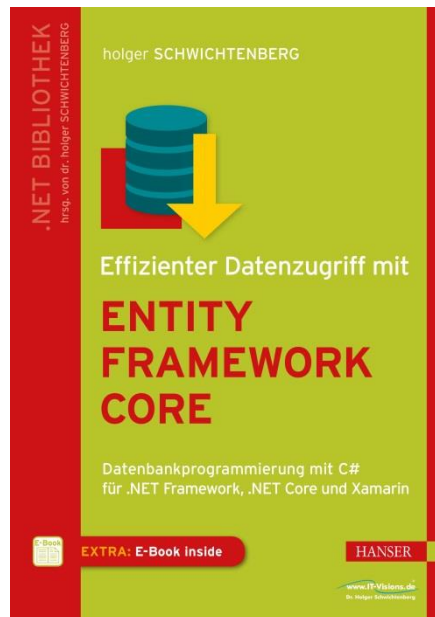


# HANSER



## Leseprobe

zu

## Effizienter Datenzugriff mit Entity Framework Core von Holger Schwichtenberg

ISBN (Buch): 978-3-446-44898-8

ISBN (E-Book): 978-3-446-44978-7

Weitere Informationen und Bestellungen unter

<http://www.hanser-fachbuch.de/>

sowie im Buchhandel

© Carl Hanser Verlag, München

# Inhalt

<b>Vorwort</b> .....	<b>XV</b>
<b>Über den Autor</b> .....	<b>XVII</b>
<b>1 Einleitung</b> .....	<b>1</b>
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
<b>2 Was ist Entity Framework Core?</b> .....	<b>13</b>
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
<b>3 Installation von Entity Framework Core</b> .....	<b>27</b>
3.1 Nuget-Pakete .....	27

3.2	Paketinstallation .....	30
3.3	Aktualisierung auf eine neue Version .....	34
<b>4</b>	<b>Konzepte von Entity Framework Core .....</b>	<b>39</b>
4.1	Vorgehensmodelle bei Entity Framework Core .....	39
4.2	Artefakte bei Entity Framework Core .....	41
<b>5</b>	<b>Reverse Engineering bestehender Datenbanken .....</b>	<b>43</b>
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
<b>6</b>	<b>Forward Engineering für neue Datenbanken .....</b>	<b>57</b>
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
<b>7</b>	<b>Anpassung des Datenbankschemas .....</b>	<b>73</b>
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 Unterrouninen .....	86
7.13.4	Strukturierung durch Konfigurationsklassen .....	87
7.14	Massenkonfiguration mit dem Fluent-API .....	88
<b>8</b>	<b>Datenbankschemamigrationen .....</b>	<b>91</b>
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
<b>9</b>	<b>Daten lesen mit LINQ .....</b>	<b>105</b>
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
<b>10</b>	<b>Objektbeziehungen und Ladestrategien .....</b>	<b>137</b>
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
<b>11</b>	<b>Einfügen, Löschen und Ändern .....</b>	<b>151</b>
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

<b>12</b>	<b>Datenänderungskonflikte (Concurrency)</b> .....	<b>181</b>
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
<b>13</b>	<b>Protokollierung (Logging)</b> .....	<b>199</b>
13.1	Verwendung der Erweiterungsmethode Log() .....	199
13.2	Implementierung der Log()-Erweiterungsmethode .....	201
13.3	Protokollierungskategorien .....	204
<b>14</b>	<b>Asynchrone Programmierung</b> .....	<b>205</b>
14.1	Asynchrone Erweiterungsmethoden .....	205
14.2	ToListAsync() .....	205
14.3	SaveChangesAsync() .....	207
14.4	ForeachAsync() .....	208
<b>15</b>	<b>Dynamische LINQ-Abfragen</b> .....	<b>211</b>
15.1	Schrittweises Zusammensetzen von LINQ-Abfragen .....	211
15.2	Expression Trees .....	213
15.3	Dynamic LINQ .....	216
<b>16</b>	<b>Daten lesen und ändern mit SQL, Stored Procedures und Table Valued Functions</b> .....	<b>219</b>
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
<b>17</b>	<b>Weitere Tipps und Tricks zum Mapping</b> .....	<b>229</b>
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
<b>18</b>	<b>Weitere Tipps und Tricks zu LINQ .....</b>	<b>263</b>
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
<b>19</b>	<b>Leistungsoptimierung (Performance Tuning) .....</b>	<b>267</b>
19.1	Vorgehensmodell zur Leistungsoptimierung bei Entity Framework Core .	267
19.2	Best Practices für Ihre eigenen Leistungstests .....	268
19.3	Leistungsvergleich verschiedener Datenzugriffstechniken 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	Lambda-Ausdrücke für Massnlöschen 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
<b>20</b>	<b>Softwarearchitektur mit Entity Framework Core .....</b>	<b>301</b>
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
<b>21</b>	<b>Zusatzwerkzeuge .....</b>	<b>311</b>
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
<b>22</b>	<b>Zusatzkomponenten</b>	<b>351</b>
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
<b>23</b>	<b>Praxislösungen</b>	<b>379</b>
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	Datenzugriffscodes .....	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	Datenzugriffscodes .....	430
23.3.8	Benutzeroberfläche .....	434
23.4	N:M-Beziehungen zu sich selbst .....	435
<b>24</b>	<b>Quellen im Internet .....</b>	<b>441</b>
<b>Index</b>	<b>.....</b>	<b>443</b>

# 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 Fachbuchautoren-hobby 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](http://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](http://www.IT-Visions.de) (Beratung, Schulung, Support) und 5Minds IT-Solutions GmbH & Co KG (Softwareentwicklung, siehe [www.5minds.de](http://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](http://www.IT-Visions.de/Leser). Dort müssen Sie sich registrieren. Bei der Registrierung wird ein Lösungswort 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](http://www.IT-Visions.de)
- Leitung der Softwareentwicklung im Bereich Microsoft/.NET bei der 5minds IT-Solutions GmbH & Co. KG ([www.5minds.de](http://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](http://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](http://www.dotnetframework.de), [www.entwicklerlexikon.de](http://www.entwicklerlexikon.de), [www.windows-scripting.de](http://www.windows-scripting.de), [www.aspnetdev.de](http://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](mailto:buero@IT-Visions.de) sowie Telefon **0201-64 95 90-0**

# 10

## Objektbeziehungen und Ladestrategien

Entity Framework Core unterstützt im Gegensatz zum bisherigen Entity Framework nur drei statt vier Ladestrategien. Und auch die Realisierung des Eager Loading entspricht nicht dem aus dem Vorgänger gewohnten Verhalten.

### ■ 10.1 Standardverhalten

Entity Framework Core beschränkt sich in der Standardeinstellung bei einer Abfrage (wie das bisherige Entity Framework auch) darauf, die tatsächlich angeforderten Objekte zu laden und lädt verbundene Objekte nicht automatisch mit. Eine LINQ-Abfrage wie

```
List<Flug> liste = (from x in ctx.FlugSet
                  where x.Abflugort == ort &&
                        x.FreiePlaetze > 0
                  orderby x.Datum, x.Abflugort
                  select x).ToList();
```

lädt in Bezug auf das in der nächsten Abbildung dargestellte Objektmodell also wirklich nur Instanzen der Klasse Flug. Damit in der Datenbank verbundene Piloten-, Buchungen- oder Flugzeugtypen-Datensätze werden nicht automatisch mitgeladen. Das Mitladen verbundener Datensätze (in der Fachsprache „Eager Loading“ genannt) wäre auch keine gute Idee für die Standardeinstellung, denn hier würden dann ggf. Daten geladen, die später gar nicht gebraucht werden. Zudem haben die verbundenen Datensätze bekanntlich selbst wieder Beziehungen, z. B. Buchungen zu Passagieren. Passagiere haben aber auch Buchungen auf anderen Flügen. Wenn man rekursiv alle diese in Beziehungen stehenden Datensätze mitladen würde, dann würde man im Beispiel des Objektmodells in der nächsten Abbildung mit großer Wahrscheinlichkeit fast alle Datensätze ins RAM laden, denn viele Passagiere sind über gemeinsame Flüge mit anderen Passagieren verbunden. Eager Loading wäre als Standardeinstellung also nicht gut.

