



Modern Data Access with Entity Framework Core

Database Programming Techniques
for .NET, .NET Core, UWP, and Xamarin
with C#

—
Holger Schwichtenberg

Apress®

Modern Data Access with Entity Framework Core

Database Programming Techniques
for .NET, .NET Core, UWP, and
Xamarin with C#

Holger Schwichtenberg

Apress®

Modern Data Access with Entity Framework Core

Holger Schwichtenberg
Essen, Germany

ISBN-13 (pbk): 978-1-4842-3551-5
<https://doi.org/10.1007/978-1-4842-3552-2>

ISBN-13 (electronic): 978-1-4842-3552-2

Library of Congress Control Number: 2018947392

Copyright © 2018 by Holger Schwichtenberg

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Joan Murray
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484235515. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

For Heidi, Felix, and Maja

Table of Contents

About the Author	xvii
Introduction	xix
Chapter 1: Introducing Entity Framework Core	1
What Is an Object-Relational Mapper?	1
OR Mappers in the .NET World.....	3
Version History of Entity Framework Core	4
Supported Operating Systems	5
Supported .NET Versions.....	5
Supported Visual Studio Versions	6
Supported Databases.....	7
Features of Entity Framework Core	8
Functions That Have Been Eliminated	9
Missing Critical Features.....	10
High-Priority Features	11
New Features in Entity Framework Core.....	13
When to Use Entity Framework Core	14
Chapter 2: Installing Entity Framework Core	15
NuGet Packages.....	15
Installing the Packages.....	19
Updating to a New Version.....	23
Chapter 3: Concepts of Entity Framework Core.....	31
Process Models for Entity Framework Core.....	31
Components of Entity Framework Core	34

- Chapter 4: Reverse Engineering of Existing Databases (Database First Development) 37**
 - Using Reverse Engineering Tools 37
 - Reverse Engineering with PowerShell Cmdlets 38
 - Generating Code..... 40
 - Looking at the Generated Program Code 44
 - Seeing an Example Client 54
 - Using the .NET Core Tool dotnet..... 56
 - Understanding the Weaknesses of Reverse Engineering..... 58

- Chapter 5: Forward Engineering for New Databases..... 61**
 - Two Types of Classes 61
 - Examples in This Chapter..... 62
 - Rules for Self-Created Entity Classes 64
 - NuGet Packages 64
 - Data Annotation Properties..... 64
 - Data Types 64
 - Relationships (Master-Detail) 65
 - Inheritance 66
 - Primary Key 66
 - Examples 67
 - Rules for the Self-Created Context Class..... 73
 - Installing the NuGet Packages..... 73
 - Base Class 74
 - Constructor 74
 - References to Entity Classes 75
 - Provider and Connection String..... 75
 - Seeing an Example 76
 - Your Own Connections..... 77
 - Thread Safety 78

Rules for Database Schema Generation	78
Looking at an Example Client.....	79
Adaptation by Fluent API (OnModelCreating()).....	81
Viewing the Generated Database Schema.....	85
Chapter 6: Customizing the Database Schema.....	87
Examples in This Chapter.....	87
Convention vs. Configuration	88
Persistent vs. Transient Classes.....	89
Names in the Database Schema.....	90
Order of Columns in a Table.....	91
Column Types/Data Types	91
Mandatory Fields and Optional Fields.....	94
Field Lengths.....	95
Primary Keys.....	95
Relationships and Foreign Keys.....	96
Optional Relationships and Mandatory Relationships.....	97
Unidirectional and Bidirectional Relationships	101
1:1 Relationships	102
Indexes.....	103
Syntax Options for the Fluent API.....	105
Sequential Configuration	105
Structuring by Lambda Statement.....	106
Structuring by Subroutines.....	107
Structuring Through Configuration Classes	108
Bulk Configuration with the Fluent API	109
Chapter 7: Database Schema Migrations	111
Creating the Database at Runtime	111
Schema Migrations at the Time of Development.....	113
Commands for the Schema Migrations.....	113
ef.exe	114

TABLE OF CONTENTS

- Add-Migration 115
- Update-Database 120
- Script-Migration 121
- Further Migration Steps 122
- Migration Scenarios 122
- More Options 125
- Problems with Schema Migration in Connection with TFS 127
- Schema Migrations at Runtime 127
- Chapter 8: Reading Data with LINQ 129**
 - Context Class 129
 - LINQ Queries 130
 - Step-by-Step Composition of LINQ Queries 134
 - Use of var 135
 - Repository Pattern 135
 - LINQ Queries with Paging 140
 - Projections 143
 - Projection to an Entity Type 143
 - Projections to an Anonymous Type 145
 - Projections to an Arbitrary Type 147
 - Querying for Single Objects 149
 - Loading Using the Primary Key with Find() 150
 - Using LINQ in RAM Instead of in the Database (Client Evaluation) 152
 - Using the Wrong Order of Commands 156
 - Using Custom Function in LINQ 158
 - Working Around the GroupBy Problem 159
 - Mapping to Nonentity Types 159
 - Creating an Entity Class for the Database View Result 160
 - Including the Entity Class in the Context Class 161
 - Using the Pseudo-Entity Class 161

