Entity Framework Core 9 – Datenbankzugriff mit .NET
Einführung in das ORM-Konzept

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

Inhaltliche Übersicht:
Objektrelationales Mapping (ORM) mit Entity Framework Core

- **Wichtige Voraussetzungen und Grundlagen für ORM**
  - Das Konzept der Relationalen Datenbanken am Beispiel des SQL Servers
    - Tabellen-Strukturen, Datentypen, referentielle Integrität und andere CONSTRAINTS
    - T-SQL-Erweiterung von SQL: Functions, Stored Procedures, Calculated Columns
  - C#
    - Das objektorientierte Paradigma
    - Die integrierte Abfragesprache LINQ
    - Delegaten, Lambda-Ausdrücke
    - ADO.NET als Vorläufer und Basis von Entity Framework (EF) und Entity Framework Core (EFC)
- **Einführung in das Objektrelationale Mapping mit Entity Framework Core**
  - Der Widerspruch zwischen den beiden Konzepten – Auflösung per ORM
  - Das Grundkonzept des Objekt-Relationalen-Mappings (ORM) mit EF Core
  - Architektur von Entity Framework Core-basierten Anwendungen

Das Grundkonzept des Objekt-Relationalen-Mappings (ORM)
Historische Anmerkungen

- Seit den 1970er Jahren entwickelte sich das Konzept der Relationalen Datenbanken (RDB). Es beruht im wesentlichen auf der Erkenntnis, dass sich streng strukturierte flache Datenstrukturen wie Tabellen so wie Mengen und Relationen behandeln lassen (Stichworte: Vereinigung, Durchschnitt, Differenz und Verbund, Projektion und Selektion). Für RDB entwickelte man die deklarative Sprache SQL, sie wurde mit prozeduralen Elementen angereichert. (Beispiel SQL Server: SQL → T-SQL)
- Etwa seit den 1980er entwickelte sich das objektorientierte (klassenbasierte) Paradigma als Reaktion auf das bis dahin dominierende Paradigma der prozeduralen Programmiersprachen (Algol, Fortran, C, Pascal, Basic …). In den 1990ern begannen objektorientierte Sprachen (die die prozeduralen Elemente der Vorgängersprachen integrierten) sich durchzusetzen: Java, C++, C#, VB.NET und andere
- Mit dem Siegeszug von Arbeitsplatzcomputern/des PC und später des Internet entwickelte sich das Bedürfnis, Desktop- bw. Web-Anwendungen zu schreiben, mit denen man über nutzerfreundliche Grafische Oberflächen (GUI) auf Datenbanken und deren Daten auswertend und manipulierend zugreifen konnte. Dabei wurde der Widerspruch zwischen dem relationalen Paradigma und dem objektorientierten Paradigma schnell sichtbar.

Das Grundkonzept des Objekt-Relationalen-Mappings (ORM)
Vorläufer: Microsoft-Technologien zum Zugriff auf relationale Datenbanken

- 1990er Jahre: DAO (Data Access Objects) und ADO – ActiveX Data Objects. Möglichkeiten, mit Visual Basic (VB) bzw. Visual Basis for Applications (VBA) aus Anwendungen wie Excel, Word, Access mittels selbst geschriebener VB-bzw. VBA-Applikationen auf Datenbanken zuzugreifen. In diesen Anwendungen leben DAO und ADO bis heute → siehe Excel-VBA-Beispiel
- Seitdem Anfang der 2000er Jahre die Programmiersprachen C# und Visual Basic.Net eingeführt wurde, entwickelte sich aus DAO/ADO die Technologie ADO.NET (Active Data Objects).

Ein Beispiel für die ADO.NET-Technologie mit C#

```
using (System.Data.SqlClient.SqlConnection con = new System.Data.SqlClient.SqlConnection(sqlConnectionString))
{
          con.Open();
          using System.Data.SqlClient.SqlCommand cmd = new System.Data.SqlClient.SqlCommand())
          {
                cmd.Connection = con;
                cmd.CommandText = sqlStatement;
                try
                {
                    nRows = cmd.ExecuteNonQuery();
                    msg = string.Format($"Es sind {nRows} Datensätze betroffen");
                }
                catch (Exception ex)
                {
                    errmsg += "\n" + ex.Message.ToString();
                    success = false;
                }
          }
}
```

Für jede Operation (Create,Insert, Update, Delete) muss der SQL-Befehl einzeln aufgebaut werden. Dabei müssen die Properties der C#-Klassen auf die Tabellen und Spalten der SQL-Datenbank und die C#-Datentypen auf die SQL-Server-Datenbank-Felddatentypen gemappt werden. Bei Abfragen (Read) erfolgt das in umgekehrter Richtung.