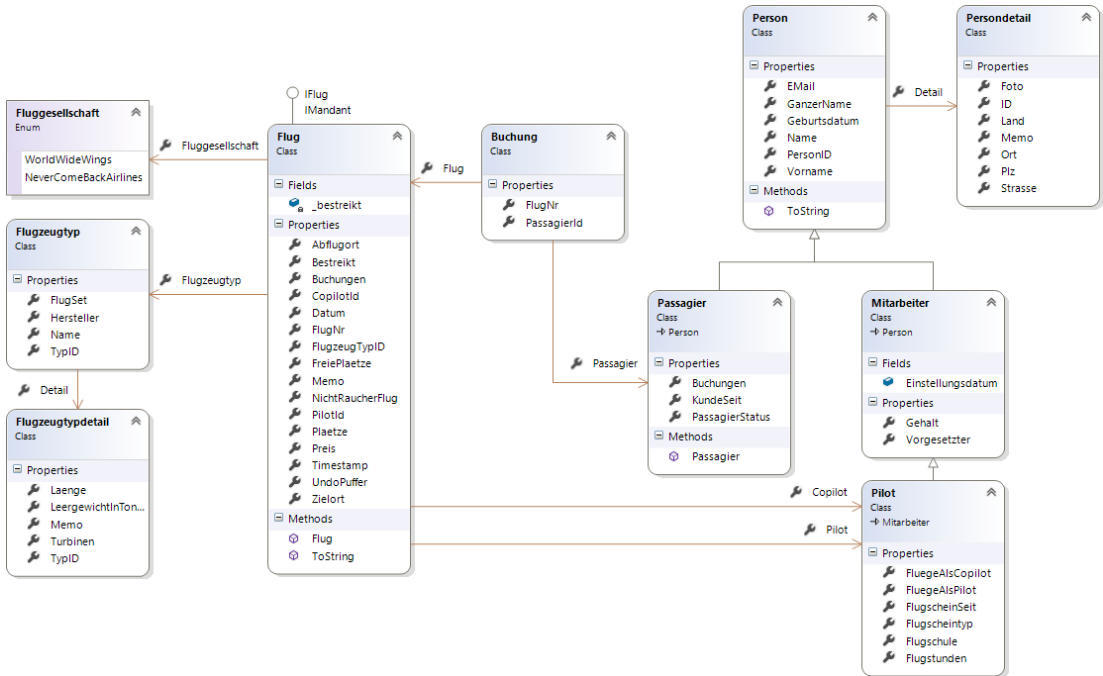


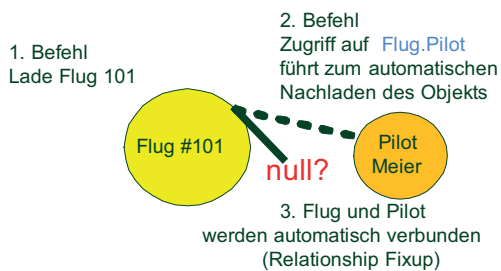
Bild 10.1 Objektmodell für die Verwaltung von Flügen, Passagieren und Piloten

## 10.2 Kein Lazy Loading

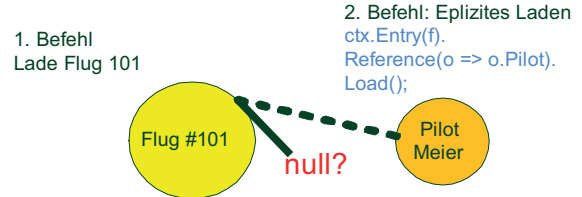
Das bisherige Entity Framework unterstützt vier Strategien für das Laden verbundener Objekte: Lazy Loading automatisch, explizites Laden (Explicit Loading), Eager Loading und Preloading mit Relationship Fixup (siehe Abbildung). In Entity Framework Core 1.0 standen die beiden Varianten des Lazy Loading nicht zur Verfügung. In Entity Framework Core Version 1.1 gibt es zumindest das explizite Laden, nicht aber das automatische Lazy Loading. Automatisches Lazy Loading ist die Standardeinstellung im bisherigen Entity Framework.



## Automatisches Lazy Loading

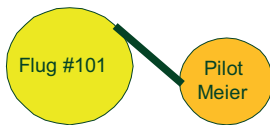


## Explizites Laden



## Eager Loading

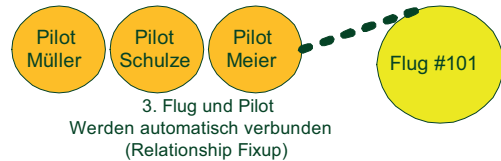
1. Befehl  
Lade Flug 101 mit Pilot: `Include()`



## Preloading

1. Befehl  
Lade alle Piloten

2. Befehl  
Lade Flug 101



© Dr. Holger Schwichtenberg, www.IT-Visions.de, 2013-2017

**Bild 10.2** Ladestrategien im Entity Framework 1.0 bis 6.x. Entity Framework Core unterstützt auch in Version 1.1 nur drei Strategien; Lazy Loading fehlt.

Das einleitende Beispiel (siehe Unterkapitel „10.1 Standardverhalten“) würde beim bisherigen Entity Framework im ersten Schritt nur den explizit angeforderten Flug laden, dann aber in den folgenden Programmcodezeilen via Lazy Loading auch die Piloten- und Copiloteninformation (mit jeweils auch deren anderen Flügen) sowie die Buchungen mit den Passagierdaten laden. Entity Framework würde hier nacheinander eine Vielzahl von SELECT-Befehlen zur Datenbank senden. Wie viele es genau sind, hängt von der Anzahl der Passagiere auf diesem Flug ab.

Bei Entity Framework Core liefert obiger Programmcode aber weder Pilot, Copilot noch Passagiere. Es wird bei „Anzahl Passagiere auf diesem Flug“ eine 0 angezeigt. Microsoft hat das Lazy Loading für Entity Framework Core noch nicht implementiert.

Lazy Loading beinhaltet eine besondere Implementierungsherausforderung, denn der OR-Mapper muss jeglichen Zugriff auf alle Objektreferenzen abfangen, um hier bei Bedarf die verbundenen Objekte nachladen zu können. Dieses Abfangen erfolgt durch die Verwendung bestimmter Klassen für Einzelreferenzen und Mengenklassen.

Im klassischen Entity Framework gibt es sowohl die Möglichkeit, Lazy Loading durch Verwendung bestimmter Lazy Loading-fähiger Klassen als auch durch meist unsichtbare Runtime Proxy-Objekte zu realisieren. Beides ist in Entity Framework Core noch nicht vorgesehen.

Lazy Loading wird man in Entity Framework Core aber gerade dann vermissen, wenn es keinen Sinn macht, verbundene Datensätze schon vorab zu laden. Typisches Beispiel ist eine Master-Detail-Ansicht auf dem Bildschirm. Wenn es viele Master-Datensätze gibt, wäre es verschwendete Zeit, zu jedem Master-Datensatz auch schon die Detail-Datensätze zu laden. Vielmehr wird man immer nur die Detail-Datensätze zu dem Master-Datensatz, den der Benutzer gerade angeklickt hat, anfordern. Während man an dieser Stelle im bisherigen

Entity Framework Core die Master-Detail-Darstellung via Lazy Loading ohne weiteren Programmcode beim Anklicken des Master-Datensatzes realisieren konnte, muss man in Entity Framework Core das Klicken abfangen und die Detail-Datensätze explizit laden.

**Listing 10.1** Vergeblicher Versuch des Zugriffs auf verbundene Objekte in Entity Framework Core

```

/// <summary>
/// Liefert keine Pilot-, Buchungs- und Passagierinformationen
/// </summary>
public static void Demo_LazyLoading()
{
    CUI.Headline(nameof(Demo_LazyLoading));

    using (var ctx = new WWWingsContext())
    {
        // Lade nur den Flug
        var f = ctx.FlugSet.SingleOrDefault(x => x.FlugNr == 101);

        Console.WriteLine($"Flug Nr {f.FlugNr} von {f.Abflugort} nach {f.Zielort} hat
{f.FreiePlaetze} freie Plätze!");
        if (f.Pilot != null) Console.WriteLine($"Pilot: {f.Pilot.Name} hat {f.Pilot.
FluegeAlsPilot.Count} Flüge als Pilot!");
        else Console.WriteLine("Kein Pilot zugewiesen!");
        if (f.Copilot != null) Console.WriteLine($"Copilot: {f.Copilot.Name} hat
{f.Copilot.FluegeAlsCopilot.Count} Flüge als Copilot!");
        else Console.WriteLine("Kein Copilot zugewiesen!");

        Console.WriteLine("Anzahl Passagiere auf diesem Flug: " + f.Buchungen.Count);

        Console.WriteLine("Passagiere auf diesem Flug:");
        foreach (var b in f.Buchungen)
        {
            Console.WriteLine("- Passagier #{0}: {1} {2}", b.Passagier.PersonID,
b.Passagier.Vorname, b.Passagier.Name);
        }
    }
}

```

## ■ 10.3 Eager Loading

Genau wie das bisherige Entity Framework unterstützt auch Entity Framework Core das Eager Loading. Die Syntax hat sich jedoch ein wenig geändert.

Im bisherigen Entity Framework konnte man bei Include() in der ersten Version nur eine Zeichenkette mit dem Namen einer Navigationseigenschaften angeben; die Zeichenkette wurde nicht vom Compiler geprüft. Ab der dritten Version (Versionsnummer 4.1) kam dann die Möglichkeit hinzu, eines Lambda-Audrucks für die Navigationseigenschaften anstelle der Zeichenkette anzugeben. Für Ladepfade über mehrere Ebenen musste man die Lambda-Ausdrücke verschachteln und auch die Select()-Methode verwenden.

