.NET BIBLIOTHEK

hrsg. von dr. holger SCHWICHTENBERG

holger SCHWICHTENBERG

Effizienter Datenzugriff mit

ENTITY FRAMEWORK CORE

Datenbankprogrammierung mit C#
für .NET Framework, .NET Core und Xamarin

HANSER

www.IT-Visions.de
Dr. Holger Schwichtenberg

Schwichtenberg

Effizienter Datenzugriff
mit Entity Framework Core

Holger Schwichtenberg

Effizienter Datenzugriff mit Entity Framework Core

Datenbankprogrammierung mit C# für .NET Framework, .NET Core und Xamarin

HANSER

Der Autor:
Dr. Holger Schwichtenberg, Essen
www.IT-Visions.de

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Lektorat: Sylvia Hasselbach
Copy editing: Matthias Bloch, Essen
Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de
Umschlagrealisation: Stephan Rönigk
Gesamtherstellung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-44898-8
E-Book-ISBN: 978-3-446-44978-7

Inhalt

Vorwort ..... **XV**

Über den Autor ..... **XVII**

1 Einleitung **1**

1.1 Programmiersprache in diesem Buch 1

1.2 Fallbeispiele in diesem Buch 1

1.2.1 Entitäten 2

1.3 Anwendungsarten in diesem Buch 5

1.4 Hilfsroutinen zur Konsolenausgabe 6

1.5 Programmcodebeispiel zum Download 10

2 Was ist Entity Framework Core? **13**

2.1 Was ist ein Objekt-Relationaler Mapper? 13

2.2 ORM in der .NET-Welt 15

2.3 Versionsgeschichte von Entity Framework Core 16

2.4 Unterstützte Betriebssysteme 17

2.5 Unterstützte .NET-Versionen 17

2.6 Unterstützte Visual Studio-Versionen 18

2.7 Unterstützte Datenbanken 19

2.8 Funktionsumfang von Entity Framework Core 20

2.9 Funktionen, die dauerhaft entfallen 21

2.10 Funktionen, die Microsoft bald nachrüsten will 21

2.11 Hohe Priorität, aber nicht kritisch 22

2.12 Neue Funktionen in Entity Framework Core 23

2.13 Einsatzszenarien 24

3 Installation von Entity Framework Core **27**

3.1 Nuget-Pakete 27

3.2 Paketinstallation 30
3.3 Aktualisierung auf eine neue Version 34

4 Konzepte von Entity Framework Core 39
4.1 Vorgehensmodelle bei Entity Framework Core 39
4.2 Artefakte bei Entity Framework Core 41

5 Reverse Engineering bestehender Datenbanken 43
5.1 Reverse Engineering mit PowerShell-Befehlen 44
5.2 Codegenerierung 47
5.3 .NET Core-Tool 54
5.4 Schwächen des Reverse Engineering 56

6 Forward Engineering für neue Datenbanken 57
6.1 Regeln für die selbsterstellten Entitätsklassen 59
6.1.1 Properties 59
6.1.2 Datentypen 59
6.1.3 Beziehungen (Master-Detail) 59
6.1.4 Vererbung 61
6.1.5 Primärschlüssel 61
6.1.6 Beispiele 61
6.2 Regeln für die selbsterstellte Kontextklasse 64
6.2.1 Nuget-Pakete 64
6.2.2 Basisklasse 65
6.2.3 Konstruktor 65
6.2.4 Beispiel 65
6.2.5 Provider und Verbindungszeichenfolge 66
6.2.6 Eigene Verbindungen 67
6.2.7 Thread-Sicherheit 67
6.3 Regeln für die Datenbankschemagenerierung 67
6.4 Beispiel-Client 68
6.5 Anpassung per Fluent-API (OnModelCreating()) 69
6.6 Das erzeugte Datenmodell 71

7 Anpassung des Datenbankschemas 73
7.1 Persistente versus transiente Klassen 74
7.2 Namen im Datenbankschema 75
7.3 Reihenfolge der Spalten in einer Tabelle 75
7.4 Spaltentypen/Datentypen 76
7.5 Pflichtfelder und optionale Felder 77
7.6 Feldlängen 77
7.7 Primärschlüssel 78

7.8 Beziehungen und Fremdschlüssel ... 78
7.9 Optionale Beziehungen und Pflichtbeziehungen ... 79
7.10 Uni- und Bidirektionale Beziehungen ... 81
7.11 1:1-Beziehungen ... 82
7.12 Indexe festlegen ... 83
7.13 Weitere Syntaxoptionen für das Fluent-API ... 85
7.13.1 Sequentielle Konfiguration ... 85
7.13.2 Strukturierung durch Statement Lambdas ... 85
7.13.3 Strukturierung durch Unterroutinen ... 86
7.13.4 Strukturierung durch Konfigurationsklassen ... 87
7.14 Massenkonfiguration mit dem Fluent-API ... 88

8 Datenbankschemamigrationen ... 91
8.1 Anlegen der Datenbank zur Laufzeit ... 91
8.2 Schemamigrationen zur Entwicklungszeit ... 92
8.3 Befehle für die Schemamigrationen ... 92
8.4 ef.exe ... 93
8.5 Add-Migration ... 94
8.6 Update-Database ... 98
8.7 Script-Migration ... 99
8.8 Weitere Migrationsschritte ... 99
8.9 Migrationsszenarien ... 100
8.10 Weitere Möglichkeiten ... 101
8.11 Schemamigrationen zur Laufzeit ... 103

9 Daten lesen mit LINQ ... 105
9.1 Kontextklasse ... 105
9.2 LINQ-Abfragen ... 106
9.3 Schrittweises Zusammensetzung von LINQ-Abfragen ... 108
9.4 Repository-Pattern ... 109
9.5 Einsatz von var ... 110
9.6 LINQ-Abfragen mit Paging ... 111
9.7 Projektionen ... 113
9.8 Abfrage nach Einzelobjekten ... 114
9.9 Laden anhand des Primärschlüssels mit Find() ... 115
9.10 LINQ im RAM statt in der Datenbank ... 115
9.11 Umgehung für das GroupBy-Problem ... 119
9.11.1 Mapping auf Nicht-Entitätstypen ... 119
9.11.2 Entitätsklasse für die Datenbanksicht anlegen ... 120
9.11.3 Einbinden der Entitätsklasse in die Kontextklasse ... 120
9.11.4 Verwendung der Pseudo-Entitätsklasse ... 121

9.11.5 Herausforderung: Migrationen ... 121
9.11.6 Gruppierungen mit Datenbanksichten ... 123
9.12 Kurzübersicht über die LINQ-Syntax ... 123
9.12.1 Einfache SELECT-Befehle (Alle Datensätze) ... 125
9.12.2 Bedingungen (where) ... 125
9.12.3 Bedingungen mit Mengen (in) ... 126
9.12.4 Sortierungen (orderby) ... 126
9.12.5 Paging (Skip() und Take()) ... 127
9.12.6 Projektion ... 127
9.12.7 Aggregatfunktionen (Count(), Min(), Max(), Average(), Sum()) ... 128
9.12.8 Gruppierungen (GroupBy) ... 129
9.12.9 Einzelobjekte (SingleOrDefault(), FirstOrDefault()) ... 130
9.12.10 Verbundene Objekte (Include()) ... 131
9.12.11 Inner Join (Join) ... 132
9.12.12 Cross Join (Kartesisches Produkt) ... 133
9.12.13 Join mit Gruppierung ... 133
9.12.14 Unter-Abfragen (Sub-Select) ... 134

10 Objektbeziehungen und Ladestrategien ... 137
10.1 Standardverhalten ... 137
10.2 Kein Lazy Loading ... 138
10.3 Eager Loading ... 140
10.4 Explizites Nachladen (Explicit Loading) ... 143
10.5 Preloading mit Relationship Fixup ... 145
10.6 Details zum Relationship Fixup ... 150

11 Einfügen, Löschen und Ändern ... 151
11.1 Speichern mit SaveChanges() ... 151
11.2 Änderungsverfolgung auch für Unterobjekte ... 154
11.3 Das Foreach-Problem ... 155
11.4 Objekte hinzufügen mit Add() ... 156
11.5 Verbundene Objekte anlegen ... 158
11.6 Verbundene Objekte ändern/Relationship Fixup ... 161
11.7 Widersprüchliche Beziehungen ... 164
11.8 Zusammenfassen von Befehlen (Batching) ... 169
11.9 Objekte löschen mit Remove() ... 170
11.10 Löschen mit einem Attrappen-Objekt ... 171
11.11 Massenlöschen ... 172
11.12 Datenbanktransaktionen ... 173
11.13 Change Tracker abfragen ... 176
11.13.1 Zustand eines Objekts ... 176
11.13.2 Liste aller geänderten Objekte ... 178

12 Datenänderungskonflikte (Concurrency) ... 181
12.1 Rückblick ... 181
12.2 Im Standard keine Konflikterkennung ... 182
12.3 Optimistisches Sperren/Konflikterkennung ... 184
12.4 Konflikterkennung für alle Eigenschaften ... 185
12.5 Konflikteinstellung per Konvention ... 186
12.6 Fallweise Konflikteinstellung ... 187
12.7 Zeitstempel (Timestamp) ... 188
12.8 Konflikte auflösen ... 190
12.9 Pessimistisches Sperren bei Entity Framework Core ... 194

13 Protokollierung (Logging) ... 199
13.1 Verwendung der Erweiterungsmethode Log() ... 199
13.2 Implementierung der Log()-Erweiterungsmethode ... 201
13.3 Protokollierungskategorien ... 204

14 Asynchrone Programmierung ... 205
14.1 Asynchrone Erweiterungsmethoden ... 205
14.2 ToListAsync() ... 205
14.3 SaveChangesAsync() ... 207
14.4 ForeachAsync() ... 208

15 Dynamische LINQ-Abfragen ... 211
15.1 Schrittweises Zusammensetzen von LINQ-Abfragen ... 211
15.2 Expression Trees ... 213
15.3 Dynamic LINQ ... 216

16 Daten lesen und ändern mit SQL, Stored Procedures und Table Valued Functions ... 219
16.1 Abfragen mit FromSql() ... 219
16.2 Zusammensetzbarkeit von LINQ und SQL ... 221
16.3 Globale Abfragefilter bei SQL-Abfragen (ab Version 2.0) ... 223
16.4 Stored Procedures und Table Valued Functions ... 224
16.5 Globale Abfragefilter bei Stored Procedures und Table Valued Functions ... 225
16.6 Nicht-Entitätsklassen als Ergebnismenge ... 226
16.7 SQL-DML-Befehle ohne Resultset ... 228

17 Weitere Tipps und Tricks zum Mapping ... 229
17.1 Shadow Properties ... 229
17.1.1 Automatische Shadow Properties ... 229
17.1.2 Festlegung eines Shadow Property ... 230