Das Grundkonzept des Objekt-Relationalen-Mappings (ORM)
Von ADO.NET zum klassischen Entity Framework

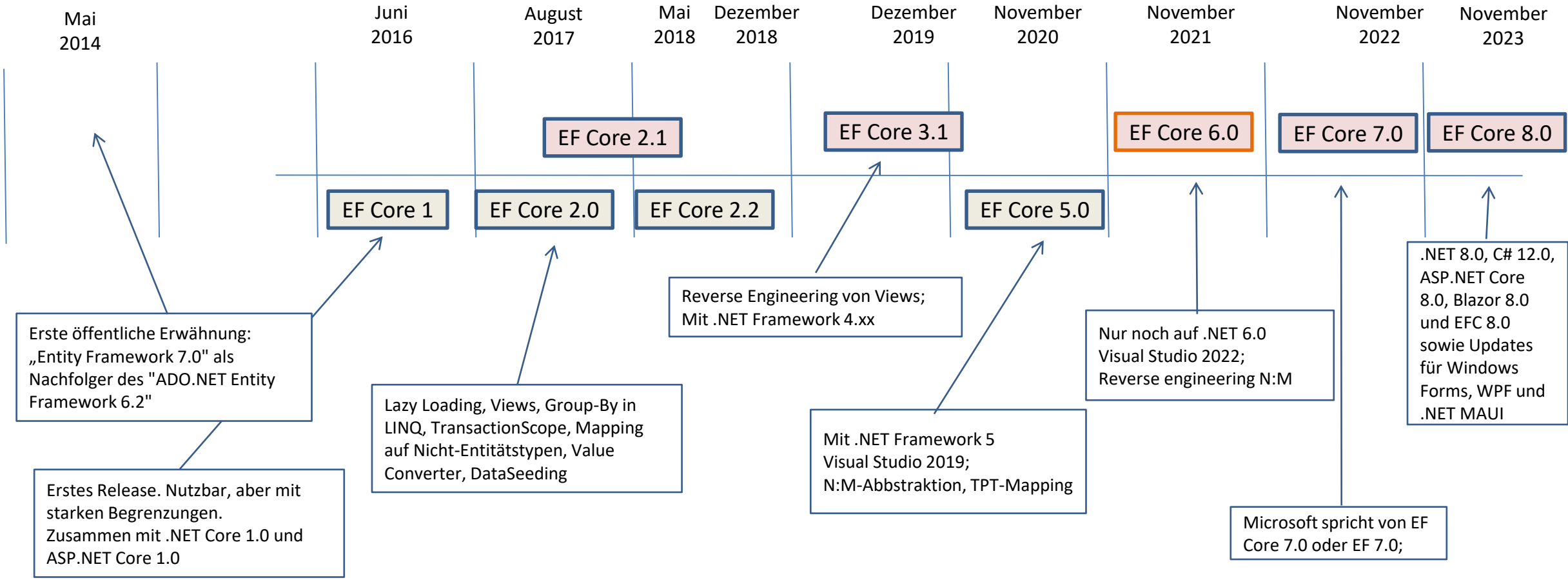
- ADO.NET in .NET 1.0 bis 3.5 enthielt kein ORM
- ADO.NET bietet eine Struktur, mit der Nutzer Daten über Verbindungen auswählen, einfügen, aktualisieren und löschen können, mit Hilfe von Connections, Commands und DataReadern.
- ADO.NET zwingen den Anwendungsentwickler dazu, die abgerufenen Daten in einer Weise zu behandeln, die eng an das physische Datenbankschema gekoppelt ist. Wenn man beispielsweise Datensätze aus der Datenbank abruft, öffnen man eine Verbindung, erstellt und führt ein Befehlsobjekt aus, und verwendet dann einen DataReader, um jeden Datensatz mit datenbankspezifischen Spaltennamen zu durchlaufen.
- Wenn man ADO.NET verwendet, muss immer die physische Struktur der Backend-Datenbank berücksichtigt werden. Das Schema jeder Datentabelle, komplexer SQL-Abfragen zur Interaktion mit Datentabellen usw. muss bekannt sein.
- Dies zwingt den Anwendungsentwickler dazu, umfangreichen C#-Code zu verfassen, da C# eben nicht direkt die Sprache des Datenbankschemas spricht. (siehe Beispiel E:\Projekte\WGJ\wgj.anlagen\Business.Definitions\Staff)
- Ein weiteres Problem für Anwendungsentwickler ist die Änderungsverfolgung. Das Abrufen der Daten aus der Datenbank ist ein Schritt des Prozesses, aber alle Änderungen, Hinzufügungen und/oder Löschungen müssen vom Entwickler nachverfolgt werden, damit sie in den Datenspeicher zurückgespielt werden können.

Das Grundkonzept des Objekt-Relationalen-Mappings (ORM)
Erste Ansätze des ORM in .NET