In Entity Framework Core gibt es weiterhin beide Varianten, jedoch wurde die Syntax für die Lambda-Ausdrücke etwas modifiziert: Anstelle der Verschachtelung mit `Select()` tritt `ThenInclude()` analog zu `OrderBy()` und `ThenOrderBy()` für die Sortierung über mehrere Spalten. Das nächste Listing zeigt das Eager Loading eines Flugs mit folgenden verbundenen Daten:

- Buchungen und zu jeder Buchung den Passagierinformationen: `Include(b => b.Buchungen).ThenInclude(p => p.Passagier)`
- Pilot und zu dem Pilot die Liste seiner weiteren Flüge als Pilot: `Include(b => b.Pilot).ThenInclude(p => p.FluegeAlsPilot)`
- Copilot und zu dem Copilot die Liste seiner weiteren Flüge als Copilot: `Include(b => b.Copilot).ThenInclude(p => p.FluegeAlsCopilot)`

**Listing 10.2** Mit Eager Loading kann man in Entity Framework Core die verbundenen Objekte nutzen.

```

/// <summary>
/// Liefert Pilot-, Buchungs- und Passagierinformationen
/// </summary>
public static void Demo_EagerLoading()
{
    CUI.Headline("Demo_EagerLoading");

    using (var ctx = new WWingsContext())
    {
        // Lade den Flug und einige verbundene Objekte via Eager Loading
        var f = ctx.FlugSet
            .Include(b => b.Buchungen).ThenInclude(p => p.Passagier)
            .Include(b => b.Pilot).ThenInclude(p => p.FluegeAlsPilot)
            .Include(b => b.Copilot).ThenInclude(p => p.FluegeAlsCopilot)
            .SingleOrDefault(x => x.FlugNr == 101);

        Console.WriteLine($"Flug Nr {f.FlugNr} von {f.Abflugort} nach {f.Zielort} hat
        {f.FreiePlaetze} freie Plätze!");
        if (f.Pilot != null) Console.WriteLine($"Pilot: {f.Pilot.Name} hat {f.Pilot.
        FluegeAlsPilot.Count} Flüge als Pilot!");
        else Console.WriteLine("Kein Pilot zugewiesen!");
        if (f.Copilot != null) Console.WriteLine($"Copilot: {f.Copilot.Name} hat
        {f.Copilot.FluegeAlsCopilot.Count} Flüge als Copilot!");
        else Console.WriteLine("Kein Copilot zugewiesen!");

        Console.WriteLine("Anzahl Passagiere auf diesem Flug: " + f.Buchungen.Count);
        Console.WriteLine("Passagiere auf diesem Flug:");
        foreach (var b in f.Buchungen)
        {
            Console.WriteLine("- Passagier #{0}: {1} {2}", b.Passagier.PersonID,
            b.Passagier.Vorname, b.Passagier.Name);
        }
    }
}

```

Das Listing liefert die Ausgabe in der nächsten Bildschirmabbildung: Sowohl die Informationen zu Pilot und Copilot als auch die Liste der gebuchten Passagiere steht zur Verfügung.



**ACHTUNG:** Der Compiler prüft bei `Include()` und `ThenInclude()` nur, ob die Klasse ein entsprechendes Property oder Field besitzt. Er prüft nicht, ob es sich dabei auch um eine Navigationseigenschaft zu einer anderen Entitätsklasse handelt. Wenn es keine Navigationseigenschaft ist, kommt es erst zur Laufzeit zum Fehler: *The property xy is not a navigation property of entity type ,ab'. The ,Include(string) method can only be used with a ,.' separated list of navigation property names.*

```
Demo_EagerLoading
WWingsContext: OnConfiguring
WWingsContext: OnModelCreating
Flug Nr 101 von Seattle nach Moskau hat 129 freie Plätze!
Pilot: Koch hat 10 Flüge als Pilot!
Copilot: Stoiber hat 6 Flüge als Copilot!
Anzahl Passagiere auf diesem Flug: 8
Passagiere auf diesem Flug:
- Passagier #12: Niklas Bauer
- Passagier #15: Jan Schäfer
- Passagier #17: Leon Klein
- Passagier #47: Lukas Schneider
- Passagier #59: Laura Wagner
- Passagier #67: Marie Weber
- Passagier #87: Leonie Schäfer
- Passagier #98: Anna Schmidt
```

**Bild 10.3** Ausgabe des Listings

Allerdings gibt es noch einen entscheidenden Unterschied zum bisherigen Entity Framework: Während Entity Framework in den Versionen 1.0 bis 6.x hier nur einen einzigen, sehr großen SELECT-Befehl zum Datenbankmanagementsystem gesendet hätte, entscheidet sich Entity Framework Core, die Abfrage in vier Teile zu teilen (siehe nächste Abbildung):

- Erst wird der Flug geladen mit Join auf die Mitarbeiter-Tabelle, in der sich auch die Pilotinformation befindet (ein Table per Hierarchy-Mapping).
- Im zweiten Schritt lädt Entity Framework Core die sechs anderen Flüge des Copiloten.
- Im dritten Schritt lädt Entity Framework Core die zehn anderen Flüge des Piloten.
- Im letzten Schritt lädt Entity Framework Core die achten gebuchten Passagiere.

Diese Strategie kann schneller sein, als einen großen SELECT-Befehl auszuführen, der ein großes Resultset, in dem Datensätze doppelt vorkommen, liefert und das der OR-Mapper dann auseinandernehmen und von den Duplikaten bereinigen muss. Die Strategie getrennter SELECT-Befehle von Entity Framework Core kann aber auch langsamer sein, da jeder Rundgang zum Datenbankmanagementsystem Zeit kostet. Im bisherigen Entity Framework hatte der Softwareentwickler die freie Wahl, wie groß er eine Eager Loading-Anweisung zuschneiden will und wo er getrennt laden will. In Entity Framework Core wird der Softwareentwickler an dieser Stelle bevormundet und verliert dabei die Kontrolle über die Anzahl der Rundgänge zum Datenbankmanagementsystem.

```

SELECT TOP(2) [b].[FlugNr], [b].[Abflugort], [b].[Bestreikt], [b].[CopilotId], [b].[Flugdatum], [b].[Fluggesellschaft], ...
SELECT [fo].[FlugNr], [fo].[Abflugort], [fo].[Bestreikt], [fo].[CopilotId], [fo].[Flugdatum], [fo].[Fluggesellschaft], ...
SELECT [f].[FlugNr], [f].[Abflugort], [f].[Bestreikt], [f].[CopilotId], [f].[Flugdatum], [f].[Fluggesellschaft], [f].[Fl...
SELECT [bo].[FlugNr], [bo].[PassagierID], [p].[PersonID], [p].[DetailID], [p].[EMail], [p].[Geburtsdatum], [p].[KundeSei...

```

```

SELECT TOP(2) [b].[FlugNr] [b].[Abflugort] [b].[Bestreikt] [b].[CopilotId] [b].[Flugdatum] [b].[Fluggesellschaft] [b].[FlugzeugtypID] [b].[FreiePlaetze],
[b].[LetzteAenderung] [b].[Memo] [b].[NichtRaucherFlug] [b].[PilotId] [b].[Plaetze] [b].[Preis] [b].[Timestamp] [b].[Zielort] [m].[PersonID] [m].[DetailID],
[m].[Discriminator] [m].[EMail] [m].[Geburtsdatum] [m].[Gehalt] [m].[Name] [m].[VorgesetzterPersonID] [m].[Vorname] [m].[FlugscheinSeit] [m].[Flugscheintyp],
[m].[Flugschule] [m].[Flugsstunden] [m].[PersonID] [m].[DetailID] [m].[Discriminator] [m].[EMail] [m].[Geburtsdatum] [m].[Gehalt] [m].[Name],
[m].[VorgesetzterPersonID] [m].[Vorname] [m].[FlugscheinSeit] [m].[Flugscheintyp] [m].[Flugschule] [m].[Flugstunden]
FROM [Flug] AS [b]
INNER JOIN (
SELECT [m] =
FROM [Mitarbeiter] AS [m]
WHERE [m].[Discriminator] = N'Pilot'
) AS [m] ON [b].[PilotId] = [m].[PersonID]
LEFT JOIN (
SELECT [mi] =
FROM [Mitarbeiter] AS [mi]
WHERE [mi].[Discriminator] = N'Pilot'
) AS [mi] ON [b].[CopilotId] = [mi].[PersonID]
WHERE [b].[FlugNr] = 101
ORDER BY [b].[FlugNr], [m].[PersonID], [mi].[PersonID]
go
SELECT [fo].[FlugNr], [fo].[Abflugort], [fo].[Bestreikt], [fo].[CopilotId], [fo].[Flugdatum], [fo].[Fluggesellschaft], [fo].[FlugzeugtypID], [fo].[FreiePlaetze],
[fo].[LetzteAenderung], [fo].[Memo], [fo].[NichtRaucherFlug], [fo].[PilotId], [fo].[Plaetze], [fo].[Preis], [fo].[Timestamp], [fo].[Zielort]
FROM [Flug] AS [fo]
INNER JOIN (
SELECT DISTINCT TOP(2) [b].[FlugNr], [m].[PersonID], [mi].[PersonID] AS [PersonID0]
FROM [Flug] AS [b]
INNER JOIN (
SELECT [m] =
FROM [Mitarbeiter] AS [m]
WHERE [m].[Discriminator] = N'Pilot'
) AS [m] ON [b].[PilotId] = [m].[PersonID]
LEFT JOIN (
SELECT [mi] =
FROM [Mitarbeiter] AS [mi]
WHERE [mi].[Discriminator] = N'Pilot'
) AS [mi] ON [b].[CopilotId] = [mi].[PersonID]
WHERE [b].[FlugNr] = 101
ORDER BY [b].[FlugNr], [m].[PersonID], [mi].[PersonID]
) AS [m0] ON [fo].[CopilotId] = [m0].[PersonID]
ORDER BY [m0].[FlugNr], [m0].[PersonID], [m0].[PersonID]
go
SELECT [f].[FlugNr], [f].[Abflugort], [f].[Bestreikt], [f].[CopilotId], [f].[Flugdatum], [f].[Fluggesellschaft], [f].[FlugzeugtypID], [f].[FreiePlaetze],
[f].[LetzteAenderung], [f].[Memo], [f].[NichtRaucherFlug], [f].[PilotId], [f].[Plaetze], [f].[Preis], [f].[Timestamp], [f].[Zielort]
FROM [Flug] AS [f]
INNER JOIN (
SELECT DISTINCT TOP(2) [b].[FlugNr], [m].[PersonID]
FROM [Flug] AS [b]
INNER JOIN (
SELECT [m] =
FROM [Mitarbeiter] AS [m]
WHERE [m].[Discriminator] = N'Pilot'
) AS [m] ON [b].[PilotId] = [m].[PersonID]
WHERE [b].[FlugNr] = 101
ORDER BY [b].[FlugNr], [m].[PersonID]
) AS [m0] ON [f].[PilotId] = [m0].[PersonID]
ORDER BY [m0].[FlugNr], [m0].[PersonID]
go
SELECT [bo].[FlugNr], [bo].[PassagierID], [p].[PersonID], [p].[DetailID], [p].[EMail], [p].[Geburtsdatum], [p].[KundeSeit], [p].[Name], [p].[PassagierStatus], [p].[Vorname]
FROM [Buchung] AS [bo]
INNER JOIN [Passagier] AS [p] ON [bo].[PassagierID] = [p].[PersonID]
WHERE EXISTS (
SELECT TOP(2) 1
FROM [Flug] AS [b]
WHERE ([b].[FlugNr] = 101) AND ([bo].[FlugNr] = [b].[FlugNr]))
ORDER BY [bo].[FlugNr]
go