17.1.3 Ausgabe aller Shadow Properties einer Entitätsklasse 230
17.1.4 Lesen und Ändern eines Shadow Property 231
17.1.5 LINQ-Abfragen mit Shadow Properties 232
17.1.6 Praxisbeispiel: Automatisches Setzen des Shadow Property bei jedem Speichern 232
17.2 Tabellenaufteilung (Table Splitting) 233
17.3 Berechnete Spalten (Computed Columns) 236
17.3.1 Automatisches SELECT 237
17.3.2 Praxistipp: Spalten mit einer Berechnungsformel anlegen 237
17.3.3 Spalten mit einer Berechnungsformel nutzen 239
17.3.4 Spalten mit einer Berechnungsformel beim Reverse Engineering 241
17.4 Standardwerte (Default Values) 242
17.4.1 Standardwerte beim Forward Engineering festlegen 242
17.4.2 Standardwerte verwenden 243
17.4.3 Praxistipp: Standardwerte schon beim Anlegen des Objekts vergeben 245
17.4.4 Standardwerte beim Reverse Engineering 246
17.5 Sequenzobjekte (Sequences) 247
17.5.1 Erstellen von Sequenzen beim Forward Engineering 247
17.5.2 Sequenzen im Einsatz 248
17.6 Alternative Schlüssel 251
17.6.1 Alternative Schlüssel definieren 252
17.6.2 Alternative Schlüssel im Einsatz 254
17.7 Kaskadierendes Löschen (Cascading Delete) 255
17.8 Abbildung von Datenbanksichten (Views) 258
17.8.1 Datenbanksicht anlegen 259
17.8.2 Entitätsklasse für die Datenbanksicht anlegen 259
17.8.3 Einbinden der Entitätsklasse in die Kontextklasse 259
17.8.4 Verwendung der Datenbanksicht 260
17.8.5 Herausforderung: Migrationen 260

18 Weitere Tipps und Tricks zu LINQ 263
18.1 Globale Abfragefilter (ab Version 2.0) 263
18.1.1 Filter definieren 263
18.1.2 Filter nutzen 264
18.1.3 Praxistipp: Filter ignorieren 265
18.2 Zukünftige Abfragen (Future Queries) 265

19 Leistungsoptimierung (Performance Tuning) 267
19.1 Vorgehensmodell zur Leistungsoptimierung bei Entity Framework Core . 267
19.2 Best Practices für Ihre eigenen Leistungstests 268
19.3 Leistungsvergleich verschiedener Dattenzugriffstechniken in .NET 268
19.4 Objektzuweisung optimieren 270

19.5 Massenoperationen ... 272
19.5.1 Einzellöschen ... 272
19.5.2 Optimierung durch Batching ... 273
19.5.3 Löschen ohne Laden mit Pseudo-Objekten ... 274
19.5.4 Einsatz von klassischem SQL anstelle des Entity Framework Core-APIs ... 275
19.5.5 Lamdba-Ausdrücke für Massenlöschen mit EFPlus ... 277
19.5.6 Massenaktualisierung mit EFPlus ... 278
19.6 Leistungsoptimierung durch No-Tracking ... 279
19.6.1 No-Tracking aktivieren ... 280
19.6.2 No-Tracking fast immer möglich ... 282
19.6.3 No-Tracking im änderbaren Datagrid ... 285
19.6.4 QueryTrackingBehavior und AsTracking() ... 286
19.6.5 Best Practices ... 288
19.7 Auswahl der besten Ladestrategie ... 288
19.8 Zwischenspeicherung (Caching) ... 289
19.8.1 MemoryCache ... 289
19.8.2 Abstraktion CacheManager ... 292
19.9 Second-Level-Caching mit EFPlus ... 296
19.9.1 Einrichten des Second-Level-Cache ... 297
19.9.2 Verwenden des Second-Level-Cache ... 297

20 Softwarearchitektur mit Entity Framework Core ... 301
20.1 Monolithisches Modell ... 301
20.2 Entity Framework Core als Datenzugriffsschicht ... 302
20.3 Reine Geschäftslogik ... 304
20.4 Geschäftsobjekt- und ViewModel-Klassen ... 305
20.5 Verteilte Systeme ... 306
20.6 Fazit ... 309

21 Zusatzwerkzeuge ... 311
21.1 Entity Framework Core Power Tools ... 311
21.1.1 Funktionsüberblick ... 311
21.1.2 Reverse Engineering mit Entity Framework Core Power Tools ... 312
21.1.3 Diagramme mit Entity Framework Core Power Tools ... 316
21.2 LINQPad ... 317
21.2.1 Aufbau von LINQPad ... 318
21.2.2 Datenquellen einbinden ... 319
21.2.3 LINQ-Befehle ausführen ... 322
21.2.4 Speichern ... 325
21.2.5 Weitere LINQPad-Treiber ... 325
21.2.6 Interaktive Programmcodeeingabe ... 326
21.2.7 Fazit zu LINQPad ... 327

21.3 Entity Developer ... 327
21.3.1 Reverse Engineering mit Entity Developer ... 328
21.3.2 Forward Engineering mit Entity Developer ... 339
21.4 Entity Framework Profiler ... 344
21.4.1 Einbinden des Entity Framework Profilers ... 346
21.4.2 Befehle überwachen mit Entity Framework Profiler ... 346
21.4.3 Warnungen vor potenziellen Problemen ... 349
21.4.4 Analysefunktionen ... 349
21.4.5 Fazit zu Entity Framework Profiler ... 350

22 Zusatzkomponenten ... 351
22.1 Entity Framework Plus (EFPlus) ... 351
22.2 Second-Level-Caching mit EFSecondLevelCache.Core ... 352
22.3 Objekt-Objekt-Mapping und AutoMapper ... 352
22.3.1 Objekt-Objekt-Mapping per Reflection ... 354
22.3.2 AutoMapper ... 357
22.3.3 Beispiel ... 358
22.3.4 Abbildungen konfigurieren ... 359
22.3.5 Abbildung ausführen mit Map() ... 360
22.3.6 Abbildungskonventionen ... 361
22.3.7 Profilklassen ... 363
22.3.8 Verbundene Objekte ... 363
22.3.9 Manuelle Abbildungen ... 364
22.3.10 Typkonvertierungen ... 366
22.3.11 Objektmengen ... 368
22.3.12 Vererbung ... 369
22.3.13 Generische Klassen ... 371
22.3.14 Zusatzaktionen vor und nach dem Mapping ... 374
22.3.15 Geschwindigkeit ... 375
22.3.16 Fazit zu AutoMapper ... 376

23 Praxislösungen ... 379
23.1 Entity Framework Core in einer ASP.NET Core-Anwendung ... 379
23.1.1 Das Fallbeispiel “MiracleList“ ... 379
23.1.2 Architektur ... 383
23.1.3 Entitätsklassen ... 385
23.1.4 Entity Framework Core-Kontextklasse ... 387
23.1.5 Lebensdauer der Kontextklasse in ASP.NET Core-Anwendungen ... 389
23.1.6 Geschäftslogik ... 390
23.1.7 WebAPI ... 398
23.1.8 Verwendung von Entity Framework Core per Dependency Injection ... 408
23.1.9 Praxistipp: Kontextinstanzpooling (DbContext Pooling) ... 411
23.2 Entity Framework Core in einer Universal Windows Platform App ... 412

23.2.1 Das Fallbeispiel “MiracleList Light“ 412
23.2.2 Architektur 413
23.2.3 Entitätsklassen 415
23.2.4 Entity Framework Core-Kontextklasse 416
23.2.5 Startcode 416
23.2.6 Erzeugte Datenbank 417
23.2.7 Datenzugriffscode 419
23.2.8 Benutzeroberfläche 423
23.3 Entity Framework Core in einer Xamarin-Cross-Platform-App 424
23.3.1 Das Fallbeispiel “MiracleList Light“ 424
23.3.2 Architektur 426
23.3.3 Entitätsklassen 428
23.3.4 Entity Framework Core-Kontextklasse 428
23.3.5 Startcode 430
23.3.6 Erzeugte Datenbank 430
23.3.7 Datenzugriffscode 430
23.3.8 Benutzeroberfläche 434
23.4 N:M-Beziehungen zu sich selbst 435

24 Quellen im Internet 441

Index 443

Vorwort

Liebe Leserinnen und Leser,

ich nutze Entity Framework in echten Softwareentwicklungsprojekten seit der allerersten Version, also seit der Version 1.0 von ADO.NET Entity Framework im Jahr 2008. Zuvor hatte ich einen selbstentwickelten Objekt-Relationalen Mapper in meinen Projekten verwendet. Entity Framework Core ist das Nachfolgeprodukt, das es seit 2016 gibt. Ich setzte seitdem auch (aber nicht ausschließlich) Entity Framework Core in der Praxis ein. Viele Projekte laufen noch mit dem klassischen Entity Framework.

Microsoft entwickelt Entity Framework Core inkrementell, d. h. die Versionen 1.x und 2.x stellen zunächst eine in vielen Punkten noch unvollständige Grundversion dar, die in den Folgeversionen dann komplettiert wird.

Dieses **inkrementelle Konzept** habe ich auch mit diesem Buch umgesetzt. Das Buch ist seit September 2016 in mehreren Versionen erschienen. Die vor Ihnen liegende Version dieses Buchs beschreibt nun alle Kernaspekte und viele Tipps und Tricks sowie Praxisszenarien zu Entity Framework Core. Ich plane, in Zukunft weitere Versionen dieses Buchs zu veröffentlichen, die die kommenden Versionen von Entity Framework Core beschreiben und auch weitere Tipps & Tricks sowie Praxisszenarien ergänzen.

Da Fachbücher leider heutzutage nicht nennenswert dazu beitragen können, den Lebensunterhalt meiner Familie zu bestreiten, ist dieses Projekt ein Hobby. Dementsprechend kann ich nicht garantieren, wann es Updates zu diesem Buch geben wird. Ich werde dann an diesem Buch arbeiten, wenn ich neben meinem Beruf als Softwarearchitekt, Berater und Dozent und meinen sportlichen Betätigungen noch etwas Zeit für das Fachbuchautorenhobby übrig habe.

Zudem möchte ich darauf hinweisen, dass ich natürlich keinen kostenfreien technischen Support zu den Inhalten dieses Buchs geben kann. Ich freue mich aber immer über konstruktives Feedback und Verbesserungsvorschläge. Bitte verwenden Sie dazu das Kontaktformular auf www.dotnet-doktor.de.

Wenn Sie **technische Hilfe** zu Entity Framework und Entity Framework Core oder anderen Themen rund um .NET, Visual Studio, Windows oder andere Microsoft-Produkte benötigen, stehe ich Ihnen im Rahmen meiner beruflichen Tätigkeit für die Firmen www.IT-Visions.de (Beratung, Schulung, Support) und 5Minds IT-Solutions GmbH & Co KG (Softwareentwicklung, siehe www.5minds.de) gerne zur Verfügung. Bitte wenden Sie sich für ein Angebot an das jeweilige Kundenteam.