- In den 2000er Jahren entwickelten sich Ansätze des ORM, zunächst von anderen Herstellern oder als Eigenentwicklung (der Vortragende hat sich z.B. für ein C#-Projekt mit SQL-Server Ende der 2000er selbst einen Generator für das Mappping zwischen den C#-Objekten und den SQL-Server-Tabellen geschrieben.)
- Die erste Version eines von Microsoft entwickelten ORM erschien als Teil des .NET Framework 3.5 (Service Pack 1) im Jahr 2008. Damals trug es den Namen **ADO.NET Entity Framework**.
- Zusammen mit .NET Core Version 1.0 und ASP.NET Core Version 1.0 ist auch **Entity Framework Core** in Version 1.0 am 27. Juni 2016 erstmals erschienen.
- Entity Framework Core 7.0 erschien am 8. November 2022 im Rahmen von .NET 7.0, zusammen mit ASP.NET Core 7.0 und C# 11. Entity Framework Core 8.0 erschien am 8. November 2023 im Rahmen von .NET 8.0, zusammen mit ASP.NET Core 8.0 und C# 12. Ihre Nutzung setzt Visual Studio 2022 voraus.
- Die Verfügbarkeit von objektrelationalen Mapping-Frameworks (gemeinhin als ORM bezeichnet) hat den Datenzugriff und die Datenmanipulation erheblich verbessert, indem sie den Großteil der CRUD-Datenzugriffsaufgaben für den Entwickler verwaltet.
- Der Entwickler erstellt ein Mapping zwischen den .NET-Objekten und der relationalen Datenbank, und das ORM verwaltet die Verbindungen, die Abfrageerstellung, die Änderungsverfolgung und die Persistenz der Daten. So kann sich der Entwickler auf die geschäftlichen Anforderungen der Anwendung konzentrieren

Die Historie von EF Core

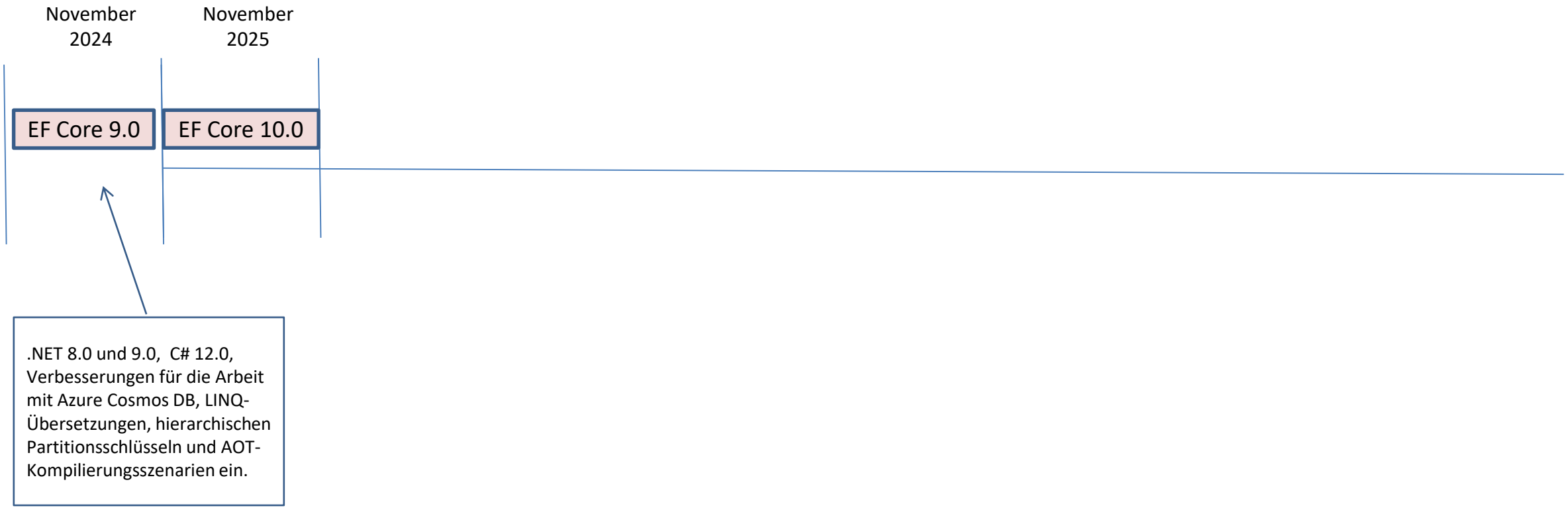
- Update Model from Database
- Vollständige TPC(T)-Vererbung
- Entity Splitting
- Reverse Engineering von SPs
- Generierung von SPs
- Mapping auf primitive Typen & Nicht-Entitätstypen
- Mapping von JSON-Spalten
- MinBy() und MaxBy()
- Take(10..5)
- Automatische Validierung beim Speichern



Genau wie die anderen Produkte der Core-Produktfamilie ist das Entity Framework Core ebenfalls plattformunabhängig. Entity Framework Core läuft in allen Anwendungsarten und allen Betriebssystemen, auf denen das moderne .NET läuft. Entity Framework Core kann man also nicht nur in Desktop-Anwendungen auf Windows, in Servern/Services und Konsolenanwendungen auf Windows, Linux und macOS, sondern auch in Apps auf iOS, Android, macOS und Windows einsetzen.