Challenge: Migrations.....	163
Groupings with Database Views	165
Brief Overview of the LINQ Syntax	165
Simple SELECT Commands (All Records)	167
Conditions (where)	168
Contains (in)	169
Sorts (orderby).....	169
Paging (Skip() and Take()).....	170
Projection	171
Aggregate Functions (Count(), Min(), Max(), Average(), Sum())	172
Groupings (GroupBy).....	173
Single Objects (SingleOrDefault(), FirstOrDefault())	174
Related Objects (Include())	175
Inner Join.....	177
Cross Join (Cartesian Product)	178
Join with a Grouping	179
Subqueries (Subselects).....	180
Chapter 9: Object Relationships and Loading Strategies	183
Overview of Loading Strategies	183
Seeing the Default Behavior	184
No Lazy Loading Yet	186
Explicit Loading.....	187
Eager Loading	190
Relationship Fixup.....	195
Example for Case 1.....	196
Example for Case 2.....	197
Example for Case 3.....	199
Preloading with Relationship Fixup	201

TABLE OF CONTENTS

- Chapter 10: Inserting, Deleting, and Modifying Records 207**
 - Saving with SaveChanges() 207
 - Tracking Changes for Subobjects 211
 - Combining Commands (Batching)..... 213
 - Dealing with foreach Loop Considerations 213
 - Adding New Objects..... 215
 - Creating Related Objects 218
 - Changing Linked Objects 222
 - Dealing with Contradictory Relationships 225
 - Deleting Objects..... 233
 - Deleting Objects with Remove() 233
 - Deleting Objects with a Dummy Object..... 235
 - Bulk Deleting 237
 - Performing Database Transactions 237
 - Example 1 238
 - Example 2..... 239
 - Using the Change Tracker 242
 - Getting the State of an Object 242
 - Listing All Changed Objects 246

- Chapter 11: Preventing Conflicts (Concurrency) 251**
 - A Look at the History of Concurrency..... 251
 - No Conflict Detection by Default 252
 - Detecting Conflicts with Optimistic Locking 255
 - Detecting Conflicts for All Properties 256
 - Settling Conflicts by Convention 258
 - Setting Up Conflict Checks Individually 260
 - Adding Timestamps 260
 - Resolving Conflicts..... 264
 - Pessimistic Locking on Entity Framework Core 270

Chapter 12: Logging	275
Using the Extension Method Log().....	275
Implementing the Log() Extension Method.....	278
Logging Categories	284
Chapter 13: Asynchronous Programming.....	287
Asynchronous Extension Methods	287
ToListAsync()	288
SaveChangesAsync()	289
ForEachAsync().....	291
Chapter 14: Dynamic LINQ Queries.....	295
Creating LINQ Queries Step-by-Step.....	295
Expression Trees	297
Using Dynamic LINQ.....	302
Chapter 15: Reading and Modifying Data with SQL, Stored Procedures, and Table-Valued Functions	305
Writing Queries with FromSql().....	306
Using LINQ and SQL Together.....	308
Using Stored Procedures and Table-Valued Functions	311
Using Nonentity Classes as Result Sets.....	313
Using SQL DML Commands Without Result Sets	315
Chapter 16: Tips and Tricks for Mapping.....	317
Shadow Properties.....	317
Automatic Shadow Properties	317
Defining a Shadow Property	318
Getting the Output of All Shadow Properties of an Entity Class.....	319
Reading and Changing a Shadow Property	319
Writing LINQ Queries with Shadow Properties	321
Practical Example: Automatically Updating the Shadow Property Every Time You Save.....	322