Die Beispiele zu diesem Buch können Sie herunterladen auf der von mir ehrenamtlich betriebenen **Leser-Website** unter www.IT-Visions.de/Leser. Dort müssen Sie sich registrieren. Bei der Registrierung wird ein Losungswort abgefragt. Bitte geben Sie dort **Ascension** ein.

Herzliche Grüße aus Essen, dem Herzen der Metropole Ruhrgebiet

Holger Schwichtenberg

Über den Autor

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Gebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Leiter des Berater- und Dozententeams bei www.IT-Visions.de
- Leitung der Softwareentwicklung im Bereich Microsoft/.NET bei der 5minds IT-Solutions GmbH & Co. KG (www.5minds.de)
- Über 65 Fachbücher beim Carl Hanser Verlag, bei O'Reilly, Microsoft Press und Addison-Wesley sowie mehr als 950 Beiträge in Fachzeitschriften
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006-2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal heise.de (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z. B. Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, BASTA, BASTA-on-Tour, .NET Architecture Camp, Advanced Developers Conference, Developer Week, OOP, DOTNET

Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum, Container Conf)

- Zertifikate und Auszeichnungen von Microsoft:
- Microsoft Most Valuable Professional (MVP)
- Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
  - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten, SOA
  - Microsoft .NET Framework, Visual Studio, C#, Visual Basic
  - .NET-Architektur/Auswahl von .NET-Technologien
  - Einführung von .NET Framework und Visual Studio/Migration auf .NET
  - Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET, JavaScript/TypeScript und Webframeworks wie Angular
  - Enterprise .NET, verteilte Systeme/Webservices mit .NET, insbesondere Windows Communication Foundation und WebAPI
  - Relationale Datenbanken, XML, Datenzugriffsstrategien
  - Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework und EF Core
  - Windows PowerShell, PowerShell Core und Windows Management Instrumentation (WMI)
- Ehrenamtliche Community-Tätigkeiten:
  - Vortragender für die International .NET Association (INETA)
  - Betrieb diverser Community-Websites: www.dotnetframework.de, www.entwickler-lexikon.de, www.windows-scripting.de, www.aspnetdev.de u. a.
- Firmenwebsites: http://www.IT-Visions.de und http://www.5minds.de
- Weblog: http://www.dotnet-doktor.de
- Kontakt: E-Mail **buero@IT-Visions.de** sowie Telefon **0201-64 95 90-0**

1 Einleitung

1.1 Programmiersprache in diesem Buch

Als Programmiersprache kommt in diesem Buch C# zum Einsatz, weil dies die bei weitem am häufigsten verwendete Programmiersprache in .NET ist. Der Autor dieses Buchs programmiert in einigen Kundenprojekten .NET-Anwendungen zwar auch in Visual Basic .NET, leider bietet dieses Buch jedoch nicht den Raum, alle Listings in beiden Sprachen wiederzugeben.

Eine Sprachkonvertierung zwischen C# und Visual Basic .NET ist im WWW kostenfrei verfügbar auf der Website *http://converter.telerik.com*.

1.2 Fallbeispiele in diesem Buch

Die meisten Beispielprogrammcodes in diesem Buch drehen sich um das Fallbeispiel der fiktiven Fluggesellschaft „World Wide Wings“, abgekürzt WWWings (siehe auch *www.world-wide-wings.de*).

Bild 1.1 Logo der fiktiven Fluggesellschaft „World Wide Wings“

HINWEIS: In einzeln Unterkapitel werden andere Fallbeispiele verwendet (z. B. die Aufgabenverwaltung „MiracleList“). Diese Fallbeispiele werden dann in den jeweiligen Kapiteln erläutert.

1.2.1 Entitäten

Im Anwendungsfall „World Wide Wings“ geht es um folgende Entitäten:

- Flüge zwischen zwei Orten, bei denen die Orte bewusst nicht als eigene Entität modelliert wurden, sondern Zeichenketten sind (dies vereinfacht das Verständnis vieler Beispiele)
- Passagiere, die auf Flügen fliegen
- Mitarbeiter der Fluggesellschaft, die wiederum Vorgesetzte haben, die auch Mitarbeiter sind
- Piloten als eine Spezialisierung von Mitarbeitern
- Personen als Sammlung der gemeinsamen Eigenschaften für alle Menschen in diesem Beispiel. Personen gibt es aber nicht eigenständig, sondern nur in den Ausprägungen/Spezialisierungen Passagier, Mitarbeiter und Pilot. Im objektorientierten Sinne ist Person also eine abstrakte Basisklasse, die keine Instanzen besitzen kann, sondern nur der Vererbung dient.

Es gibt zwei Datenmodelle:

- Das etwas einfachere Modell #1 ist das Ergebnis klassischen relationalen Datenbankdesigns mit Normalisierung. Das Objektmodell daraus entsteht per Reverse Engineering.
- Modell #2 ist das Ergebnis des Forward Engineering mit Entity Framework Core aus einem Objektmodell. Zusätzlich gibt es hier weitere Entitäten (Persondetail, Flugzeugtyp und Flugzeugtypdetail), um weitere Modellierungsaspekte aufzeigen zu können.

In Modell #1 gibt es eine jeweils eigene Tabelle für Personen (auch wenn es keine eigenständigen Personen gibt), Mitarbeiter, Piloten und Passagiere. Diese Aufteilung entspricht den Klassen im Objektmodell.

In Modell #2 gibt es lediglich die Tabellen Passagiere und Mitarbeiter für diese vier Entitäten. Entity Framework Core ist derzeit etwas eingeschränkt und unterstützt das Konzept Table per Type (also eine eigenständige Tabelle für jede Klasse) nicht. Daher umfasst die Tabelle Passagiere auch alle Eigenschaften von Person. Die Tabelle Mitarbeiter umfasst neben den Personeneigenschaften die Eigenschaften der Entitäten Mitarbeiter und Pilot. In der Tabelle wird per Diskriminatorspalte unterschieden zwischen Datensätzen, die ein Mitarbeiter sind, und solchen, die ein Pilot sind. Entity Framework Core mischt hier die Konzepte Table per Concrete Type (TPC) und Table per Hierarchy (TPH). Einen dezidierten Einfluss auf diese Abbildung hat man in Entity Framework Core 1.x/2.0 noch nicht. Das klassische Entity Framework bietet hier mehr Optionen.

Abhängigkeitsarten sind hier:

- Ein Flug muss einen Piloten besitzen. Der Copilot ist optional.
- Ein Flug kann optional einen Flugzeutyp zugeordnet haben.
- Jede Person und damit auch jeder Pilot und Passagier muss ein Persondetail-Objekt besitzen.

In diesem Buch kommen beide Datenmodelle vor, teilweise auch in modifizierter Form, um bestimmte Szenarien (z. B. Datenbankschemamigrationen) aufzuzeigen.

Bitte beachten Sie, dass die Objektmodelle, die in diesem Buch zu den Datenmodellen erstellt werden, nicht das Idealbild eines Objektmodells darstellen können, denn Entity Framework Core unterstützt einige Mapping-Möglichkeiten wie z. B. das N:M-Mapping noch nicht. Das Objektmodell zum einfachen Datenmodell ist das automatisch von Entity Framework Core aus der Datenbank generierte Objektmodell (Reverse Engineering); es ist bewusst nicht verändert worden.