Die Historie von EF Core - Fortsetzung

- Update Model from Database
- Vollständige TPC(T)-Vererbung
- Entity Splitting
- Reverse Engineering von SPs
- Generierung von SPs
- Mapping auf primitive Typen & Nicht-Entitätstypen
- Mapping von JSON-Spalten
- MinBy() und MaxBy()
- Take(10..5)
- Automatische Validierung beim Speichern

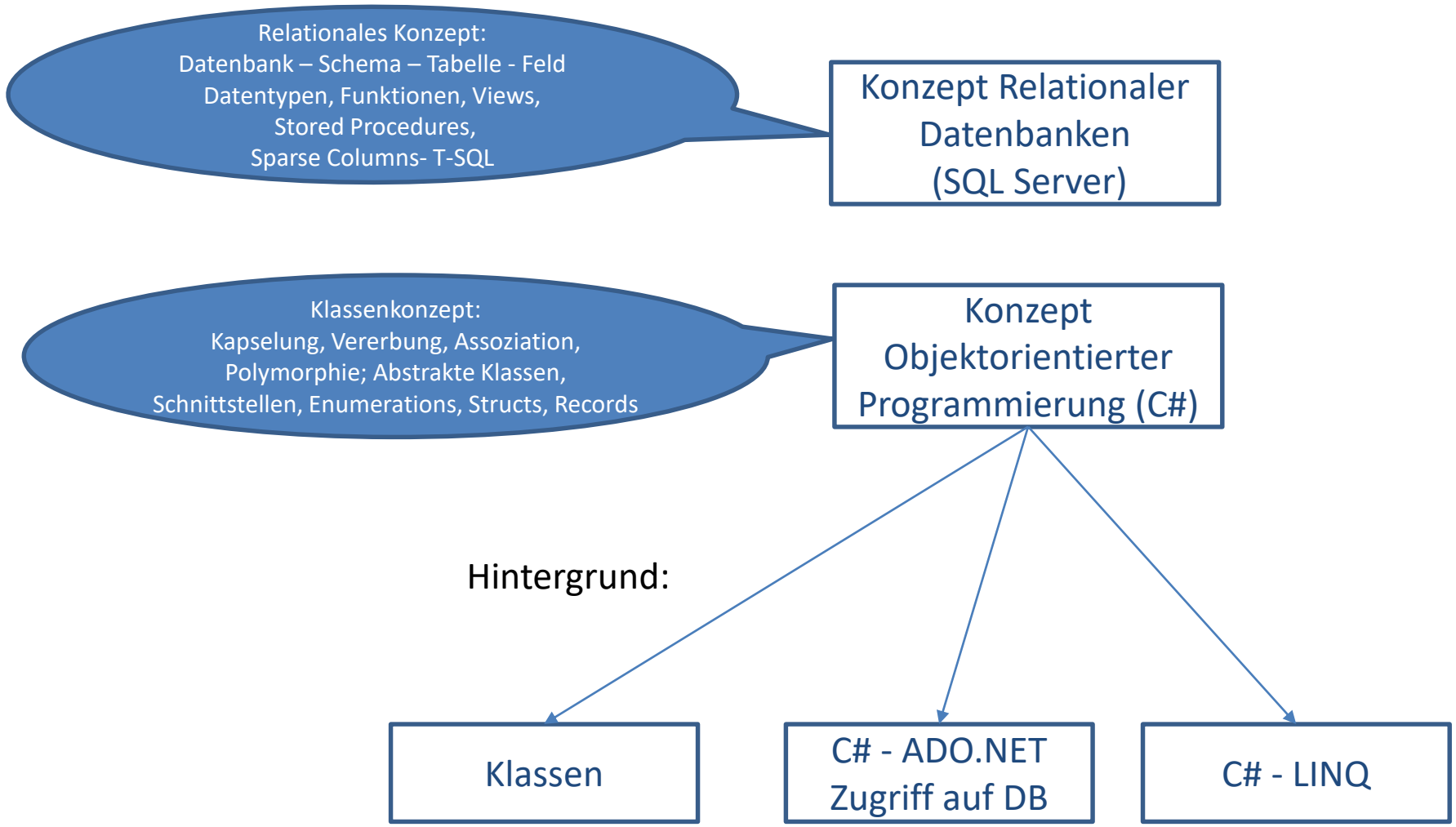


Genau wie die anderen Produkte der Core-Produktfamilie ist das Entity Framework Core ebenfalls plattformunabhängig. Entity Framework Core läuft in allen Anwendungsarten und allen Betriebssystemen, auf denen das moderne .NET läuft. Entity Framework Core kann man also nicht nur in Desktop-Anwendungen auf Windows, in Servern/Services und Konsolenanwendungen auf Windows, Linux und macOS, sondern auch in Apps auf iOS, Android, macOS und Windows einsetzen.

Perspektiven von .NET, C# und Entity Framework Core

- .NET 9.0 erschien am 14. November 2023, EF Core 9 (EF9) wurde mit ausgeliefert
  - What's New in EF Core 9 – Satish Yadav – Blog by Satish Yadav
  - Data Seeding - EF Core | Microsoft Learn (Neu: UseSeeding, UseSeedingAsync)
  - 10 New Features in EF 9
- Mai 2025: Microsoft hat mit der Preview-Reihe von .NET 10.0 begonnen. .NET 10.0 beinhaltet:
  - C# 13.0
  - ASP.NET Core 9.0
  - Entity Framework Core 10.0
  - sowie neue Features für .NET MAUI (Multi-platform App UI)

Aufgabe: Zusammenführen der beiden unterschiedlichen Konzepte



Das Konzept des Objektrelationalen Mapping (ORM)

