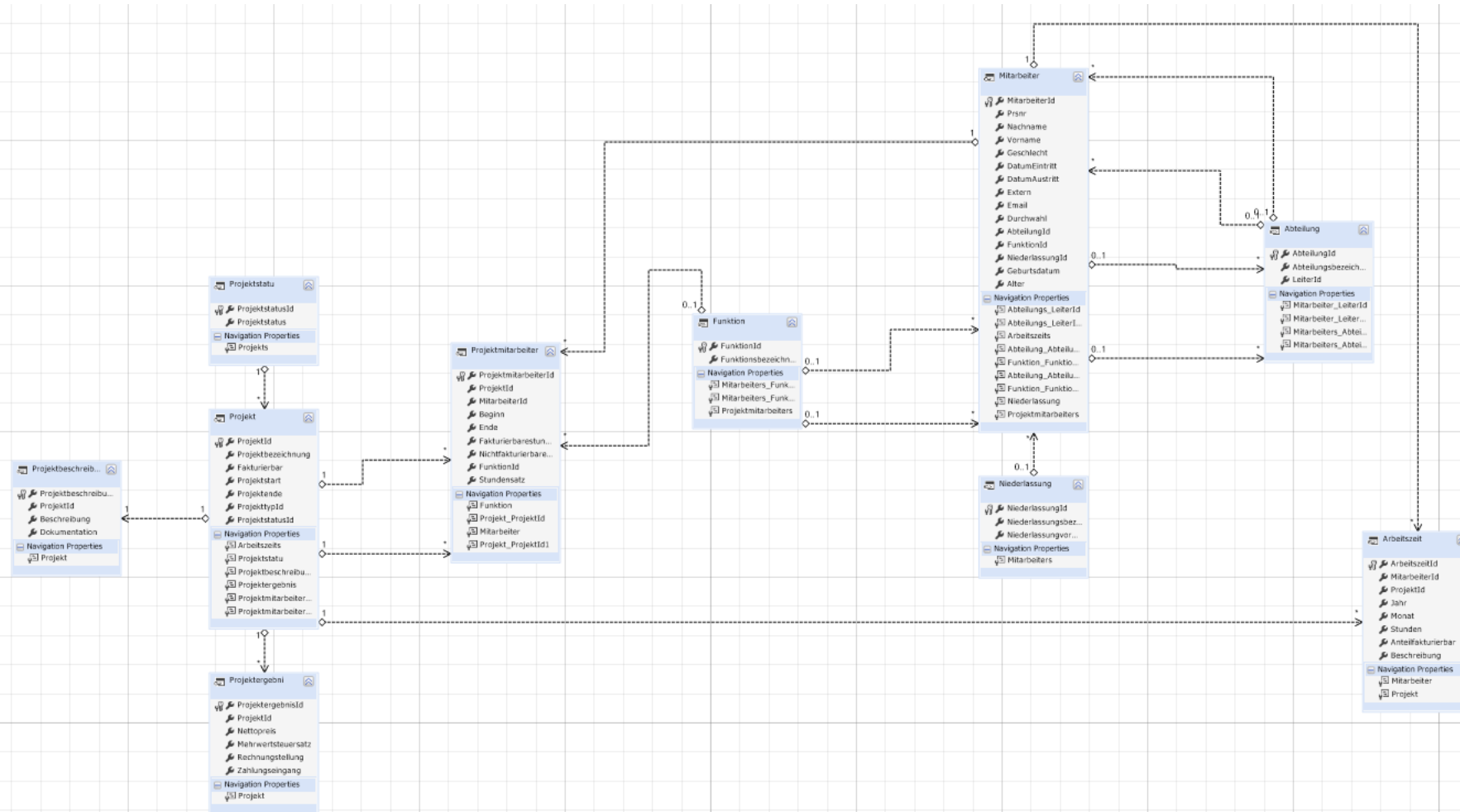
Abschließende Operationen



Das Datenbankschema



Das Klassendiagramm nach der Generierung durch Reverse Engineering /DB First



Variante mit Scaffolding

- Der Vorteil des Devart-Entity-Developers beim Reverse Engineering ist die Generierung von Klassendiagrammen, an denen der Entwickler noch Erweiterungen und Korrekturen vornehmen kann, um anschließend die Modell-Klassen und die DbContext-Klasse zu generieren
- In der PMC kann man mit dem Scaffold-Befehl die Modellklassen und die DbContext-Klasse direkt erzeugen

Scaffold-DbContext

```
"Server=localhost; Database=EFCXY; Trusted_Connection=True;  
TrustServerCertificate=Yes; Encrypt=False;"
```

```
Microsoft.EntityFrameworkCore.SqlServer
```

```
-OutputDir Models
```

```
-ContextDir Models
```

```
-Context EFCXYDbContext
```

SQL-Connection-String

Provider

Verzeichnis der Modell-
Klassen und der
DbContext-Klasse

Name der DbContext-
Klasse

Einzelne Tabelle(n)