```

**Bild 10.4** Der SQL Server-Profiler zeigt die vier SQL-Befehle, die das Eager Loading-Beispiel in Entity Framework Core auslöst.

## 10.4 Explizites Nachladen (Explicit Loading)

In Entity Framework Core Version 1.1, das am 16.11.2016 erschienen ist, hat Microsoft das explizite Nachladen nachgerüstet. Hier gibt der Softwareentwickler mit Hilfe der Methoden `Reference()` (für Einzelobjekte), `Collection()` (für Mengen) und einem danach folgenden `Load()` an, dass in Beziehung stehende Objekte jetzt zu laden sind. Diese Methoden stehen aber nicht auf dem Entitätsobjekt selbst zur Verfügung, sondern sind Teil der Klasse `EntityEntry<T>`, die man durch die Methode `Entry()` in der Klasse `DbContext` erhält (siehe Listing). Mit `IsLoaded()` kann man prüfen, ob das Objekt schon geladen wurde. `IsLoaded()` liefert auch dann `true`, wenn es kein passendes Objekt in der Datenbank gab. Es zeigt also nicht an, ob eine Navigationsbeziehung ein Gegenobjekt hat, sondern ob in der aktuellen Kontextinstanz schon einmal versucht wurde, ein passendes Objekt dafür zu laden. Wenn also im nächsten Listing der Flug 101 zwar bereits einen zugewiesenen Piloten (Herrn

Koch), aber noch keinen Copiloten hat, führt dies zur Ausgabe in der nächsten Abbildung. Man muss sich bewusst sein, dass jede Ausführung von Load() zu einem expliziten Absenden eines SQL-Befehls zum Datenbankmanagementsystem führt.

**Listing 10.3** Durch das explizite Nachladen sendet Entity Framework Core sehr viele einzelne SQL-Befehle zur Datenbank

```

/// <summary>
/// Liefert Pilot-, Buchungs- und Passagierinformationen via explizitem Laden
/// </summary>
public static void Demo_ExplizitLoading_v11()
{
    CUI.Headline(nameof(Demo_ExplizitLoading_v11));

    using (var ctx = new WWingsContext())
    {
        // Lade nur den Flug
        var f = ctx.FlugSet
            .SingleOrDefault(x => x.FlugNr == 101);

        Console.WriteLine($"Flug Nr {f.FlugNr} von {f.Abflugort} nach {f.Zielort} hat
{f.FreiePlaetze} freie Plätze!");

        // Lade nur den Pilot und Copilot nach
        if (!ctx.Entry(f).Reference(x => x.Pilot).IsLoaded)
            ctx.Entry(f).Reference(x => x.Pilot).Load();
        if (!ctx.Entry(f).Reference(x => x.Copilot).IsLoaded)
            ctx.Entry(f).Reference(x => x.Copilot).Load();

        // Prüfung, ob geladen
        if (ctx.Entry(f).Reference(x => x.Pilot).IsLoaded) Console.
WriteLine("Pilot ist geladen!");
        if (ctx.Entry(f).Reference(x => x.Copilot).IsLoaded) Console.WriteLine("Co-Pilot
ist geladen!");

        if (f.Pilot != null) Console.WriteLine($"Pilot: {f.Pilot.Name} hat {f.Pilot.
FluegeAlsPilot.Count} Flüge als Pilot!");
        else Console.WriteLine("Kein Pilot zugewiesen!");
        if (f.Copilot != null) Console.WriteLine($"Copilot: {f.Copilot.Name} hat
{f.Copilot.FluegeAlsCopilot.Count} Flüge als Copilot!");
        else Console.WriteLine("Kein Copilot zugewiesen!");

        // Lade nur die Buchungsliste nach
        if (!ctx.Entry(f).Collection(x => x.Buchungen).IsLoaded)
            ctx.Entry(f).Collection(x => x.Buchungen).Load();

        Console.WriteLine("Anzahl Passagiere auf diesem Flug: " + f.Buchungen.Count);
        Console.WriteLine("Passagiere auf diesem Flug:");
        foreach (var b in f.Buchungen)
        {

            // Lade nur den Passagier für diese Buchung nach
            if (!ctx.Entry(b).Reference(x => x.Passagier).IsLoaded)
                ctx.Entry(b).Reference(x => x.Passagier).Load();

            Console.WriteLine("- Passagier #{0}: {1} {2}", b.Passagier.PersonID,
b.Passagier.Vorname, b.Passagier.Name);
        }
    }
}

```

```

    }
  }
}

```

```

Demo_EagerLoading
WWWingsContext: OnConfiguring
WWWingsContext: OnModelCreating
Flug Nr 101 von Seattle nach Moskau hat 129 freie Plätze!
Pilot: Koch hat 10 Flüge als Pilot!
Copilot: Stoiber hat 6 Flüge als Copilot!
Anzahl Passagiere auf diesem Flug: 8
Passagiere auf diesem Flug:
- Passagier #12: Niklas Bauer
- Passagier #15: Jan Schäfer
- Passagier #17: Leon Klein
- Passagier #47: Lukas Schneider
- Passagier #59: Laura Wagner
- Passagier #67: Marie Weber
- Passagier #87: Leonie Schäfer
- Passagier #98: Anna Schmidt

```

Bild 10.5 Ausgabe zum obigem Listing

## ■ 10.5 Preloading mit Relationship Fixup

Entity Framework Core unterstützt wie das bisherige Entity Framework eine weitere Lade-strategie: das Preloading in Verbindung mit dem Relationship Fixup im RAM. Dabei schickt der Softwareentwickler mehrere LINQ-Befehle für die verbundenen Objekte explizit ab, und der OR-Mapper setzt die jeweils neu hinzukommenden Objekte bei ihrer Materialisierung mit denjenigen Objekten zusammen, die sich bereits im RAM befinden. Nach der Befehls-folge

```

var flug = ctx.FlugSet
    .SingleOrDefault(x => x.FlugNr == 101);
ctx.PilotSet.Where(p => p.FluegeAlsPilot.Any(x => x.FlugNr == 101) ||
    p.FluegeAlsCopilot.Any(x => x.FlugNr == 101)).ToList();

```

findet man im RAM beim Zugriff auf `flug.Pilot` und `flug.Copilot` tatsächlich das entsprechende Pilot- und Copilotobjekt von Flug 101. Entity Framework Core erkennt beim Laden der beiden Piloten, dass es im RAM bereits ein Flug-Objekt gibt, das diese beiden Piloten als Pilot bzw. Copilot braucht. Es setzt dann im RAM das Flug-Objekt mit den beiden Piloten-Objekten zusammen. Diese Funktion nennt man Relationship Fixup.

Während in den beiden obigen Zeilen gezielt Pilot und Copilot für den Flug 101 geladen wurden, kann der Softwareentwickler das Relationship Fixup auch für die Optimierung durch Caching verwenden. Das nächste Listing zeigt, dass alle Piloten geladen werden und danach einige Flüge. Zu jedem geladenen Flug stehen danach Pilot- und Copilot-Objekt zur

Verfügung. Wie immer beim Caching braucht man hier natürlich etwas mehr RAM, da auch Pilot-Objekte geladen werden, die niemals gebraucht werden. Außerdem muss man sich bewusst sein, dass man ein Aktualitätsproblem haben kann, weil die abhängigen Daten auf dem gleichen Stand sind wie die Hauptdaten. Aber das verhält sich beim Caching bekanntlich stets auf diese Weise. Allerdings spart man Rundgänge zum Datenbankmanagementsystem, und damit verbessert man die Geschwindigkeit.