Widerspruch zwischen dem Klassenkonzept und dem Datenbank-Konzept

Lösungsansatz ORM

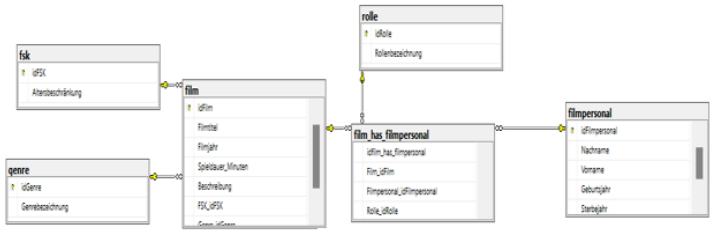
Lösungsansatz Eigene Mapping-Klassen

Lösungsansatz EntityFramework

Lösungsansatz EntityFramework Core

Das Grundkonzept des Objekt-Relationalen-Mappings (ORM): Lösungen, um den Widerspruch zwischen dem relationalen Paradigma und dem objektorientierten Paradigma zu entschärfen

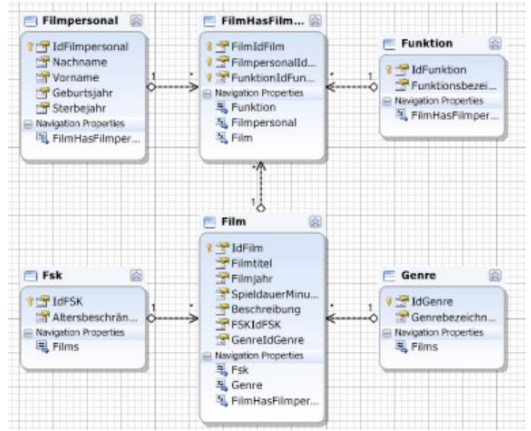
Relational



1:n, n:m, m:n-Beziehungen; referentielle Integrität! Tabellen mit Datensätzen; Funktionen, Views, Stored Procedures

???

Objektorientiert



Klassen, aus denen Objekte abgeleitet werden: Kapselung, Vererbung, Assoziation, Polymorphie;

Object-relational Impedance Mismatch
(objektrelationale Unverträglichkeit)

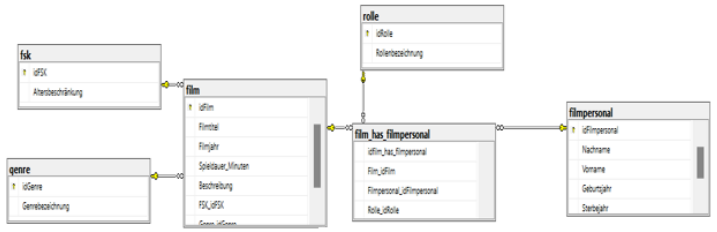
Das Grundkonzept des Objekt-Relationalen-Mappings (ORM): Die Beziehungen zwischen den Tabellen einer Datenbank und den Objekten in einem C#-Programm

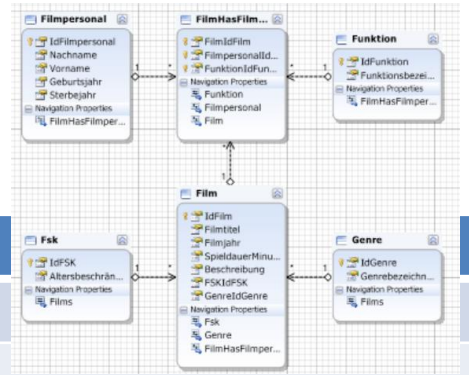
1:1, 1:n, n:m-Beziehungen;
Tabellen mit Datensätzen

Relational

Objektorientiert

Klassen, aus denen Objekte abgeleitet werden: Kapselung, Vererbung, Assoziation, Polymorphie;



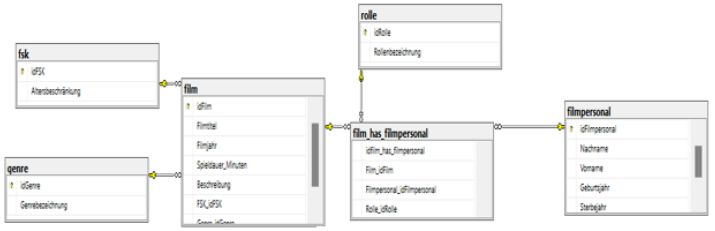


Datenbank	korrespondiert mit	Klasse bzw. Objekt
Tabellenstruktur		Klasse
Feldname/Spaltenüberschrift		Attribut/Property
Datentyp einer Spalte		Datentyp der Property
Zeilen unterhalb der Überschriftenzeile - Datensatz		Entsprechen jeweils den Ausprägungen der Properties eines aus der Klasse abgeleiteten Objekts → Elemente einer Collection (List, Array,….)
Feldwert		Ausprägung der Property in einem aus der Klasse abgeleiteten Objekt
Primärschlüssel (UNIQUE und NOT NULL, evtl. Autowert, GUID)		Eine eindeutige Klasseninstanz
Fremdschlüssel – Beziehungen zu anderen Tabellen		Referenz zu einer anderen Klasse (Assoziation) - Navigationsproperties
Stored Procedure, Berechnete Felder, Funktionen		Methoden eines Objekts
Abfragen, SELECT … FROM … WHERE		.NET LINQ

Das Grundkonzept des Objekt-Relationalen-Mappings (ORM)
Die Korrespondenz der Datentypen