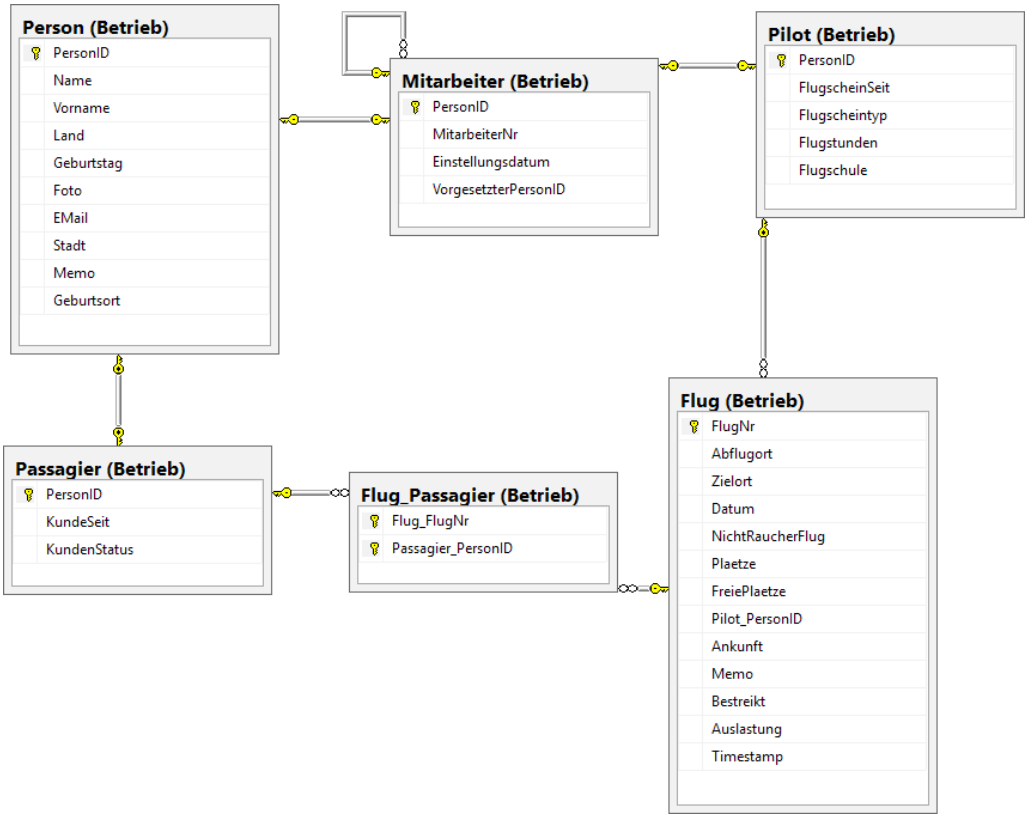


Bild 1.2 World Wide Wings-Datenmodell in der einfacheren Version

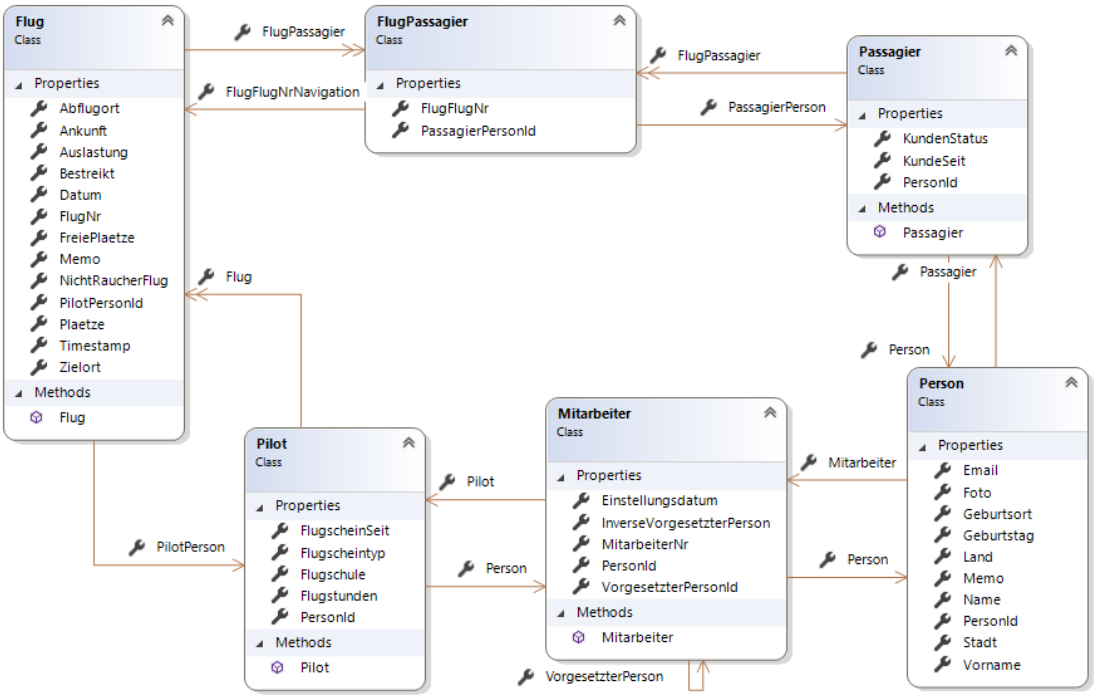


Bild 1.3 Objektmodell zum World Wide Wings-Datenmodell in der einfacheren Version

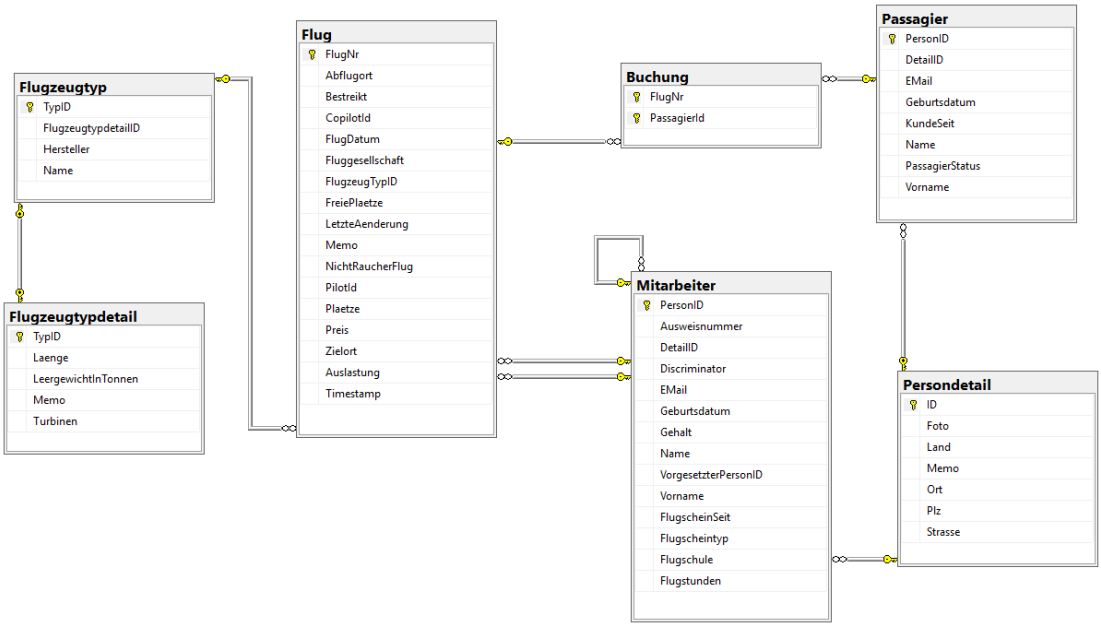


Bild 1.4 World Wide Wings-Datenmodell in der komplexeren Version

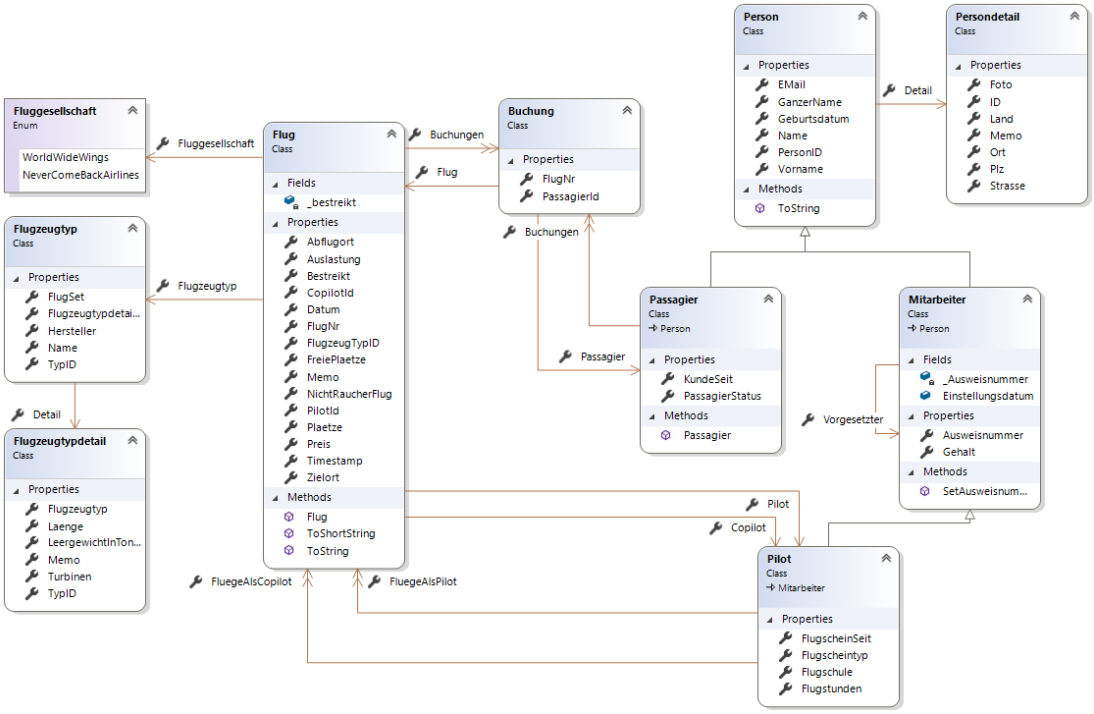


Bild 1.5 Objektmodell zum World Wide Wings-Datenmodell in der komplexeren Version

1.3 Anwendungsarten in diesem Buch

In diesem Buch erfolgen Bildschirmausgaben meist an der textbasierten Konsole in Konsolenanwendungen, denn dies ermöglicht die Fokussierung auf den Datenbankzugriff. Beim Einsatz von grafischen Benutzeroberflächen wie WPF, Windows Forms, ASP.NET Webforms oder ASP.NET MVC ist die Darstellung durch Datenbindung entkoppelt, das heißt man würde immer ein zweites Listing brauchen, um zu verstehen, dass die Datenzugriffe überhaupt liefern. Eingaben des Benutzers werden in den Konsolenbeispielen durch Variablen zu Beginn des Programmcodes simuliert.

Der Autor dieses Buchs führt seit vielen Jahren Schulungen und Beratungseinsätze im Bereich Datenzugriff durch und hat dabei die Erfahrung gemacht, dass Konsolenausgaben das didaktisch beste Instrument sind, da die Listings sonst sehr umfangreich und damit schlechter zu verstehen sind.

Natürlich ist die Konsolenausgabe in 99 % der Fälle der Softwareentwicklung nicht die gängige Praxis. Grafische Benutzeroberflächen sind Inhalt anderer Bücher, und die Datenbindung hat in der Regel keinen Einfluss auf die Form des Datenzugriffs. Dort, wo der Datenzugriff doch relevant ist, wird dieses Buch auch Datenbindungsbeispiele zeigen.

1.4 Hilfsroutinen zur Konsolenausgabe

Für die Bildschirmausgabe an der Konsole wird an mehreren Stellen nicht nur Console.WriteLine() verwendet, sondern auch Hilfsroutinen kommen zur Anwendung, die farbige Bildschirmausgaben erzeugen. Diese Hilfsroutinen in der Klasse *CUI* aus der ITV_DemoUtil.dll sind hier zum besseren Verständnis abgedruckt:

Listing 1.1 Klasse CUI mit Hilfsroutinen für die Bildschirmausgabe an der Konsole

```
using System;
using System.Runtime.InteropServices;
using System.Web;
using ITVisions.UI;
using System.Diagnostics;

namespace ITVisions
{
 /// <summary>
 /// Hilfsroutinen für Konsolen-UIs
 /// (C) Dr. Holger Schwichtenberg 2002-2017
 /// </summary>
 public static class CUI
 {
  public static bool IsDebug = false;
  public static bool IsVerbose = false;

  #region Ausgaben unter bestimmten Bedingungen
  /// <summary>
  /// Ausgabe an Console, Trace und Datei - nur wenn Anwendung im DEBUG-Modus
  /// </summary>
  public static void PrintDebug(object s)
  {
   PrintDebug(s, System.Console.ForegroundColor);
  }
  /// <summary>
  /// Ausgabe an Console, Trace und Datei - nur wenn Anwendung im VERBOSE-Modus
  /// </summary>
  public static void PrintVerbose(object s)
  {
   PrintVerbose(s, System.Console.ForegroundColor);
  }
  #endregion

  #region Ausgaben mit vordefinierten Farben
  public static void MainHeadline(string s)
  {
   Print(s, ConsoleColor.Black, ConsoleColor.Yellow);

  }
  public static void Headline(string s)
  {
   Print(s, ConsoleColor.Yellow);
  }
  public static void HeaderFooter(string s)
  {
```

```
 Console.ForegroundColor = ConsoleColor.Green;
 Console.WriteLine(s);
 Console.ForegroundColor = ConsoleColor.Gray;
}

public static void PrintSuccess(object s)
{
 Print(s, ConsoleColor.Green);

}

public static void PrintDebugSuccess(object s)
{
 PrintDebug(s, ConsoleColor.Green);

}

public static void PrintVerboseSuccess(object s)
{
 PrintVerbose(s, ConsoleColor.Green);

}

public static void PrintWarning(object s)
{
 Print(s, ConsoleColor.Cyan);

}

public static void PrintDebugWarning(object s)
{
 PrintDebug(s, ConsoleColor.Cyan);

}

public static void PrintVerboseWarning(object s)
{
 PrintVerbose(s, ConsoleColor.Cyan);
}

public static void PrintError(object s)
{
 Print(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void PrintDebugError(object s)
{
 PrintDebug(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void PrintVerboseError(object s)
{
 Print(s, ConsoleColor.White, ConsoleColor.Red);
}

public static void Print(object s)
{
 PrintInternal(s, null);
```

```
  }
  #endregion

  #region Ausgaben mit wählbarer Farbe

  /// <summary>
  /// Ausgabe an Console, Trace und Datei - nur wenn Anwendung im DEBUG Modus
  /// </summary>
  public static void Print(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
  {
   PrintInternal(s, farbe, hintergrundfarbe);
  }

  public static void PrintDebug(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
  {
   if (IsDebug || IsVerbose) PrintDebugOrVerbose(s, farbe, hintergrundfarbe);
  }

  public static void PrintVerbose(object s, ConsoleColor farbe)
  {
   if (!IsVerbose) return;
   PrintDebugOrVerbose(s, farbe);
  }
  #endregion

  #region Spezielle Ausgabe mit Zusatzdaten

  /// <summary>
  /// Ausgabe mit Thread-ID
  /// </summary>
  public static void PrintWithThreadID(string s, ConsoleColor c = ConsoleColor.White)
  {
   var ausgabe = String.Format("Thread #{0:00} {1:}: {2}", System.Threading.Thread.
CurrentThread.ManagedThreadId, DateTime.Now.ToLongTimeString(), s);
   CUI.Print(ausgabe, c);
  }

  /// <summary>
  /// Ausgabe mit Uhrzeit
  /// </summary>
  public static void PrintWithTime(object s, ConsoleColor c = ConsoleColor.White)
  {
   CUI.Print(DateTime.Now.Second + "." + DateTime.Now.Millisecond + ":" + s);
  }

  private static long count;
  /// <summary>
  /// Ausgabe mit fortlaufendem Zähler
  /// </summary>
  private static void PrintWithCounter(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
  {
   count += 1;
   s = String.Format("{0:0000}: {1}", count, s);
   CUI.Print(s, farbe, hintergrundfarbe);
  }
```

```
  #endregion

  #region interne Hilfsroutinen
  private static void PrintDebugOrVerbose(object s, ConsoleColor farbe, ConsoleColor?
hintergrundfarbe = null)
  {
   count += 1;
   s = String.Format("{0:0000}: {1}", count, s);
   Print(s, farbe, hintergrundfarbe);
   Debug.WriteLine(s);
   Trace.WriteLine(s);
   Trace.Flush();
  }

  /// <summary>
  /// Ausgabe an Console, Trace und Datei
  /// </summary>
  /// <param name="s"></param>
  //[DebuggerStepThrough()]
  private static void PrintInternal(object s, ConsoleColor? farbe = null,
ConsoleColor? hintergrundfarbe = null)
  {
   if (s == null) return;

   if (HttpContext.Current != null)
   {
    try
    {
     //Achtung: Keine Textausgaben in Ausgabestrom von ASP.NET Webservices und WCF-
Diensten und WebAPI schreiben!
     if (farbe != null)
     {
      HttpContext.Current.Response.Write("<span style='color:" + farbe.Value.
DrawingColor().Name + "'>");
     }
     if (!HttpContext.Current.Request.Url.ToString().ToLower().Contains(".asmx") &&
!HttpContext.Current.Request.Url.ToString().ToLower().Contains(".svc") &&
!HttpContext.Current.Request.Url.ToString().ToLower().Contains("/api/")) HttpContext.
Current.Response.Write(s.ToString() + "<br>");

     if (farbe != null)
     {
      HttpContext.Current.Response.Write("</span>");
     }
    }
    catch (Exception)
    {
    }
   }
   else
   {
    object x = 1;
    lock (x)
    {
     ConsoleColor alteFarbe = Console.ForegroundColor;
     ConsoleColor alteHFarbe = Console.BackgroundColor;
```

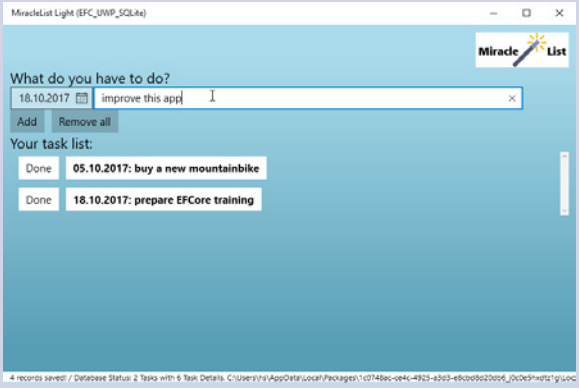
```
      if (farbe != null) Console.ForegroundColor = farbe.Value;
      if (hintergrundfarbe != null) Console.BackgroundColor = hintergrundfarbe.Value;

      //if (farbe.ToString().Contains("Dark")) Console.BackgroundColor = ConsoleColor.
White;
      //else Console.BackgroundColor = ConsoleColor.Black;

      Console.WriteLine(s);
      Console.ForegroundColor = alteFarbe;
      Console.BackgroundColor = alteHFarbe;
     }
    }
   }
   #endregion

   #region Position des Konsolenfensters setzen
   [DllImport("kernel32.dll", ExactSpelling = true)]
   private static extern IntPtr GetConsoleWindow();
   private static IntPtr MyConsole = GetConsoleWindow();

   [DllImport("user32.dll", EntryPoint = "SetWindowPos")]
   public static extern IntPtr SetWindowPos(IntPtr hWnd, int hWndInsertAfter, int x,
int Y, int cx, int cy, int wFlags);

   // Setze Position des Konsolenfensters ohne Größe
   public static void SetConsolePos(int xpos, int ypos)
   {
    const int SWP_NOSIZE = 0x0001;
    SetWindowPos(MyConsole, 0, xpos, ypos, 0, 0, SWP_NOSIZE);
   }

   // Setze Position des Konsolenfensters mit Größe
   public static void SetConsolePos(int xpos, int ypos, int w, int h)
   {
    SetWindowPos(MyConsole, 0, xpos, ypos, w, h, 0);
   }
   #endregion
  }
}
```

1.5 Programmcodebeispiel zum Download

Die Beispiele zu diesem Buch können Sie als Visual Studio-Projekte herunterladen auf der Leser-Website unter www.IT-Visions.de/Leser. Dort müssen Sie sich einmalig registrieren. Bei der Registrierung wird ein **Losungswort** abgefragt, das Sie als Käufer dieses Buchs ausweist. Bitte geben Sie dort **Ascension** ein. Durch die Registrierung erhalten Sie ein persönliches Kennwort per E-Mail zugesendet, das Sie dann für die Anmeldung nutzen können.

Bitte beachten Sie, dass nicht jede einzelne Zeile Programmcode, die Sie in diesem Buch finden, in den herunterladbaren Projekten enthalten sein kann. Die Projekte bilden funkti-

onierende Lösungen. In diesem Buch warden auch alternative Lösungen für Einzelfälle diskutiert, die nicht unbedingt zu einer Gesamtlösung passen.

Dateiname	Inhalt
EFC_Reverse.rar	Beispiel aus dem Kapitel „Reverse Engineering“. Das Paket enthält auch die SQL-Skripte für das einfachere „World Wide Wings“-Datenmodell inkl. Testdaten.
EFC_Forward.rar	„World Wide Wings“-Beispiel aus dem Kapitel „Forward Engineering“
EFC_Hauptbeispielsammlung.rar	Beispiele aller anderen Kapitel, die auf einer erweiterten Variante des „World Wide Wings“-Beispiels aus dem Kapitel „Forward Engineering“ basieren. Hierin enthalten ist das Objektmodell, aus dem das komplexere Datenmodell zur Entwicklungs- oder Laufzeit angelegt werden kann. Testdaten lassen sich durch einen Datengenerator generieren, der im Quellcode enthalten ist.
EFC_UWP_SQLite.rar und EFC_Xamarin_SQLite.rar	Beispielanwendung: Einfacher Merkzettel „MiracleList Light“ als Universal App für Windows 10 und Cross-Platform-App für iOS, Android und Windows 10. Die App speichert Daten mit Hilfe von Entity Framework Core in SQLite. 
EFC_Countries_NMSelf.rar	Ländergrenzen-Beispiel aus dem Kapitel „Praxislösungen“

2 Was ist Entity Framework Core?

Entity Framework Core ist ein Objekt-Relationaler Mapper (ORM) für .NET (.NET Framework, .NET Core, Mono und Xamarin). Entity Framework Core ist eine Neuimplementierung des „ADO.NET Entity Framework“.

Zusammen mit .NET Core Version 1.0 und ASP.NET Core Version 1.0 ist auch Entity Framework Core Version 1.0 am 27. Juni 2016 erstmals erschienen. Die neue Variante von Microsofts Objekt-Relationalem Mapper enthält in den ersten Versionen allerdings noch einige gravierende Lücken, die den Einsatzbereich beschränken. Die Version 2.0 ist am 14. August 2017 erschienen. Version 2.1 ist in Arbeit.

2.1 Was ist ein Objekt-Relationaler Mapper?

In der Datenbankwelt sind relationale Datenbanken vorherschend, in der Programmierwelt sind es Objekte. Zwischen den beiden Welten gibt es erhebliche semantische und syntaktische Unterschiede, die man unter dem Begriff „Impedance Mismatch“ (zu deutsch: Unverträglichkeit, vgl. [*https://dict.leo.org/englisch-deutsch/impedance%20mismatch*]) oder „Semantic Gap“ (zu deutsch: semantische Lücke) zusammenfasst.

Kern des objektorientierten Programmierens (OOP) ist die Arbeit mit Objekten als Instanzen von Klassen im Hauptspeicher. Die meisten Anwendungen beinhalten dabei auch die Anforderung, in Objekten gespeicherte Daten dauerhaft zu speichern, insbesondere in Datenbanken. Grundsätzlich existieren objektorientierte Datenbanken (OODB), die direkt in der Lage sind, Objekte zu speichern. Allerdings haben objektorientierte Datenbanken bisher nur eine sehr geringe Verbreitung. Der vorherrschende Typus von Datenbanken sind relationale Datenbanken, die Datenstrukturen jedoch anders abbilden als Objektmodelle.

Um die Handhabung von relationalen Datenbanken in objektorientierten Systemen natürlicher zu gestalten, setzt die Software-Industrie seit Jahren auf O/R-Mapper (auch: OR-Mapper oder ORM geschrieben). O steht dabei für objektorientiert und R für relational. Diese Werkzeuge bilden demnach Konzepte aus der objektorientierten Welt, wie Klassen, Attribute oder Beziehungen zwischen Klassen, auf entsprechende Konstrukte der relationalen Welt, wie zum Beispiel Tabellen, Spalten und Fremdschlüssel, ab. Der Entwickler kann

somit in der objektorientierten Welt verbleiben und den O/R-Mapper anweisen, bestimmte Objekte, welche in Form von Datensätzen in den Tabellen der relationalen Datenbank vorliegen, zu laden bzw. zu speichern. Wenig interessante und fehleranfällige Aufgaben wie das manuelle Erstellen von INSERT-, UPDATE- oder DELETE-Anweisungen übernimmt der O/R-Mapper hierbei ebenfalls, was zu einer weiteren Entlastung des Entwicklers führt.