- Einzelne Tabellen können übertragen werden:

```
Scaffold-DbContext "Server=localhost;Database=EFCXY; Trusted_Connection=True;
TrustServerCertificate=Yes; Encrypt=False;"
Microsoft.EntityFrameworkCore.SqlServer
-Tables "Skill"
-OutputDir Models
-ContextDir Models
-Context EFCXYDbContext
```

- Vorhandene Entities werden mit **-force** überschrieben:

```
Scaffold-DbContext "Server=localhost;Database=EFTest; Trusted_Connection=True;
TrustServerCertificate=Yes; Encrypt=False;"
Microsoft.EntityFrameworkCore.SqlServer
-Tables "Skill"
-OutputDir Models
-ContextDir Models
-Context EFCXYDbContext
-Force
```

[Entity Framework scaffold-dbcontext Commands with example in .NET - TheCodeBuzz](#)

Überschreiben einer Datenbank, in der bereits Daten vorhanden sind, per Migrationsbefehl

- Grundsätzlich ist nach Modelländerungen bzw. -erweiterungen die Migration in eine Datenbank, in der bereits Daten gespeichert sind, möglich.
- Zu beachten ist dabei die nach der Realisierung des Add-Migration-Befehls (vor dem eigentlich update) verfügbare Migrationsklasse mit ihren Methoden up und down. Hier können sich mögliche Probleme verstecken, die ein update-database verhindern. Durch Löschen nicht erforderlicher Anweisungen kann man das Update erfolgreich realisieren.
- Probleme werden z.B. diskutiert:
 - [c# - Adding EF Core Migrations to an existing database, while still enabling creation of the database from scratch - Stack Overflow](#)
 - [c# - How to add EF Core Migrations to existing db when you have no EFMigrationsHistory table? - Stack Overflow](#)

DIE VERFEINERUNG DER KONFIGURATION NACH DEM KLASSENENTWURF PER FLUENT-API

Entity Developer Modelle, Reverse Engineering am Beispiel der Mitarbeiterklasse

```
private void MitarbeiterMapping(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Mitarbeiter>().ToTable(@"mitarbeiter", @"dbo");
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.MitarbeiterId).HasColumnName(@"mitarbeiterId").HasColumnType(@"int").IsRequired().ValueGeneratedOnAdd().HasPrecision(10, 0);
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.Prsnr).HasColumnName(@"prsnr").HasColumnType(@"varchar(4)").ValueGeneratedOnAdd().HasMaxLength(4).HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.Nachname).HasColumnName(@"nachname").HasColumnType(@"nvarchar(50)").ValueGeneratedNever().HasMaxLength(50);
    modelBuilder.Entity<Mitarbeiter>().Property(x => x.Vorname).HasColumnName(@"vorname").HasColumnType(@"nvarchar(50)").ValueGeneratedNever().HasMaxLength(50);
    modelBuilder.Entity<Mitarbeiter>().Property(x => x.Geschlecht).HasColumnName(@"geschlecht").HasColumnType(@"varchar(1)").ValueGeneratedOnAdd().HasMaxLength(1)
        .HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.DatumEintritt).HasColumnName(@"datum_eintritt").HasColumnType(@"date").ValueGeneratedOnAdd().HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.DatumAustritt).HasColumnName(@"datum_austritt").HasColumnType(@"date").ValueGeneratedOnAdd().HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x => x.Extern).HasColumnName(@"extern").HasColumnType(@"bit").ValueGeneratedOnAdd().HasDefaultValueSql(@"0");
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.Email).HasColumnName(@"email").HasColumnType(@"varchar(50)").ValueGeneratedOnAdd().HasMaxLength(50).HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.Durchwahl).HasColumnName(@"durchwahl").HasColumnType(@"varchar(50)").ValueGeneratedOnAdd().HasMaxLength(50).HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x => x.AbteilungId).HasColumnName(@"abteilungId").HasColumnType(@"int").ValueGeneratedOnAdd().HasPrecision(10,
        0).HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x => x.FunktionId).HasColumnName(@"funktionId").HasColumnType(@"int").ValueGeneratedOnAdd().HasPrecision(10,
        0).HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x =>
        x.NiederlassungId).HasColumnName(@"niederlassungId").HasColumnType(@"int").ValueGeneratedOnAdd().HasPrecision(10, 0).HasDefaultValueSql(@"NULL");
    modelBuilder.Entity<Mitarbeiter>().Property(x => x.Geburtsdatum).HasColumnName(@"geburtsdatum").HasColumnType(@"datetime2").HasPrecision(0)
        .ValueGeneratedNever();
    modelBuilder.Entity<Mitarbeiter>().Property(x => x.DatumErstellung).HasColumnName(@"DatumErstellung").HasColumnType(@"datetime").IsRequired()
        .ValueGeneratedOnAddOrUpdate();
    modelBuilder.Entity<Mitarbeiter>().HasKey(@"MitarbeiterId");
}
```