TABLE OF CONTENTS

- Computed Columns..... 323
 - Automatic SELECT 323
 - Practical Example: Creating Columns with a Calculation Formula 324
 - Using Columns with a Calculation Formula 327
 - Using Columns with a Calculation Formula in Reverse Engineering 329
- Default Values 329
 - Defining Default Values for Forward Engineering 329
 - Using Default Values 331
 - Practical Example: Defaults Already Assigned When Creating the Object 333
 - Using Default Values for Reverse Engineering 334
- Table Splitting 335
- Sequence Objects (Sequences) 339
 - What Are Sequences?..... 340
 - Creating Sequences at Forward Engineering 340
 - Seeing Sequences in Action 342
- Alternative Keys 347
 - Defining Alternative Keys 348
 - Seeing Alternative Keys in Use 350
- Cascading Delete 355
 - Delete Options in Entity Framework Core..... 355
 - Looking at an Example 357
- Mapping of Database Views..... 364
 - Creating a Database View..... 365
 - Creating an Entity Class for the Database View 365
 - Including the Entity Class in the Context Class 366
 - Using the Database View 367
 - Challenge: Migrations..... 367
- Global Query Filters..... 370
 - Defining a Filter 370
 - Using Filters in LINQ 371
 - Practical Example: Ignoring a Filter..... 372

Global Query Filters for SQL Queries	373
Global Query Filters for Stored Procedures and Table-Valued Functions	373
Future Queries	374
Chapter 17: Performance Tuning	377
Process Model for Performance Optimization in Entity Framework Core	377
Best Practices for Your Own Performance Tests	378
Performance Comparison of Various Data Access Techniques in .NET	378
Optimizing Object Assignment	380
Bulk Operations.....	383
Single Delete	383
Optimization with Batching	384
Delete Without Loading with Pseudo-Objects	387
Using Classic SQL Instead of the Entity Framework Core API	388
Lambda Expressions for Mass Deletion with EFPlus.....	390
Bulk Update with EFPlus	393
Performance Optimization Through No-Tracking.....	394
Activating No-Tracking Mode	395
No-Tracking Mode Is Almost Always Possible	396
No-Tracking Mode in an Editable Data Grid.....	400
QueryTrackingBehavior and AsTracking().....	411
Consequences of No-Tracking Mode.....	413
Best Practices.....	414
Selecting the Best Loading Strategy.....	414
Caching	415
MemoryCache	415
CacheManager	418
Second-Level Caching with EFPlus.....	429
Setting Up a Second-Level Cache	430
Using the Second-Level Cache.....	431

TABLE OF CONTENTS

- Chapter 18: Software Architecture with Entity Framework Core 435**
 - Monolithic Model 435
 - Entity Framework Core as a Data Access Layer..... 436
 - Pure Business Logic..... 438
 - Business Classes and ViewModel Classes..... 440
 - Distributed Systems..... 441
 - Conclusion 445

- Chapter 19: Commercial Tools..... 447**
 - Entity Framework Core Power Tools 447
 - Features..... 447
 - Reverse Engineering with Entity Framework Core Power Tools..... 448
 - Charts with Entity Framework Core Power Tools..... 452
 - LINQPad 454
 - Using LINQPad 455
 - Including Data Sources..... 456
 - Executing LINQ Commands 460
 - Saving..... 462
 - Other LINQPad Drivers..... 462
 - Interactive Program Code Entry..... 462
 - Conclusion to LINQPad 464
 - Entity Developer..... 464
 - Selecting the ORM Technique..... 465
 - Reverse Engineering with Entity Developer 467
 - Forward Engineering with Entity Developer 480
 - Entity Framework Profiler 487
 - Integrating Entity Framework Profiler 488
 - Monitoring Commands with the Entity Framework Profiler 489
 - Alerts and Suggestions..... 492
 - Analysis..... 493
 - Command-Line Support and API..... 494
 - Conclusion to Entity Framework Profiler 494

Chapter 20: Additional Components	495
Oracle Database Driver by DevArt.....	495
Installation.....	496
Tools	496
Context Class.....	496
Entity Classes	497
Data Types	497
Entity Framework Plus.....	499
Second-Level Caching with EFSecondLevelCache.Core	500
Object-Object Mapping with AutoMapper	501
Object-to-Object Mapping via Reflection.....	504
AutoMapper	507
Looking at an Example	508
Configuring Mappings	511
Running the Mapping with Map().....	512
Using the Nonstatic API	513
Mapping Conventions	514
Changing Mapping Conventions	519
Profile Classes	520
Ignoring Subobjects.....	521
Custom Mappings.....	522
Type Conversions.....	525
Collections	528
Inheritance	529
Generic Classes	534
Additional Actions Before and After the Mapping	538
Performance.....	540
Conclusion to AutoMapper.....	542