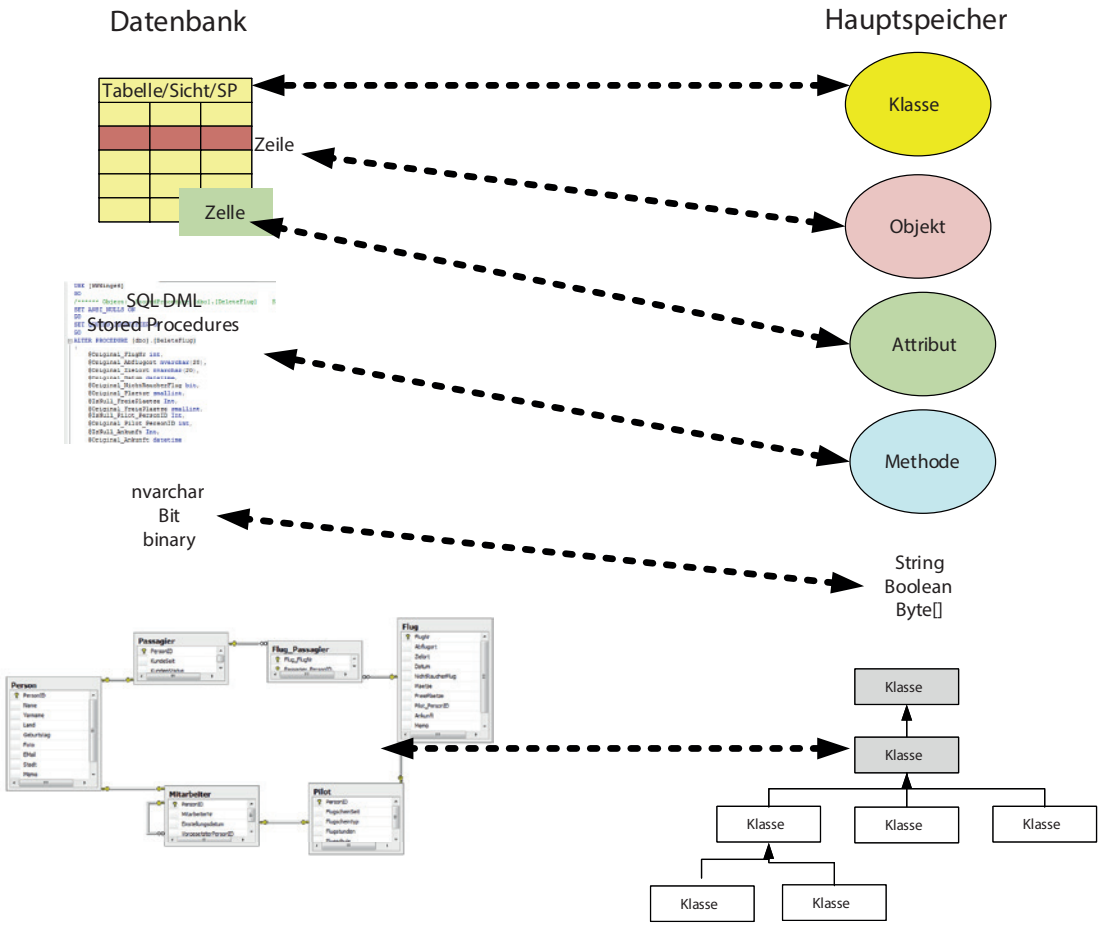


Bild 2.1 Beim ORM bildet man Konstrukte der OOP-Welt auf die relationale Welt ab.

Zwei besonders hervorstechende Unterschiede zwischen Objektmodell und Relationenmodell sind N:M-Beziehungen und Vererbung. Während man in einem Objektmodell eine N:M-Beziehung zwischen Objekten durch eine wechselseitige Objektmenge abbilden kann, benötigt man in der relationalen Datenbank eine Zwischentabelle. Vererbung kennen relationale Datenbanken gar nicht. Hier gibt es verschiedene Möglichkeiten der Nachbildung, doch dazu später mehr.

2.2 ORM in der .NET-Welt

Wenn ein .NET-Entwickler aus einer Datenbank mit einem DataReader oder DataSet Daten einliest, dann betreibt er noch kein OR Mapping. DataReader und DataSet sind zwar .NET-Objekte, aber diese verwalten nur Tabellenstrukturen. DataReader und DataSet sind aus der Sicht eines Objektmodells untypisierte, unspezifische Container. Erst wenn ein Entwickler spezifische Klassen für die in den Tabellen gespeicherten Strukturen definiert und die Inhalte aus DataSet oder DataReader in diese spezifischen Datenstrukturen umkopiert, betreibt er OR Mapping. Solch ein „händisches OR Mapping“ ist für den Lesezugriff (gerade bei sehr breiten Tabellen) eine sehr aufwändige, mühselige und eintönige Programmierarbeit. Will man dann Änderungen in den Objekten auch noch wieder speichern, wird die Arbeit allerdings zur intellektuellen Herausforderung. Denn man muss erkennen können, welche Objekte verändert wurden, da man sonst ständig alle Daten aufs Neue speichert, was in Mehrbenutzerumgebungen ein Unding ist.

Während in der Java-Welt das ORM-Werkzeug schon sehr lange zu den etablierten Techniken gehört, hat Microsoft diesen Trend lange verschlafen bzw. es nicht vermocht, ein geeignetes Produkt zur Marktreife zu führen. ADO.NET in .NET 1.0 bis 3.5 enthielt keinen ORM, sondern beschränkte sich auf den direkten Datenzugriff und die Abbildung zwischen XML-Dokumenten und dem relationalen Modell.

Viele .NET-Entwickler haben sich daher daran gesetzt, diese Arbeit mit Hilfsbibliotheken und Werkzeugen zu vereinfachen. Dies war die Geburtsstunde einer großen Vielfalt von ORM-Werkzeugen für .NET. Dabei scheint es so, dass viele .NET-Entwickler das geflügelte Wort, dass ein Mann in seinem Leben einen Baum gepflanzt, ein Kind gezeugt und ein Haus gebaut haben sollte, um den Punkt „einen OR-Mapper geschrieben“ ergänzt haben (wobei der Autor dieses Buchs sich davon auch nicht freisprechen kann, weil er ebenfalls einen OR-Mapper geschrieben hat). Anders ist die Vielfalt der ähnlichen Lösungen kaum erklärbar. Neben den öffentlich bekannten ORM-Werkzeugen für .NET findet man in den Unternehmen zahlreiche hauseigene Lösungen.

Bekannte öffentliche ORM für .NET von Drittanbietern (z. T. Open Source) sind:

- nHibernate
- Telerik Data Access (alias Open Access)
- Genome
- LLBLGen Pro
- Wilson
- Subsonic
- OBJ.NET
- .NET Data Objects (NDO)
- Dapper
- PetaPoco
- Massive
- Developer Express XPO

Neben den aktiven Entwicklern von ORM-Werkzeugen für .NET und den passiven Nutzern gibt eine noch größere Fraktion von Entwicklern, die ORM bisher nicht einsetzen. Meist herrscht Unwissenheit, die auch nicht aufgearbeitet wird, denn es herrscht das Motto „Wenn Microsoft es nicht macht, ist es auch nicht wichtig!“

Mit LINQ-to-SQL und dem ADO.NET Entity Framework sowie Entity Framework bietet Microsoft selbst jedoch inzwischen sogar drei verschiedene Produkte an. Der Softwarekonzern hat aber inzwischen verkündet, dass sich die Weiterentwicklungsbemühungen allein auf das Entity Framework Core konzentieren.

2.3 Versionsgeschichte von Entity Framework Core

Die folgende Tabelle zeigt die Versionsgeschichte von Entity Framework Core.

Version	Downloads	Last updated
2.0.0 (current version)	145,895	2 months ago
2.0.0-preview2-final	18,978	3 months ago
2.0.0-preview1-final	33,319	5 months ago
1.1.3	12,940	16 days ago
1.1.2	315,456	5 months ago
1.1.1	303,386	7 months ago
1.1.0	724,401	11/16/2016
1.1.0-preview1-final	19,298	10/24/2016
1.0.5	304	16 days ago
1.0.4	7,101	5 months ago
1.0.3	61,481	7 months ago
1.0.2	43,864	10 months ago
1.0.1	298,195	9/13/2016
1.0.0	410,041	6/27/2016
1.0.0-rc2-final	70,867	5/16/2016

Bild 2.2 Entity Framework Core-Versionsgeschichte
[Quelle: *https://www.nuget.org/packages/Microsoft.EntityFrameworkCore*]

Versionsnummernänderungen an der dritten Stelle (z. B. 1.0.1 und 1.0.2) enthalten nur Fehlerbehebungen. Bei Versionsnummernänderungen an der zweiten Stelle sind auch neue Funktionen enthalten. In diesem Buch wird darauf hingewiesen, wenn eine Funktion besprochen wird, die eine bestimmte Versionsnummer voraussetzt.

HINWEIS: Die endgültige Version der Entity Framework Core-Werkzeuge für Entity Framework Core 1.x ist erst am 6.3.2017 im Rahmen von Entity Framework Core 1.1.1 und Visual Studio 2017 erschienen. Zuvor gab es nur „Preview"-Versionen. Seit Entity Framework Core 2.0 werden die Werkzeuge immer mit den neuen Produktreleases ausgeliefert.

2.4 Unterstützte Betriebssysteme

Genau wie die anderen Produkte der Core-Produktfamilie ist das Entity Framework Core (früherer Name: Entity Framework 7.0) ebenfalls plattformunabhängig. Die Core-Variante des etablierten Objekt-Relationalen Mappers läuft nicht nur auf dem .NET „Full" Framework, sondern auch auf .NET Core und Mono inklusive Xamarin. Damit kann man Entity Framework Core auf Windows, Windows Phone/Mobile, Linux, MacOS, iOS und Android nutzen.

2.5 Unterstützte .NET-Versionen

Entity Framework Core 1.x läuft auf .NET Core 1.x, .NET Framework ab Version 4.5.1, Mono ab Version 4.6, Xamarin.iOS ab Version 10, Xamarin Android ab Version 7.0 und der Windows Univeral Platform (UWP).