Bemerkenswert ist, dass weder beim obigen Laden der beiden Piloten noch beim Laden aller Piloten in dem Beispiel das Ergebnis der LINQ-Abfrage einer Variablen zugewiesen wird. Dies ist tatsächlich nicht notwendig, denn Entity Framework Core enthält (genau wie das bisherige Entity Framework) in seinem First-Level-Cache einen Verweis im RAM auf alle Objekte, die jemals in eine bestimmte Instanz der Kontextklasse geladen wurden. Das Relationship Fixup funktioniert daher auch ohne Speicherung in einer Variablen. Die Zuweisung zu einer Variablen (`List<Pilot> allePiloten = ctx.PilotSet.ToList();`) ist freilich nicht schädlich, sondern kann sinnvoll sein, wenn man im Programmablauf eine Liste aller Piloten braucht. Zu beachten ist auch, dass das Relationship Fixup nicht kontextinstanzübergreifend funktioniert. Ein dafür notwendiger Second-Level-Cache ist in Entity Framework Core bisher nicht vorhanden.

Das obige Codefragment zeigt beim Laden der beiden Piloteninformationen auch sehr schön, dass man den Join-Operator in Entity Framework Core vermeiden kann, wenn man die Navigationseigenschaften und die `Any()`-Methode verwendet. `Any()` prüft, ob es mindestens einen Datensatz gibt, der eine Bedingung erfüllt oder nicht erfüllt. Im obigen Fall reicht es, dass der Pilot einmal für den gesuchten Flug als Pilot oder Copilot zugeteilt wurde. In anderen Fällen kann man die LINQ-Methode `All()` einsetzen: Wenn man eine Menge von Datensätzen ansprechen will, die alle eine Bedingung erfüllen oder nicht erfüllen.

**Listing 10.4** Caching der Piloten. Egal, welchen Flug man danach lädt, Pilot- und Copilot-Objekt sind vorhanden.

```

/// <summary>
/// Liefert Pilot-, Buchungs- und Passagierinformationen via Preloading /
RelationshipFixup
/// </summary>
public static void Demo_PreLoading()
{
    CUI.Headline(nameof(Demo_PreLoading));

    using (var ctx = new WWingsContext())
    {

        int flugNr = 101;

        // 1. Lade nur den Flug selbst
        var f = ctx.FlugSet
            .SingleOrDefault(x => x.FlugNr == flugNr);

        // 2. Lade beide Piloten für obigen Flug
        ctx.PilotSet.Where(p => p.FluegeAlsPilot.Any(x => x.FlugNr == flugNr) ||
            p.FluegeAlsCopilot.Any(x => x.FlugNr == flugNr)).ToList();

        // 3. Lade andere Flüge dieser beiden Piloten
        ctx.FlugSet.Where(x => x.PilotId == f.PilotId || x.CopilotId == f.CopilotId).ToList();
    }
}

```



```

// 4. Lade Buchungen für obigen Flug
ctx.BuchungSet.Where(x => x.FlugNr == flugNr).ToList();

// 5. Lade Passagiere für obigen Flug
ctx.PassagierSet.Where(p => p.Buchungen.Any(x => x.FlugNr == flugNr)).ToList();

// nicht notwendig: ctx.ChangeTracker.DetectChanges();

Console.WriteLine($"Flug Nr {f.FlugNr} von {f.Abflugort} nach {f.Zielort} hat
{f.FreiePlaetze} freie Plätze!");
if (f.Pilot != null) Console.WriteLine($"Pilot: {f.Pilot.Name} hat {f.Pilot.
FluegeAlsPilot.Count} Flüge als Pilot!");
else Console.WriteLine("Kein Pilot zugewiesen!");
if (f.Copilot != null) Console.WriteLine($"Copilot: {f.Copilot.Name} hat
{f.Copilot.FluegeAlsCopilot.Count} Flüge als Copilot!");
else Console.WriteLine("Kein Copilot zugewiesen!");

Console.WriteLine("Anzahl Passagiere auf diesem Flug: " + f.Buchungen.Count);
Console.WriteLine("Passagiere auf diesem Flug:");
foreach (var b in f.Buchungen)
{
    Console.WriteLine("- Passagier #{0}: {1} {2}", b.Passagier.PersonID,
b.Passagier.Vorname, b.Passagier.Name);
}
}
}
}

```

Das nächste Listing zeigt die Neufassung der Aufgabe, dieses Mal mit Preloading und Relationship Fixup statt Eager Loading. Hier werden Flug, Piloten, deren andere Flüge, Buchungen und Passagiere einzeln geladen. Der Programmcode sendet also fünf SELECT-Befehle zum Datenbankmanagementsystem (im Gegensatz zu den vier SELECT-Befehlen, die die Lösung mit Eager Loading sendet), vermeidet dabei aber einige Joins.

Der Relationship Fixup-Trick wirkt sich positiv aus, wenn eine oder mehrere der folgenden Bedingungen erfüllt sind:

- Die Ergebnismenge der Hauptdaten ist groß und die Menge der abhängigen Daten ist klein.
- Es gibt mehrere verschiedene abhängige Datenmengen, die vorab geladen werden können.
- Die vorab geladenen Objekte sind selten veränderliche (Stamm-)Daten.
- Man führt in einer einzigen Kontextinstanz mehrere Abfragen aus, die die gleichen abhängigen Daten besitzen.

**Listing 10.5** Laden von Flug, Piloten, Buchungen und Passagieren in getrennten LINQ-Befehlen. Entity Framework Core setzt die getrennt geladenen Objekte im RAM zusammen.

```

/// <summary>
/// Liefert Pilot-, Buchungs- und Passagierinformationen via Preloading /
RelationshipFixup
/// </summary>
public static void Demo_PreLoading()
{
    CUI.Headline("Demo_PreLoading");
}

```

```
using (var ctx = new WWingsContext())
{
    int flugNr = 101;

    // 1. Lade nur den Flug
    var f = ctx.FlugSet
        .SingleOrDefault(x => x.FlugNr == flugNr);

    // 2. Lade beide Piloten
    ctx.PilotSet.Where(p => p.FluegeAlsPilot.Any(x => x.FlugNr == flugNr) ||
p.FluegeAlsCopilot.Any(x => x.FlugNr == flugNr)).ToList();

    // 3. Lade andere Flüge der Piloten
    ctx.FlugSet.Where(x => x.PilotId == f.PilotId || x.CopilotId == f.CopilotId).ToList();

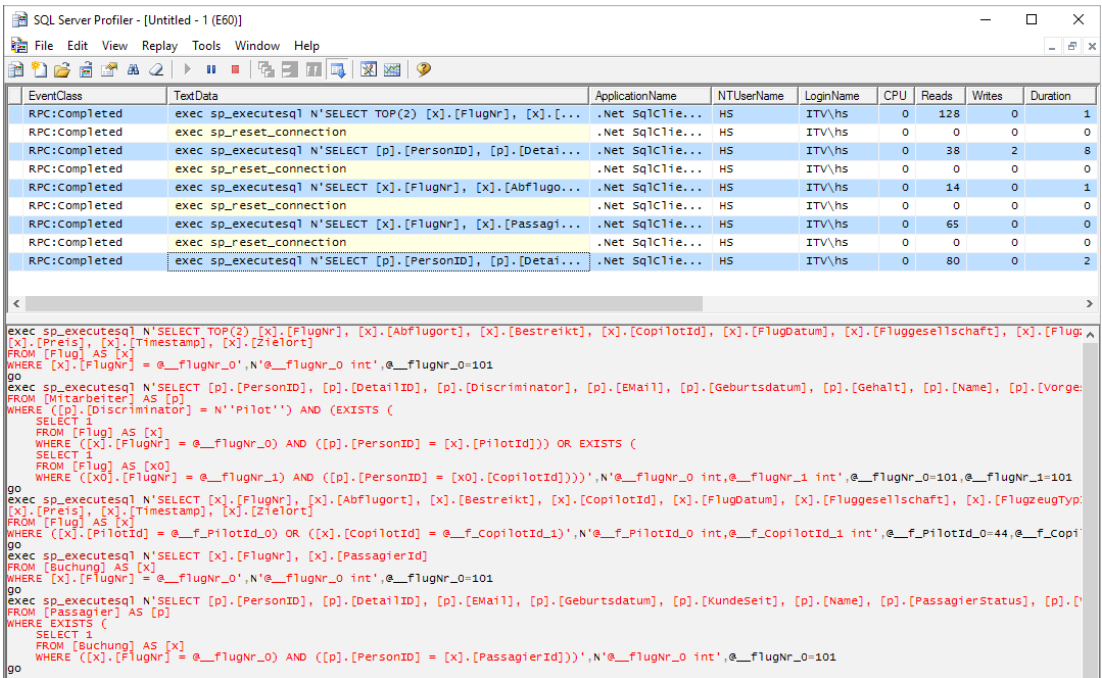
    // 4. Lade Buchungen
    ctx.BuchungSet.Where(x => x.FlugNr == flugNr).ToList();

    // 5. Lade Passagiere
    ctx.PassagierSet.Where(p => p.Buchungen.Any(x => x.FlugNr == flugNr)).ToList();

    // nicht notwendig: ctx.ChangeTracker.DetectChanges();

    Console.WriteLine($"Flug Nr {f.FlugNr} von {f.Abflugort} nach {f.Zielort} hat
{f.FreiePlaetze} freie Plätze!");
    if (f.Pilot != null) Console.WriteLine($"Pilot: {f.Pilot.Name} hat {f.Pilot.
FluegeAlsPilot.Count} Flüge als Pilot!");
    else Console.WriteLine("Kein Pilot zugewiesen!");
    if (f.Copilot != null) Console.WriteLine($"Copilot: {f.Copilot.Name} hat
{f.Copilot.FluegeAlsCopilot.Count} Flüge als Copilot!");
    else Console.WriteLine("Kein Copilot zugewiesen!");

    Console.WriteLine("Anzahl Passagiere auf diesem Flug: " + f.Buchungen.Count);
    Console.WriteLine("Passagiere auf diesem Flug:");
    foreach (var b in f.Buchungen)
    {
        Console.WriteLine("- Passagier #{0}: {1} {2}", b.Passagier.PersonID,
b.Passagier.Vorname, b.Passagier.Name);
    }
}
```



The screenshot shows the SQL Server Profiler interface. The top part displays a list of events with columns: EventClass, TextData, ApplicationName, NTUserName, LoginName, CPU, Reads, Writes, and Duration. The bottom part shows a SQL query listing for loading 50 flights with pilots and passengers.

EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration
RPC:Completed	exec sp_executesql N'SELECT TOP(2) [X].[FlugNr], [X].[...]	.Net SqlClie...	HS	ITV\hs	0	128	0	1
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	HS	ITV\hs	0	0	0	0
RPC:Completed	exec sp_executesql N'SELECT [p].[PersonID], [p].[Detail...]	.Net SqlClie...	HS	ITV\hs	0	38	2	8
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	HS	ITV\hs	0	0	0	0
RPC:Completed	exec sp_executesql N'SELECT [X].[FlugNr], [X].[Abflugo...]	.Net SqlClie...	HS	ITV\hs	0	14	0	1
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	HS	ITV\hs	0	0	0	0
RPC:Completed	exec sp_executesql N'SELECT [X].[FlugNr], [X].[Passagi...]	.Net SqlClie...	HS	ITV\hs	0	65	0	0
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	HS	ITV\hs	0	0	0	0
RPC:Completed	exec sp_executesql N'SELECT [p].[PersonID], [p].[Detail...]	.Net SqlClie...	HS	ITV\hs	0	80	0	2

```

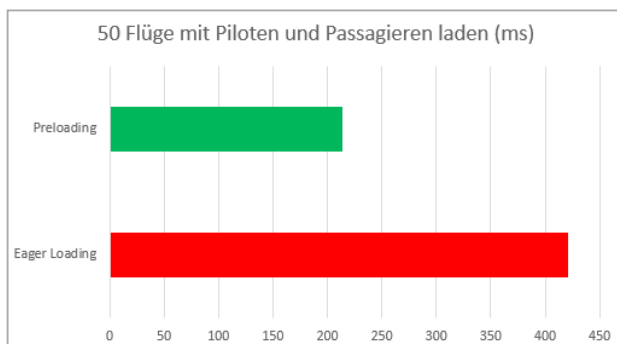
exec sp_executesql N'SELECT TOP(2) [X].[FlugNr], [X].[Abflugort], [X].[Bestreikt], [X].[CopilotID], [X].[Flugdatum], [X].[Fluggesellschaft], [X].[Flugzeugtyp], [X].[Preis], [X].[Timestamp], [X].[Zielort]
FROM [Flug] AS [X]
WHERE [X].[FlugNr] = @__flugNr_0',N'@__flugNr_0 int',@__flugNr_0=101
go
exec sp_executesql N'SELECT [p].[PersonID], [p].[DetailID], [p].[Email], [p].[Geburtsdatum], [p].[Gehalt], [p].[Name], [p].[VorgeName]
FROM [Mitarbeiter] AS [p]
WHERE ([p].[Discriminator] = N'Pilot') AND (EXISTS (
SELECT 1
FROM [Flug] AS [X]
WHERE ([X].[FlugNr] = @__flugNr_0 AND ([p].[PersonID] = [X].[PilotID])) OR EXISTS (
SELECT 1
FROM [Flug] AS [X0]
WHERE ([X0].[FlugNr] = @__flugNr_1 AND ([p].[PersonID] = [X0].[CopilotID]))))',N'@__flugNr_0 int,@__flugNr_1 int',@__flugNr_0=101,@__flugNr_1=101
go
exec sp_executesql N'SELECT [X].[FlugNr], [X].[Abflugort], [X].[Bestreikt], [X].[CopilotID], [X].[Flugdatum], [X].[Fluggesellschaft], [X].[Flugzeugtyp], [X].[Preis], [X].[Timestamp], [X].[Zielort]
FROM [Flug] AS [X]
WHERE ([X].[PilotID] = @__f_PilotID_0) OR ([X].[CopilotID] = @__f_CopilotID_1)',N'@__f_PilotID_0 int,@__f_CopilotID_1 int',@__f_PilotID_0=44,@__f_CopilotID_1=101
go
exec sp_executesql N'SELECT [X].[FlugNr], [X].[PassagierID]
FROM [Buchung] AS [X]
WHERE [X].[FlugNr] = @__flugNr_0',N'@__flugNr_0 int',@__flugNr_0=101
go
exec sp_executesql N'SELECT [p].[PersonID], [p].[DetailID], [p].[Email], [p].[Geburtsdatum], [p].[KundeSet], [p].[Name], [p].[PassagierStatus], [p].[VorgeName]
FROM [Passagier] AS [p]
WHERE EXISTS (
SELECT 1
FROM [Buchung] AS [X]
WHERE ([X].[FlugNr] = @__flugNr_0 AND ([p].[PersonID] = [X].[PassagierID]))',N'@__flugNr_0 int',@__flugNr_0=101
go

```

**Bild 10.6** Der SQL Server-Profiler zeigt die fünf SQL-Befehle, die das obige Listing

Im Geschwindigkeitsvergleich zeigt sich auch bei dem hier besprochenen Szenario des Ladens eines Flugs mit Piloten und Passagieren bereits ein Geschwindigkeitsvorteil für das Preloading. Die Messung, die in nachstehenden Abbildung dargestellt ist, wurde zur Vermeidung von Messabweichungen mit 51 Durchläufen ermittelt, wobei der erste Durchlauf (Kaltstart für den Entity Framework Core-Kontext und ggf. auch die Datenbank) jeweils nicht berücksichtigt wurde. Zudem wurden alle Bildschirmausgaben ausgebaut.

Selbstverständlich kann man Eager Loading und Preloading beliebig mischen. In der Praxis muss man jedoch für jeden einzelnen Fall das optimale Verhältnis finden.



**Bild 10.7** Geschwindigkeitsvergleich von Eager Loading und Preloading

## ■ 10.6 Details zum Relationship Fixup

Wie beim klassischen Entity Framework erfolgt auch bei Entity Framework Core im Rahmen des Relationship Fixup eine Zuweisung der Instanz eines Mengentyps zu der Navigationseigenschaft, wenn die Navigationseigenschaft den Wert null hat.

Dieser Automatismus ist gegeben, unabhängig davon, ob der Softwareentwickler bei der Deklaration der Navigationseigenschaft als Typ eine Schnittstelle oder eine Klasse verwendet hat. Entity Framework Core instanziiert die Mengenkategorie selbst. Im Fall der Deklaration der Navigationseigenschaft mit einem Schnittstellentyp wählt Entity Framework Core selbst eine geeignete Mengenkategorie. Bei `ICollection<T>` wird `HashSet<T>` gewählt, bei `IList<T>` eine `List<T>`.

# Index

## Symbole

1 zu 1 82  
1 zu N 22, 60, 82  
.csproj 54, 56  
.edml 328  
.edps 335  
.efml 328, 339  
.hbml 328  
.NET 357  
  – Versionen 17  
.NET Core 13, 37, 54, 105, 383  
.NET Framework 13  
.NET Standard 37  
.tmpl 334, 341  
.view 328

## A

Abfragefilter 223, 225, 263  
AbsoluteExpiration 289, 293  
Add() 156 ff., 171, 219  
AddDbContext() 409 ff.  
AddDbContextPool() 411  
Added 158, 176  
AddEntityFrameworkStores() 409  
Add-Migration 94, 102  
AddProfile() 363  
AddRange() 169  
AddScoped() 411  
AddSingleton() 411  
AddTransient() 411  
ADO.NET 106  
AfterMap() 374  
Aggregatoperator 106  
AllAsync() 205

Alternativer Schlüssel 252  
Änderungskonflikt 177, 181, 284  
Android 23, 379  
Angular 383  
Any() 146  
AnyAsync() 205  
App 412, 424  
Appender 346  
ApplicationDbContext 408  
Application Insights 383, 400, 404  
ApplicationUser 408  
ApplyConfiguration() 88  
appsettings.json 408  
Arraytyp 76  
Artefakt 41  
AsDgml() 317  
AsNoTracking() 151, 281, 285 f.  
ASP.NET 379  
ASP.NET Core 24, 54, 379, 383, 400  
AsTracking() 286, 288  
async 205  
Attach() 171, 282, 285  
Aufzählungstyp 68  
Auslagerungsdatei 268  
AutoMapper 114, 306, 352, 357, 359, 363,  
  370, 376  
Autowert 236  
Average() 128  
AverageAsync() 205  
await 205  
Azure Table Storage 23, 325  
Azure Web App 383

**B**

Batching 24, 169, 273  
 BeforeMap() 374  
 BeginningEdit() 285  
 BeginTransaction() 195  
 Berechnete Spalte 236  
 Berechnungsformel 236  
 Best Practice 268  
 Betriebssysteme 17  
 Beziehung 78, 81  
 – optional 79  
 Beziehungszuweisung 270  
 Bigcommerce 19  
 Bootstrap 383  
 Bulk Insert 267