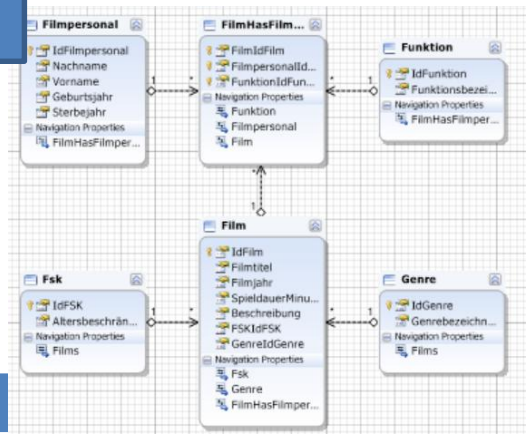
1:n, n:m, m:n-Beziehungen;
Tabellen mit Datensätzen

Relational



Klassen, aus denen Objekte abgeleitet werden:
Kapselung, Vererbung, Assoziation, Polymorphie;

Objektorientiert

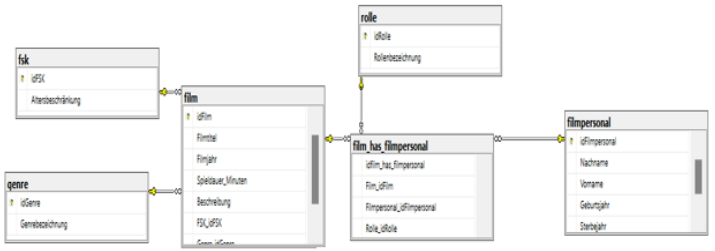


Datentypen	korrespondiert mit	Datentypen
tinyint, smallint, int, bigint		byte, short, integer, long
real, float, decimal(18,2), money		float, double, decimal
bit		boolean
nvarchar(max) ,varchar(n), char(n)		char, string
datetime2(7), datetime2(0), date, time		DateTime, DateOnly, TimeOnly
varbinary(max)		byte[]
hierarchid		hierarchid

Das Grundkonzept des Objekt-Relationalen-Mappings (ORM)
Die Korrespondenz weiterer Datenstrukturen

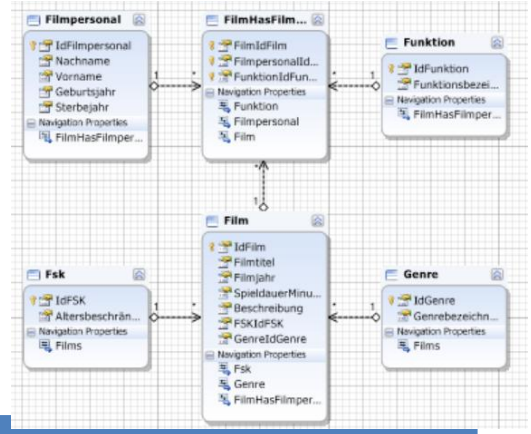
1:n, n:m, m:n-Beziehungen;
Tabellen mit Datensätzen

Relational



Klassen, aus denen Objekte abgeleitet werden:
Kapselung, Vererbung, Assoziation, Polymorphie;

Objektorientiert

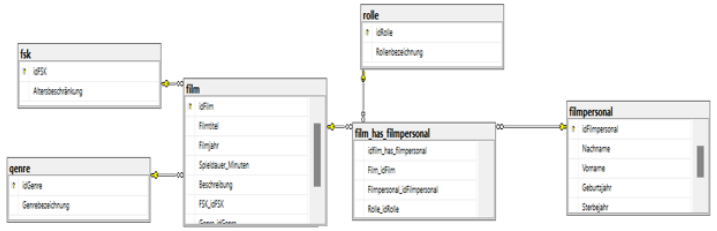


Kollektionen	korrespondiert mit	Kollektionen
Tabelle mit Datensätzen		list oder array oder dictionary als Sammlungen mit struct, class, record als Elementen
Separate Nachschlagetabelle oder Check-Constraint für ein Feld in einer Tabelle		enum oder class
Abfrage oder Tabellenwertfunktion		Array<T>, List<T>, Dictionary<key,value>
Ausschnitt aus einem Datensatz		Tupel
Datensatz		object, struct, record

Das Grundkonzept des Objekt-Relationales-Mappings (ORM)
Die Korrespondenz der Methoden

1:n, n:m, m:n-Beziehungen;
Tabellen mit Datensätzen

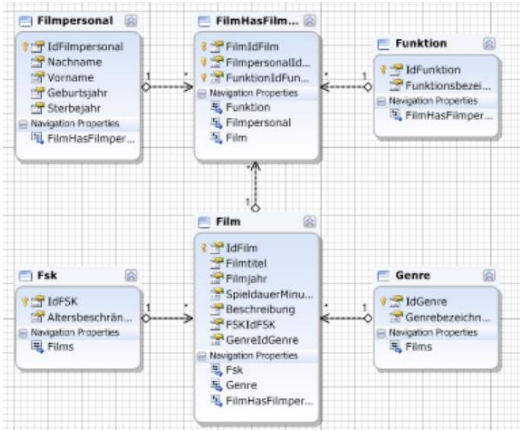
Relational



SQList ist die deklarative Sprache des SQL-Server (Select …)
T-SQL ist die prozedurale Sprache des SQLServers