Entity Framework Core 2.0 basiert auf .NET Standard 2.0 und setzt daher eine der folgenden .NET-Implementierungen voraus:

- .NET Core 2.0 (oder höher)
- .NET Framework 4.6.1 (oder höher)
- Mono 5.4 (oder höher)
- Xamarin.iOS 10.14 (oder höher)
- Xamarin.Mac 3.8 (oder höher)
- Xamarin.Android 7.5 (oder höher)
- Universal Windows Platform (UWP) 10.0.16299 (oder höher)

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework (with .NET Core 1.x SDK)	4.5	4.5	4.5.1	4.6	4.6.1	4.6.2		
.NET Framework (with .NET Core 2.0 SDK)	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299
Windows	8.0	8.0	8.1					
Windows Phone	8.1	8.1	8.1					
Windows Phone Silverlight	8.0							

Bild 2.3 Implementierungen von .NET Standard
[Quelle: *https://docs.microsoft.com/de-de/dotnet/standard/library*]

HINWEIS: Microsoft begründet die Beschränkung auf .NET Standard in Entity Framework Core 2.0 in [*https://github.com/aspnet/Announcements/issues/246*]. Unter anderem kann dadurch die Größe der Nuget-Pakete deutlich reduziert werden.

2.6 Unterstützte Visual Studio-Versionen

Für die Nutzung von Entity Framework Core 2.0 benötigt man zwingend Visual Studio 2017 Update 3 oder höher, auch wenn man mit dem klassischen .NET Framework programmiert, da Visual Studio nur mit diesem Update .NET Standard 2.0 kennt und versteht, dass .NET Framework 4.6.1 und höher Implementierungen von .NET Standard 2.0 sind.

Wenn man für .NET Core programmiert, benötigt man für Entity Framework Core 1.x Visual Studio 2017, (die Werkzeuge für Visual Studio 2015 sind veraltert und werden von Microsoft nicht mehr aktualisiert). Für Entity Framework Core 1.x in Verbindung mit dem klassischen .NET Framework reicht auch eine ältere Visual Studio-Version.

2.7 Unterstützte Datenbanken

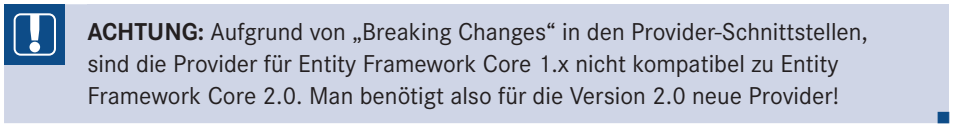
Die folgende Tabelle zeigt die von Entity Framework Core durch Microsoft (SQL Server, SQL Compact und SQLite von Microsoft) und Drittanbieter (PostgreSQL, DB2, Oracle, MySQL u. a.) unterstützten Datenbankmanagementsysteme.

Auf Mobilgeräten mit Xamarin bzw. im Rahmen von Windows 10 Universal Platform Apps konnte Entity Framework Core 1.x nur lokale Datenbanken (SQLite) ansprechen. Mit der Einführung von .NET Standard 2.0 steht nun der Microsoft SQL Server-Client auch auf Xamarin und der Windows 10 Universal Platform (ab dem Herbst 2017 Creators Update) zur Verfügung.

Die geplante Unterstützung für NoSQL-Datenbanken wie Redis und Azure Table Storage ist in Version 1.x/2.x von Entity Framework Core noch nicht enthalten. Es gibt aber für MongoDB ein Entwicklungsprojekt auf Github [*https://github.com/crhairr/EntityFrameworkCore.MongoDb*].

Tabelle 2.1 Verfügbare Datenbanktreiber für Entity Framework Core

Datenbank	Anbieter / Preis	URL
Microsoft SQL Server	Microsoft / kostenfrei	*www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer*
Microsoft SQL Server Compact 3.5	Microsoft / kostenfrei	*www.nuget.org/packages/EntityFrameworkCore.SqlServerCompact35*
Microsoft SQL Server Compact 4.0	Microsoft / kostenfrei	*www.nuget.org/packages/EntityFrameworkCore.SqlServerCompact40*
SQLite	Microsoft / kostenfrei	*www.nuget.org/packages/Microsoft.EntityFrameworkCore.Sqlite*
In-Memory	Microsoft / kostenfrei	*www.nuget.org/packages/Microsoft.EntityFrameworkCore.InMemory*
MySQL	Oracle / kostenfrei	*www.nuget.org/packages/MySQL.Data.EntityFrameworkCore*
PostgreSQL	Open Source-Team *npgsql.org* / kostenfrei	*www.nuget.org/packages/Npgsql.EntityFrameworkCore.PostgreSQL*
DB2	IBM / kostenfrei	*www.nuget.org/packages/EntityFramework.IBMDataServer*
MySQL, Oracle, PostgreSQL, SQLite, DB2, Salesforce, Dynamics CRM, SugarCRM, Zoho CRM, QuickBooks, FreshBooks, MailChimp, ExactTarget, Bigcommerce, Magento	Devart / kostenpflichtig (99 bis 299 Dollar pro Treiberart)	*www.devart.com/purchase.html#dotConnect*



ACHTUNG: Aufgrund von „Breaking Changes“ in den Provider-Schnittstellen, sind die Provider für Entity Framework Core 1.x nicht kompatibel zu Entity Framework Core 2.0. Man benötigt also für die Version 2.0 neue Provider!

2.8 Funktionsumfang von Entity Framework Core

Die Abbildung visualisiert, dass Entity Framework Core (gelb) gegenüber dem bisherigen Entity Framework (blau, aktuelle Version 6.1.3) einige neue Funktionen enthält (Bereich, der nur gelb, aber nicht blau ist). Es gibt aber auch einige Bereiche, die nur blau und nicht gelb sind: Das sind die Funktionen, die in Entity Framework 6.1.3 enthalten sind, aber nicht in Entity Framework Core 1.x/2.0. Microsoft wird einige Funktionen davon in den kommenden Versionen von Entity Framework Core nachrüsten, andere Funktionen werden für immer entfallen.

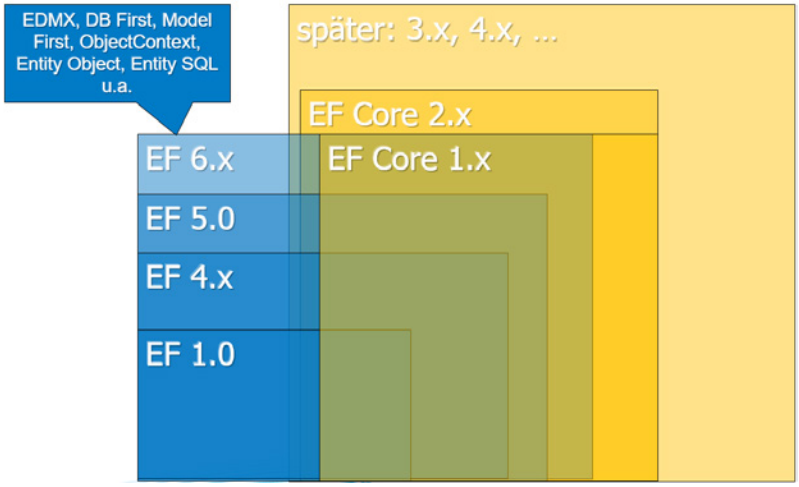


Bild 2.4 Funktionsumfang des bisherigen Entity Framework im Vergleich zu Entity Framework Core. Links zeigt eine Sprechblase einige Features, die dauerhaft entfallen sind.

2.9 Funktionen, die dauerhaft entfallen

Folgende Funktionen hat Microsoft grundsätzlich gestrichen:

- Die Vorgehensweise Database First und Model First. Es gibt 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 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 gehört 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.

2.10 Funktionen, die Microsoft bald nachrüsten will

In der Roadmap für Entity Framework-Core [*https://github.com/aspnet/EntityFramework/wiki/Roadmap*] dokumentiert Microsoft-Entwickler Rowan Miller, welche Features in Entity Framework-Core fehlen, die man „bald“ nachrüsten will. Dabei ist dies nicht mit einem konkreten Zeitplan hinterlegt. Bemerkenswert ist, dass Microsoft einige dieser Funktionen selbst als „kritisch“ bezeichnet. Zu diesen „kritischen“ fehlenden Funktionen gehören:

- Entity Framework Core unterstützt nur den Zugriff auf Tabellen, nicht aber auf Views (Sichten) in der Datenbank. Man kann Views nur nutzen, wenn man den View sowie den Programmcode manuell erstellt und den View wie eine Tabelle behandelt.
- Stored Procedures können bisher nur zum Abfragen von Daten (SELECT), nicht aber zum Einfügen (INSERT), Aktualisieren (UPDATE) und Löschen (DELETE) verwendet werden.

- Einige LINQ-Befehle werden derzeit nicht in der Datenbank, sondern im RAM ausgeführt. Dazu gehört auch der group by-Operator, d. h. bei allen Gruppierungen werden alle Datensätze aus der Datenbank ins RAM gelesen und dort gruppiert, was bei allen Tabellen (außer sehr kleinen) zu einer katastrophalen Performance führt.
- Es gibt weder ein automatisches Lazy Loading noch ein explizites Nachladen im Entity Framework Core-API. Aktuell kann der Entwickler verbundene Datensätze nur direkt mitladen (Eager Loading) oder mit separaten Befehlen nachladen.
- Direktes SQL und Stored Procedures können nur genutzt werden, wenn sie Entitätstypen zurückliefern. Andere Typen werden bisher nicht unterstützt.
- Reverse Engineering bestehender Datenbanken kann man bisher nur von der Kommandozeile bzw. der Nuget-Konsole in Visual Studio starten. Den GUI-basierten Assistenten gibt es nicht mehr.
- Es gibt auch kein „Update Model from Database“ für bestehende Datenbanken, d. h. nach einem Reverse Engineering einer Datenbank muss der Entwickler Datenbankschemaänderungen im Objektmodell manuell nachtragen oder das ganze Objektmodell neu generieren. Diese Funktion gab es aber auch bisher schon bei Code First nicht, sondern nur bei Database First.
- Komplexe Typen (Complex Types), also Klassen, die keine eigene Entität, sondern Teil einer anderen Entität darstellen, gibt es nicht.

2.11 Hohe Priorität, aber nicht kritisch

In einer zweiten Liste nennt Microsoft weitere Funktionen, die sie nicht als kritisch ansehen, die aber dennoch „hohe Priorität“ haben:

- Es gibt bisher keine grafische Visualisierung eines Objektmodells, wie das bislang bei EDMX möglich war.
- Einige der bisher vorhandenen Typkonvertierungen, z. B. zwischen XML und String, gibt es noch nicht.
- Die Geo-Datentypen Geography und Geometry von Microsoft SQL Server werden bisher nicht unterstützt.
- Entity Framework Core unterstützt keine N:M-Abbildungen: Bisher muss der Entwickler dies mit zwei 1:N-Abbildung und einer Zwischenentität analog zur Zwischentabelle in der Datenbank nachbilden.
- Table per Type wird bislang nicht als Vererbungsstrategie unterstützt. Entity Framework Core verwendet TPH, wenn es für die Basisklasse ein DBSet<T> gibt, sonst TPC. TPC kann man nicht explizit konfigurieren.
- Das Befüllen der Datenbank mit Daten im Rahmen der Migration (Seed()-Funktion) ist nicht möglich.