**C**

C# 1, 318  
 Cache 106  
 Cache Invalidation 291  
 CacheManager 292f.  
 cacheMemoryLimitMegabytes 290  
 Caching 289, 352  
 Cascade 256f.  
 Cascading Delete *siehe* Kaskadierendes Löschen  
 Change Tracker 151, 176  
 Change Tracking 154  
 ClientSetNull 256f.  
 Cloud 379  
 Code-based Modelling 21  
 Code First 21, 40  
 Codegenerierungsvorlage 328, 341  
 Column 75, 77  
 CommandBuilder 181  
 Command Interceptor 23  
 Commit() 195  
 Complex Type 22  
 Computed Column 236  
 Concurrency 181  
 ConcurrencyCheck 184, 274, 339  
 ConcurrencyNoCheck 186  
 Condition() 364  
 Console 6  
 Controller 389  
 Convert() 367  
 ConvertUsing() 367

Cordova 383  
 CORS *siehe* Cross-Origin Resource Sharing  
 Count() 106, 110, 128  
 CountAsync() 205  
 Create() 156  
 CreateLogger() 201  
 CreateMap() 359, 363, 375f.  
 CreateTable() 121  
 Cross Join 133  
 Cross-Origin Resource Sharing 400  
 Cross-Platform 379  
 Cross-Platform-App 424  
 CSS 383  
 CUI 6  
 CurrentValue 176

**D**

Dapper 15  
 Database 91, 228, 416, 430  
 Database Explorer 336  
 Database First 21, 40, 328  
 DatabaseGenerated 78  
 DataContract 335  
 Data Definition Language 99  
 Datagrid 285  
 Data Manipulation Language 228  
 DataMember 335  
 DataReader 269  
 DataSet 181  
 Datenbank 19  
 Datenbankmanagementsystem 42  
 Datenbankschema 73  
 Datenbanksicht 123, 258  
 Datenbanktransaktion 173  
 Datentransferobjekt 305  
 Datentyp 76  
 Datenzugriffsschicht 302, 306  
 DB2 19  
 DbConcurrencyException 152  
 DbContext 21, 47, 65, 67, 105, 187, 347  
 DbDataReader 227  
 DBMS *siehe* Datenbankmanagementsystem  
 DbSet 65, 68, 74, 156, 286  
 DbSet<T> 120, 388  
 DbUpdateConcurrencyException 190  
 DbUpdateException 153  
 DDL *siehe* Data Definition Language  
 Default Value 242  
 Defered Execution 213

Delete() 277  
DELETE 151, 228, 267, 272, 351  
Deleted 176  
Dependency Injection 408f., 428  
Detached 157, 176  
DetectChanges() 162, 436  
DevArt 327  
DevExpress XPO 325  
DevForce 325  
Dezimalzahl 339  
DGML *siehe* Directed Graph Markup Language  
Diagramm 316  
Directed Graph Markup Language 316  
DisplayName 341  
DisplayName() 88  
Dispose() 105, 109  
DML *siehe* Data Manipulation Language  
DTO *siehe* Datentransferobjekt  
Dynamic LINQ 216  
DynamicQueryable 216  
Dynamics CRM 19

## E

Eager Loading 23, 138, 140, 267  
EDM *siehe* Entity Data Model  
EDMX 21f., 40  
ef.exe 93  
EFMigrationsHistory 99, 340  
EFPlus  
– Entity Framework Plus 277  
EFSecondLevelCache.Core 296  
Einhorn 441  
Electron 383  
ElementAt() 213  
EnsureCreated() 68f., 91, 103, 416, 430  
Entitätsklasse 59, 68, 226, 230, 415, 428  
Entity() 69  
Entity Developer 327f., 340  
EntityEntry 230  
Entity Framework Plus 265, 277f., 296, 351  
Entity Framework Profiler 344f.  
EntityFrameworkQueryableExtensions 205  
EntityObject 305  
Entity SQL 21, 318  
EntityState 176, 282  
EntityTypeConfiguration 87  
Entries() 178  
Entry() 143, 152, 176, 230  
EntryObject 176f.

Entwicklungsteam 441  
Enumerator 213  
Equals() 335  
Erweiterungsmethode 205  
ESQL *siehe* Entity SQL  
E-Tag 23  
ETW *siehe* Event Traces for Windows  
Event Traces for Windows 325  
ExactTarget 19  
ExecuteSqlCommand() 228  
ExecuteSqlQuery() 227  
Explicit Loading 138, 143  
Expression Tree 106, 213

## F

F# 318  
Feldlänge 77  
FETCH 112  
FileDb 325  
Find() 115  
Find-Package 33  
First() 106, 114f., 213  
FirstAsync() 205  
First-Level-Cache 296  
FirstOrDefault() 106, 114f., 130  
FirstOrDefaultAsync() 205  
Fluent-API 56, 69, 74, 79, 88  
Foreach 155  
ForeachAsync() 208  
ForMember() 364, 370  
ForSqlServerIsClustered() 83  
Forward Engineering 2, 39, 57, 327f., 339  
Fremdschlüssel 78  
Fremdschlüsseleigenschaft 68, 270  
Fremdschlüsselspalte 79  
FreshBooks 19  
FromCache() 297  
FromSql() 121, 195, 219, 222, 224, 227, 336  
Future() 265  
Future Query 265

## G

Garbage Collection 268  
Genome 15  
Geography 22  
Geometry 22  
Geschäftslogik 304, 306

Geschäftsobjekt 305  
 GetDatabaseValues() 177, 191  
 GetEntityTypes() 185  
 Get-Package 33  
 GetProperties() 185  
 Github 441  
 group by 22, 115  
 GroupBy 116, 119, 129  
 GroupBy() 216  
 Gruppierung 115, 119, 123  
 GUID 78, 253

## H

HasAlternateKey() 252  
 HasColumnName() 75  
 HasColumnType() 77  
 HasComputedColumnSql() 238  
 HasDefaultValue() 242  
 HasDefaultValueSql() 242, 246 f.  
 HasForeignKey() 70, 82  
 HashSet 78, 150  
 HasIndex() 83  
 HasKey() 70, 78, 120, 259  
 HasMany() 70, 79, 82  
 HasMaxLength() 78  
 HasName() 84, 253  
 HasOne() 79  
 HasQueryFilter() 223, 225, 263  
 HasSequence() 247  
 Header 398  
 Hibernating Rhinos 345 f.  
 HostFileChangeMonitor 291  
 HTML 383  
 HTTP 398  
 HTTPS 398

## I

ICloneable 335  
 ICollection 150, 368  
 Identity 236  
 IdentityDbContext 408  
 IdentityUser 408  
 IEntityTypeConfiguration 87  
 IEnumerable 106 f., 368  
 Ignore() 74, 123, 371  
 IgnoreQueryFilters() 265  
 IList 150, 368

ILogger 201  
 ILoggerFactory 201  
 ILoggerProvider 201  
 IModel 188  
 IMutableEntityType 88, 388  
 IMutableEntityType 88  
 IMutalModel 188  
 in 126  
 Include() 131  
 Index 83  
   - Unique 252  
 Initialize() 359, 363  
 In-Memory 19, 27, 104  
 INotifyPropertyChanged 335, 341  
 INotifyPropertyChanging 335, 341  
 INSERT 151, 156, 228, 267  
 Install-Package 30, 44, 64, 92  
 International .NET Association XVIII  
 InvalidOperationException 389  
 InverseProperty 82  
 IObservable 111  
 iOS 23, 379  
 IQueryable 106 f., 109, 111, 199, 219  
 IsConcurrencyToken 185, 188  
 IsConcurrencyToken() 189, 274  
 IsKey() 254  
 IsLoaded() 143  
 IsModified 282  
 IsPrimaryKey() 254  
 IsRequired() 77, 80  
 IsRowVersion() 189  
 IsUnique() 83

## J

Java 15  
 Join 132 f.

## K

Kaltstart 268 f.  
 Kardinalität 82  
 Kaskadierendes Löschen 53, 70, 73, 255  
 Key 78, 120, 259  
 Klasse  
   - generisch 371  
   - partiell 334  
 Konflikterkennung 182, 184, 274  
 Konsolenanwendung 5, 268



Konsolenausgabe 5f.  
Kontextklasse 64, 105, 387, 416, 428  
Konvertierungsoperator 106, 211, 222

## L

Lambda 124  
Language Integrated Query 21, 105, 114, 123, 211, 219, 232, 302, 317  
Lazy Loading 138f.  
Lebensdauer 389  
Leistungsoptimierung 267, 279  
Leistungstest 268  
LINQ 22  
LINQPad 317  
LINQ *siehe* Language Integrated Query  
LINQ-to-Entities 105  
LINQ-to-Objects 179  
LINQ-to-SQL 16  
Linux 23, 379  
List 78, 82, 107, 150, 368  
LLBLGen Pro 15, 325  
Load() 144  
Log() 199, 201, 298  
Logger 201  
Logger Provider 201  
Logging 199

## M

MacOS 23, 379  
Magento 19  
MailChimp 19  
Mandantenfähigkeit 263  
Map() 360, 368  
MapFrom() 364, 370  
Massenoperation 267  
Massive 15  
Master-Detail 59, 78, 139  
Max() 106, 128  
MaxAsync() 205  
MaxLength 78  
MemoryCache 289f., 296f., 352  
MemoryCacheEntryOptions 297  
MEST *siehe* Multiple Entity Sets per Type  
Metadata 240  
Microsoft Certified Solution Developer XVIII  
Microsoft.EntityFrameworkCore 205