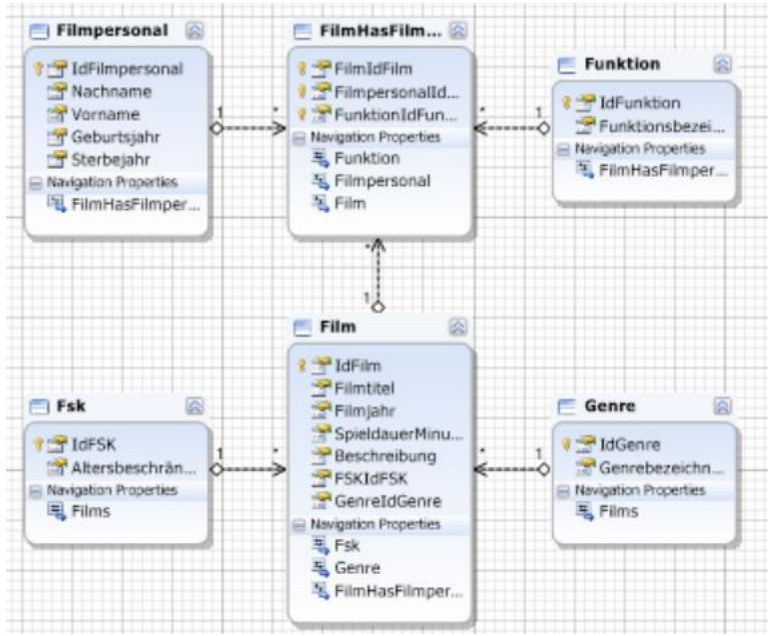
Klassen, aus denen Objekte abgeleitet werden:
Kapselung, Vererbung, Assoziation, Polymorphie;

Objektorientiert



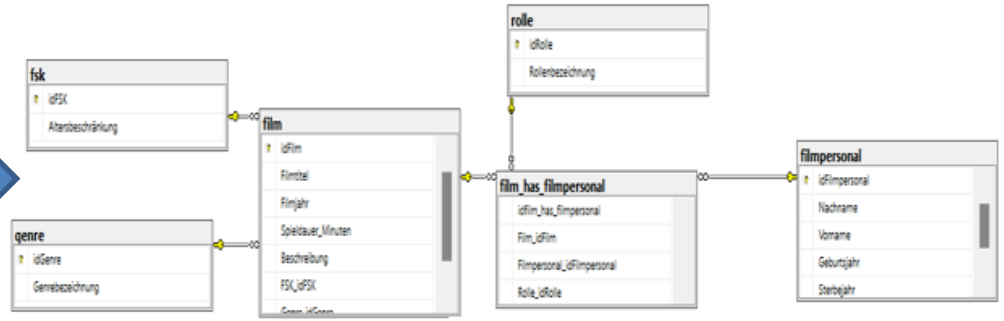
Prozedurale Elemente (T-SQL)	korrespondiert mit	Methoden
Calculated Column		
Trigger		
Benutzerdefinierte Skalarwertfunktion, Tabellenwertfunktion, Aggregatfunktion		Methode
Benutzerdefinierte Stored Procedure		

Das Grundkonzept des Objekt-Relationales-Mappings (ORM)



Klassenmodell

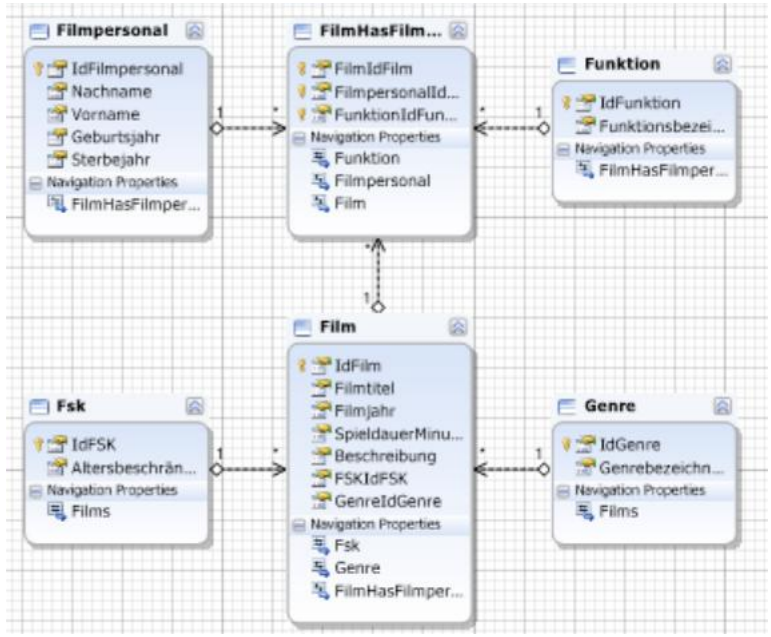
Entity Framework Core



Datenbankschema

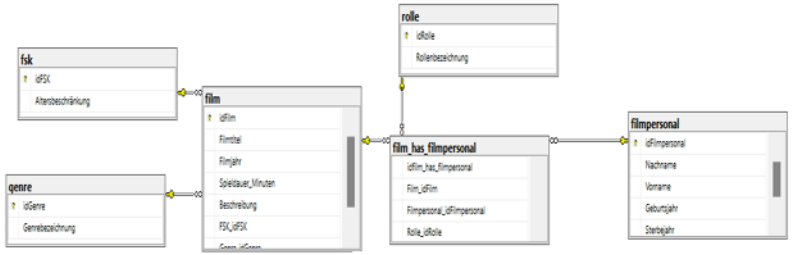
Das Grundkonzept des Objekt-Relationales-Mappings (ORM) Migration