- Die mit Entity Framework 6.0 eingeführten Command Interceptors, mit denen ein Softwareentwickler von Entity Framework zur Datenbank gesendete Befehle vor und nach der Ausführung in der Datenbank beeinflussen kann, gibt es noch nicht.

Einige Punkte auf dieser High Priority-Liste von Microsoft sind zudem auch neue Features, die Entity Framework 6.1.3 selbst (noch) gar nicht beherrscht:

- Festlegung von Bedingungen für mitzuladene Datensätze beim Eager Loading (Eager Loading Rules)
- Unterstützung für E-Tags.
- Unterstützung für Nicht-Relationale Datenspeicher („NoSQL“) wie Azure Table Storage und Redis.

Diese Priorisierung stammt aus der Sicht von Microsoft. Der Autor dieses Buchs würde auf Basis seiner Praxiserfahrung einige Punkte anders priorisieren, zum Beispiel die N:M-Abbildung als „kritisch“ hochstufen: Eine Nachbildung von N:M durch zwei 1:N-Beziehungen im Objektmodell ist zwar möglich, macht aber den Programmcode komplexer. Die Migration von bestehenden Entity Framework-Lösungen zu Entity Framework Core wird damit sehr erschwert.

Das gilt auch für die fehlende Unterstützung von Table per Type-Vererbung: Auch hier muss bestehender Programmcode umfangreich geändert werden. Und auch für neue Anwendungen mit einem neuen Datenbankschema und Forward Engineering gibt es ein Problem: Wenn die Vererbung erst mal mit TPH oder TPC realisiert ist, muss man aufwändig die Daten im Datenbankschema umschichten, wenn man später doch auf TPH setzen will.

Außerdem fehlen in Microsofts Listen auch Features wie etwa die Validierung von Entitäten, die unnötige Roundtrips zur Datenbank ersparen kann, wenn schon im RAM klar ist, dass die Entität die erforderlichen Bedingungen nicht erfüllt.

2.12 Neue Funktionen in Entity Framework Core

Entity Framework Core kann insbesondere mit folgenden Vorteilen gegenüber dem Vorgänger auftrumpfen:

- Entity Framework Core läuft nicht nur in Windows, Linux und MacOS, sondern auch auf Mobilgeräten mit Windows 10, iOS und Android. Auf den Mobilgeräten ist freilich lediglich ein Zugriff auf lokale Datenbanken (z. B. SQLite) vorgesehen.
- Entity Framework Core bietet eine höhere Ausführungsgeschwindigkeit – insbesondere beim Datenlesen (dabei wird fast die Leistung wie beim handgeschriebenen Umkopieren von Daten aus einem DataReader-Objekt in ein typisiertes .NET-Objekt erreicht).
- Projektionen mit Select() können nun direkt auf Entitätsklassen abgebildet werden. Der Umweg über anonyme .NET-Objekte ist nicht mehr notwendig.

- Per „Batching“ fasst Entity Framework Core nun INSERT-, DELETE- und UPDATE-Operationen zu einem Rundgang zum Datenbankmanagementsystem zusammen, statt jeden Befehl einzeln zu senden.
- Standardwerte für Spalten in der Datenbank werden nun sowohl beim Reverse Engineering als auch beim Forward Engineering unterstützt.
- Zur Schlüsselgenerierung sind neben den klassischen Autowerten nun auch neuere Verfahren wie Sequenzen erlaubt.
- Als „Shadow Properties“ bezeichnet Entity Framework Core den jetzt möglichen Zugriff auf Spalten der Datenbanktabelle, für die es kein Attribut in der Klasse gibt.

2.13 Einsatzszenarien

Angesichts dieser langen Liste von fehlenden Funktionen stellt sich die Frage, ob und wofür Entity Framework Core in der Version 1.x/2.0 überhaupt zu gebrauchen ist.

Das Haupteinsatzgebiet liegt auf den Plattformen, wo Entity Framework bisher gar nicht lief: Windows Phone/Mobile, Android, iOS, Linux und MacOS/X.

- Universal Windows Platform (UWP) Apps und Xamarin Apps können nur Entity Framework Core verwenden.
- Wenn man eine neue ASP.NET Core-Anwendung entwickeln will und diese nicht auf .NET „Full“ Framework, sondern .NET Core basieren soll, führt kein Weg an Entity Framework Core vorbei, denn das bisherige Entity Framework 6.1.3 läuft nicht auf .NET Core. Allerdings gibtes für ASP.NET Core auch den Weg, als Basis das .NET Framework 4.6.x zu verwenden, sodass man dann auch Entity Framework 6.x nutzen kann.

Für Projekte auf anderen Plattformen gilt:

- Eine aufwändige Migration bestehenden Programmcodes werden die verbesserten Features und die höhere Leistung von Entity Framework Core meist nicht rechtfertigen.
- Aber in neuen Projekten können Entwickler schon jetzt Entity Framework Core als performante Zukunftstechnik einsetzen und ggf. als Zwischenlösung für Lücken parallel dort noch das bisherige Entity Framework nutzen.

Ein Szenario, in dem der Einsatz von Entity Framework Core auf dem Webserver empfohlen werden kann, ist das Offline-Szenario, bei dem es auf dem Mobilgerät eine lokale Kopie der Serverdatenbank geben soll. In diesem Fall kann man auf dem Client und dem Server mit demselben Datenzugriffscode arbeiten: Der Client verwendet Entity Framework Core für den Zugriff auf SQLite und der Webserver denselben Entity Framework Core-Programmcode für den Zugriff auf einen Microsoft SQL Server (siehe folgende Abbildung).

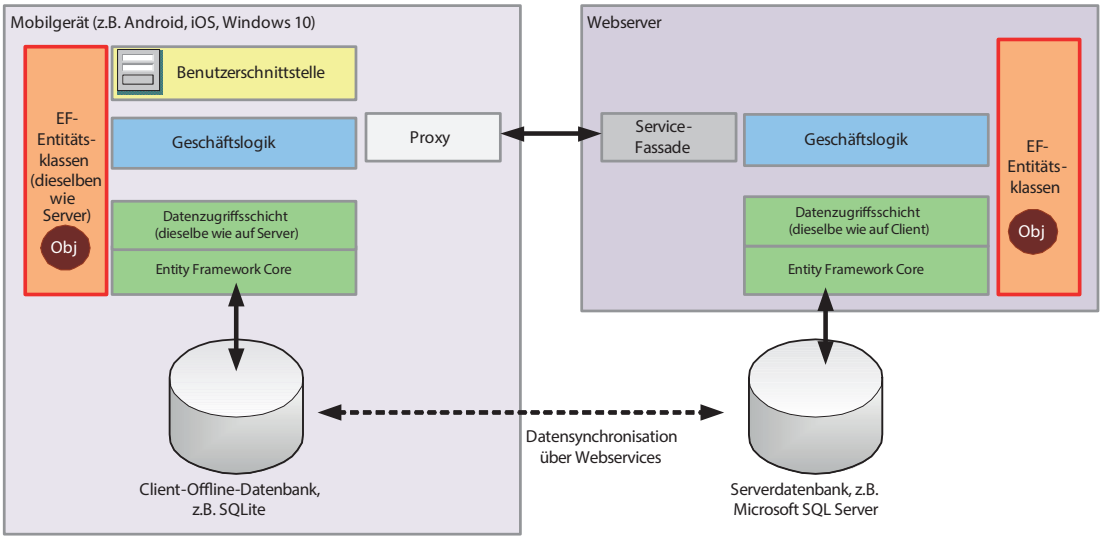


Bild 2.5 Teilen der Datenzugriffsschicht zwischen Mobilgerät und Webserver mit Entity Framework Core

Wenig Sinn macht zum jetzigen Zeitpunkt eine Migration von Entity Framework 6.1.3 auf Entity Framework Core. Dafür sprechen könnte nur die bessere Performance von Entity Framework Core in einigen Fällen. Man muss aber den hohen Umstellungsaufwand im Programmcode beachten.

3 Installation von Entity Framework Core

Für Entity Framework Core gibt es keine Setup.exe. Entity Framework Core installiert man in einem Projekt über Nuget-Pakete.

3.1 Nuget-Pakete

Entity Framework Core besteht im Gegensatz zum klassischen Entity Framework aus mehreren Nuget-Paketen. Die folgende Tabelle zeigt nur die Wurzelpakete. Deren Abhängigkeiten, zu denen Nuget die zugehörigen Pakete dann automatisch mitinstalliert, sind hier nicht genannt.

Tabelle 3.1 Die wichtigsten auf *nuget.org* verfügbaren Pakete für Entity Framework Core

Datenbankmanagementsystem	Nuget-Paket, das zur Laufzeit benötigt wird	Nuget-Paket, das zur Entwicklungszeit für Reverse Engineering oder Schemamigrationen benötigt wird
Microsoft SQL Server Express, Standard, Enterprise, Developer, LocalDB (ab Version 2008)	Microsoft.EntityFrameworkCore.SqlServer	Microsoft.EntityFrameworkCore.Tools Microsoft.EntityFrameworkCore.SqlServer (für EF Core 2.0) Microsoft.EntityFrameworkCore.SqlServer.Design (für EF Core 1.x)
Microsoft SQL Server Compact 3.5	EntityFrameworkCore.SqlServerCompact35	Nicht verfügbar
Microsoft SQL Server Compact 4.0	EntityFrameworkCore.SqlServerCompact40	Nicht verfügbar
SQLite	Microsoft.EntityFrameworkCore.Sqlite	Microsoft.EntityFrameworkCore.Tools Microsoft.EntityFrameworkCore.Sqlite (für EF Core 2.0) Microsoft.EntityFrameworkCore.Sqlite.Design (für EF Core 1.x)

Tabelle 3.1 Die wichtigsten auf *nuget.org* verfügbaren Pakete für Entity Framework Core *(Fortsetzung)*

Datenbankmanagementsystem	Nuget-Paket, das zur Laufzeit benötigt wird	Nuget-Paket, das zur Entwicklungszeit für Reverse Engineering oder Schemamigrationen benötigt wird
In-Memory	Microsoft.EntityFrameworkCore.InMemory	Nicht verfügbar – macht keinen Sinn
PostgreSQL	Npgsql.EntityFrameworkCore.PostgreSQL	Microsoft.EntityFrameworkCore.Tools Npgsql.EntityFrameworkCore.PostgreSQL (für EF Core 2.0) Npgsql.EntityFrameworkCore.PostgreSQL.Design (für EF Core 1.x)
MySQL	MySQL.Data.EntityFrameworkCore	MySQL.Data.EntityFrameworkCore.Design

In Entity Framework Core Version 2.0 hat Microsoft den Zuschnitt der Pakete abermals geändert. Vorher gab es zu jedem Treiber zwei Pakete, eins davon mit „Design“ im Namen. Die „Design“-Pakete wurden aufgelöst und in die eigentlichen Treiber-Assemblies integriert.