Microsoft.EntityFrameworkCore.Metadata.  
Builders 87  
Microsoft.EntityFrameworkCore.Tools.DotNet  
54  
Microsoft.Extensions.Caching.Abstractions  
297  
Microsoft.Extensions.Caching.Memory 289  
Microsoft.Extensions.DependencyInjection  
410  
Microsoft SQL Server 19, 27, 103, 112  
Microsoft SQL Server Compact 27, 103  
Microsoft SQL Server LocalDB 408  
Migrate() 103  
Migration 25  
Min() 106, 128  
MinAsync() 205  
MiracleList 1, 379  
ModelBuilder 69, 88, 188  
Model First 21, 40, 328, 339  
Model View ViewModel 305, 419, 430  
Modified 153, 176, 282  
MongoDB 19  
Mono 13, 105  
Most Valuable Professional XVIII  
Multi-Threading 389  
MVVM *siehe* Model View ViewModel  
MySQL 19, 28, 325, 328

## N

Nachladen 143, 267  
Navigationseigenschaft 68, 74, 150  
NDO 15  
New 156  
nHibernate 15  
NoSQL 23, 325  
NotMapped 74  
No-Tracking 268, 279  
Nuget 27, 30, 47, 94  
null 77  
Nullable 77  
NullLogger 201  
NullSubstitute() 364  
N zu M 22, 60, 435

**O**

ObjectCache 289  
 ObjectContext 21, 47  
 Objektmenge 368  
 Objektmodell 124  
 Objekt-Objekt-Mapping 306, 352  
 Objektorientierte Datenbank 13  
 Objekt-Relationaler Mapper 13  
 Objektverweis 270  
 Objekt-zu-Objekt-Mapping 306  
 OBJ.NET 15  
 OData *siehe* Open Data Protocol  
 OFFSET 112  
 OnConfiguring() 66, 174, 387, 416  
 OnDelete() 256  
 OnModelCreating() 69 f., 85, 88, 122, 187, 230,  
 233, 263, 388, 416  
 OODB *siehe* Objektorientierte Datenbank  
 OOM *siehe* Objekt-Objekt-Mapping  
 Open Access 15  
 Open Data Protocol 325  
 Open Source 15  
 Oracle 19, 76, 112, 325, 328  
 ORA\_ROWSCN 188  
 orderby 126  
 OrderBy() 216  
 OriginalValue 176  
 OriginalValues 191  
 ORM 13  
 OwnsOne() 233

**P**

Paging 111, 127  
 persistent 74  
 PetaPoco 15  
 Pflichtbeziehung 79  
 physicalMemoryLimitPercentage 290  
 Plain Old CLR Object 21, 47, 59, 176  
 POCO *siehe* Plain Old CLR Object  
 pollingInterval 290  
 PostgreSQL 19, 28, 104, 328  
 PowerShell 30  
 Power Tools 311 f., 316  
 Preloading 138, 145, 267  
 Primärschlüssel 42, 56, 68, 78, 115, 120, 247,  
 251, 259  
 project.json 56

Projektion 113  
 Property() 230  
 PropertyEntry 176, 240  
 Protokollierung *siehe* Logging  
 Proxyklasse 306

**Q**

QueryTrackingBehavior 286, 288  
 QuickBooks 19

**R**

RAM 115  
 Range 341  
 RavenDB 325  
 Redis 23, 296, 352  
 Reflection 375  
 RegularExpression 341  
 Relational() 88  
 Relationship Fixup 145 f., 150, 161 f.  
 Reload() 191  
 Remove() 170 ff., 219, 257, 272  
 Remove-Migration 102  
 Repository-Pattern 109  
 Required 77  
 ResolveUsing() 364  
 REST 398  
 Restrict 256 f.  
 Resultset 228  
 Reverse Engineering 2, 22, 39, 43, 56, 312,  
 327 f.  
 rownumber() 112  
 rowversion 188

**S**

Salesforce 19  
 SaveChanges() 68, 151, 153, 156, 161 f., 173,  
 191 f., 219, 232, 240, 272, 284  
 SaveChangesAsync() 205, 207  
 Scaffold-DbContext 47, 56, 312  
 Schemamigration 91  
 Schlüssel  
 –Alternativ 251  
 Schlüsselgenerierung 24  
 Script-Migration 99, 102  
 Second-Level-Cache 296, 351

Second-Level-Caching 352  
Select() 216  
SELECT 125  
Sequence *siehe* Sequenz  
Sequenz 24, 247  
Sequenzobjekt *siehe* Sequenz  
Sequenzzyklus 247, 251  
Serializable 335  
SetNull 256 f.  
SetValues() 191  
Shadow Property 24, 229, 271, 339  
Shadow State 229  
Shared Contract 306  
Sicht *siehe* Datenbanksicht  
Silverlight 357  
Single() 106, 114 f., 213  
SingleAsync() 205  
SingleOrDefault() 106, 114 f., 130, 151  
SingleOrDefaultAsync() 205  
Skip() 111 f., 127  
SlidingExpiration 293  
Soft Delete 263  
Softwarearchitektur 301  
SortedDictionary 213  
Spaltentyp 76  
Sperrern 181  
– optimistisch 181, 184  
Sprachkonvertierung 1  
SP *siehe* Stored Procedure  
SQL 21, 106, 219, 267  
– Injektionsangriff 219  
SQL Azure 383  
SqlChangeMonitor 291  
SQLite 19, 27, 76, 103, 325, 328, 412  
– DB Browser 417  
SqlQuery() 226  
Standardwert 24, 236, 242  
State 176  
Stored Procedure 21 f., 56, 172, 219, 224, 267, 302, 336  
StreamInsight 325  
Subsonic 15  
SugarCRM 19  
Sum() 106, 128  
SumAsyn() 205  
Swagger 383, 400  
System 289  
System.Linq.Dynamic 216  
System.Linq.Expressions 213  
System.Runtime.Caching 290, 293, 296

**T**

T4 *siehe* Text Template Transformation Toolkit  
Tabelle  
– temporal 56  
Tabellenaufteilung 233  
Table 75, 88, 259, 388  
Table per Concrete Type 21, 71  
Table per Hierarchy 21, 71  
Table per Type 21 ff.  
Table Splitting 233  
Table Valued Function 21, 219, 224, 267  
Take() 111, 127  
Telerik Data Access 15, 328  
Text Template Transformation Toolkit 334  
thread-safe 389  
Thread-Sicherheit 67  
Timestamp 188 f., 236, 274  
ToArray() 106, 125, 213  
ToArrayAsync() 205  
ToDictionary() 106, 125, 213  
ToList() 106 f., 125, 213, 222  
ToListAsync() 205  
ToLookup() 106, 125  
ToTable() 75  
TPC *siehe* Table per Concrete Type  
TPH *siehe* Table per Hierarchy  
TPT *siehe* Table per Type  
TrackAll 286  
Tracking 279  
TransactionScope 173  
Transaktion 173, 194  
transient 74, 157  
Try-Catch 153  
TVF *siehe* Table Valued Function  
Twitter 441  
TypeScript 383  
Typkonvertierung 366

**U**

Unchanged 153, 176  
Unicorn 441  
Union() 117, 436  
Unique Constraint 252  
Unique Index 252  
Unit Test 385  
Universal Windows Platform 24, 357, 412, 424  
Unter-Abfrage 134  
Update() 277 f.

UPDATE 151, 228, 267, 272, 278 f., 351  
 Update-Database 98, 102  
 Update Model from Database 56  
 Update-Package 35  
 UPDLOCK 194  
 UseDestinationValue() 364  
 UseSqlite() 387, 416  
 UseSqlServer() 66 f., 174  
 UseTransaction() 174  
 UseValue() 364  
 using() 109  
 UWP *siehe* Universal Windows Platform

## V

ValueGenerated 240  
 ValueGeneratedNever() 78  
 ValueGeneratedOnAddOrUpdate() 189  
 Value Injector 306  
 Value Type 77  
 var 110  
 Verbindungszeichenfolge 66  
 Vererbung 369  
 Versionsgeschichte 16  
 Verteiltes System 306  
 View 21, 56, 267, 336  
 ViewModel 305  
 View *siehe* Datenbanksicht  
 Visual Basic .NET 1, 318  
 Visual Studio  
 – Versionen 18

## W

WCF *siehe* Windows Communication Founda-  
 tion  
 WebAPI 383, 398

Weblog 441  
 where 125  
 Where() 216  
 Wilson 15  
 Windows 379  
 Windows 10 23, 412, 424  
 Windows 10 SDK 413  
 Windows Communication Foundation 335  
 WithMany() 79  
 WithOne() 79  
 World Wide Wings 1 f.  
 Wunderlist 379  
 WWWings *siehe* World Wide Wings  
 www.IT-Visions.de XVIII

## X

Xamarin 13, 24, 105, 424, 434  
 Xamarin Forms 424, 428, 434 f.  
 XAML 423, 435  
 XML 22, 74  
 XmlDocument 74  
 XPO 15  
 XUnit 385

## Z

Zeitstempel *siehe* Timestamp  
 Z.EntityFramework.Plus 277, 351  
 Zoho CRM 19  
 Zwischenspeicherung *siehe* Caching  
 Zwischentabelle 60