- Es gibt zwei Möglichkeiten der Realisierung des ORM:
  - Forward Engineering (Code First): Hierbei werden die Modellklassen und ihre Beziehungen in Tabellen und Relationen überführt
  - Reverse Engineering (Database First): Hierbei werden die Tabellen und deren Relationen einer vorhandenen Datenbank in Modellklassen und ihre Beziehungen überführt



Forward Engineering (Code First)

Reverse Engineering (Database First)



Die Ablösung von Entity Framework durch Entity Framework Core (EFC)

- Das klassische ADO.NET bzw. auch ADO.NET Entity Framework wird vollständig durch EFC abgelöst.
- Folgende Funktionen entfallen vollständig:
  - In Entity Framework Core gibt es nur noch das Code-based Modelling (früher Code First), mit dem man sowohl Programmcode aus Datenbanken erzeugen kann (Reverse Engineering) als auch Datenbanken aus Programmcode (Forward Engineering).
  - Das Entity Data Model (EDM) und die XML-Repräsentation davon (EDMX) entfallen. Bisher wurde auch beim Code First intern ein EDM im RAM erzeugt. Der Overhead entfällt.
  - Die Basisklasse ObjectContext für den Entity Framework-Kontext entfällt. Es gibt nur noch die Basisklasse DbContext. DbContext ist jetzt in Entity Framework Core kein Wrapper um ObjectContext mehr, sondern eine komplett neue, eigenständige Implementierung.
  - Die Basisklasse EntityObject für Entitätsklassen entfällt. Die Entitätsklassen sind nun immer Plain Old CLR Objects (POCOs). ▪ Auch die Abfragesprache Entity SQL (ESQL) entfällt. Es gibt nur noch Unterstützung für LINQ, SQL und Stored Procedures (SPs) sowie Table Valued Functions (TVFs).
  - Automatische Schemamigrationen werden nicht mehr angeboten. Schemamigrationen inklusive der Ersterstellung eines Datenbankschemas sind nun zur Entwicklungszeit immer manuell auszuführen. Zur Laufzeit kann eine Migration weiterhin beim ersten Zugriff auf die Datenbank erfolgen.
  - Einige Szenarien des komplexeren Mappings zwischen Tabellen und Typen entfallen. Dazu gehören das Multiple Entity Sets per Type (MEST, verschiedene Tabellen auf dieselbe Entität abbilden) und das Kombinieren der Strategien Table per Hierarchy (TPH), Table per Type (TPT) und Table per Concrete Type (TPC) in einer Vererbungshierarchie.

Was ist Entity Framework Core

- Entity Framework Core ist eine .Net Technologie, die den Datenzugriff für objektorientierten Programmcode abstrahiert. EF Core ist am Ende eine Abstraktion auf der ADO.NET Technologie von .NET.
- Es überbrückt den Unterschied unserer Datenbanken (zumeist Relationale Datenbanken oder SQL Datenbanken aber auch objektrationale Datenbanken) zu objektorientiertem Code.
- Dabei stellt es nicht nur eine Zugriffstechnologie dar, sondern auch eine Technologie zum Verwalten des gesamten Datenbank-Managements, inklusive der folgenden Punkte:
  - Datenbanksession
  - Transaktionen
  - Schemaverwaltung
  - Datentypen und Beziehungen definieren

Vorteil von EF Core gegenüber ADO.NET

- Genau wie die anderen Produkte der Core-Produktfamilie ist das Entity Framework Core ebenfalls plattformunabhängig. Entity Framework Core läuft in allen Anwendungsarten und allen Betriebssystemen, auf denen das moderne .NET läuft. Entity Framework Core kann man also nicht nur in Desktop-Anwendungen auf Windows, in Servern/Services und Konsolenanwendungen auf Windows, Linux und macOS, sondern auch in Apps auf iOS, Android, macOS und Windows einsetzen.
- Bei der traditionellen ADO.NET - Arbeitsweise mit untypisierten DataSets werden Tabellen und Spalten durch Indexer-Argumente vom Zeichenfolgentyp angesprochen, und auf die Werte einer Tabellenzeile greift man über die DataRow-Eigenschaft ItemArray mit dem Datentyp Object[] zu, was lästige und fehleranfällige Typumwandlungen erfordert.

- Bei der Arbeit mit dem EF Core spricht man typsicher die Eigenschaften von Objekten an, was zu einer höheren Effektivität und einer geringeren Fehlerquote führt
- Dadurch, dass EF Core uns die Konfigurationsarbeit mit der Datenbank abnimmt, können wir uns vollständig auf die Businesslogik unserer Applikation konzentrieren.