TABLE OF CONTENTS

- Appendix A: Case Studies 543**
 - Using Entity Framework Core in an ASP.NET Core Application..... 543
 - Architecture 547
 - Entity 549
 - Entity Framework Core Context Class 553
 - Lifetime of Context Class in ASP.NET Core Applications 555
 - Business Logic 556
 - Web API 570
 - Using Entity Framework Core via Dependency Injection 587
 - Practical Example: Context Instance Pooling (DbContext Pooling) 592
 - Using Entity Framework Core in a Universal Windows Platform App..... 592
 - Architecture 593
 - Entity 594
 - Entity Framework Core Context Class 596
 - Start Code..... 597
 - Generated Database 598
 - Data Access Code 601
 - User Interface 606
 - Using Entity Framework Core in a Xamarin Cross-Platform App 608
 - Architecture 609
 - Entity 611
 - Entity Framework Core Context Class 611
 - Start Code..... 614
 - Generated Database 615
 - Data Access Code 615
 - User Interface 620
 - Many-to-Many Relationship to Oneself..... 622
- Appendix B: Internet Resources 631**
- Appendix C: New Features in Entity Framework Core 2.1 633**
- Index..... 635**

About the Author



Holger Schwichtenberg is a .NET MVP with more than 20 years of experience as a developer and trainer. He is currently a technical lead with the German company IT-Visions, where he consults and trains at companies throughout Europe. He also serves as a software architect for 5Minds IT-Solutions. Holger is a huge fan of Entity Framework (EF) and Entity Framework Core and regularly speaks about both. He has used EF in projects of all sizes, most recently on a big data project containing billions of records. He is a prolific writer, having published more than 65 books and 1,000 technical articles in well-known IT professional and developer journals, including MSDN. He has presented at events such as TechEd Europe, Microsoft IT Forum, Advanced Developer Conference, Microsoft Launch, MSDN Technical Summit, and others. Holger has a PhD in business informatics.

His company web sites are www.IT-Visions.de and www.5minds.de, and he regularly blogs at dotnet-doktor.de. His office can be reached at office@IT-Visions.de.

Introduction

I have always been a big fan of object-relational mapping (ORM); in fact, I developed my own OR mapper for my software development projects in the early days of .NET. I switched to the ADO.NET Entity Framework when Microsoft introduced it in .NET 3.5 Service Pack 1. Nowadays, I prefer its successor, Entity Framework Core. Interestingly, some of my projects are still running the classic Entity Framework. As Microsoft continues to do incremental releases of Entity Framework Core, many of the challenges and gripes developers had with earlier versions have gone away, so my plan is to switch the management of all my projects to Entity Framework Core.

The book you hold in your hands came from an idea I had to cover all the important database access scenarios. I hadn't found much collective information in one place and felt that a compendium could be of great value to others. In this book, you will be introduced to database access concepts, get hands-on experience installing Entity Framework Core, and learn about reverse engineering and forward engineering for existing or legacy databases. I'll delve into topics such as schema migrations, data reading, and data modification with LINQ, Dynamic LINQ, APIs, SQL, stored procedures and table-valued functions, object relationships, and asynchronous programming. I'll also talk about third-party products such as LINQPad, Entity Developer, Entity Framework Profiler, Entity Framework Plus, and AutoMapper.

I'll discuss how to apply Entity Framework Core through case studies using Universal Windows Platform (UWP) apps, Xamarin, and ASP.NET Core. Of course, no book would be complete without sharing a healthy dose of hard-earned tips and tricks from my experience with Entity Framework and Entity Framework Core over the years.

Expectations of the Reader

This book is intended for software developers who have experience with .NET and C# as well as some relational database experience and who now want to use Entity Framework Core to create data access code in .NET, .NET Core, UWP apps, and Xamarin. Previous knowledge in predecessor technologies such as ADO.NET and the classic ADO.NET Entity Framework is useful but not necessary to understand this book.

Programming Language Used in This Book

I chose to use C# in this book because it is by far the most commonly used programming language in .NET. While I still occasionally develop .NET applications in Visual Basic .NET, it doesn't make sense to print all the listings in both languages.

If you are interested, a language converter between C# and Visual Basic .NET is freely available on several web sites, including <http://converter.telerik.com> and <https://www.mindfusion.eu/convert-cs-vb.html>.

The Use of Case Studies and Fictitious Enterprises

Most of the sample code in this book revolves around the fictitious airline World Wide Wings, abbreviated as WWWings or just WWW (see Figure 1).



Figure 1. Logo of the fictional airline World Wide Wings

Note You'll see other case studies used in some chapters, such as the task management app MiracleList.

The World Wide Wings use case deals with the following entities:

- *Flights* between two places where the places were deliberately not modeled as separate entities but as strings (this simplifies the understanding of many examples).
- *Passengers* flying on a flight.
- *Employees* of the airline, who have supervisors who are also employees.
- *Pilots* as a specialization of employees. A flight has only one pilot. There are no copilots at World Wide Wings.

- *Persons* as a collection of common characteristics for all people in this example. A person is not available on their own, but only in one of three specializations: passenger, employee, and pilot. In the object-oriented sense, therefore, Person is an abstract base class that cannot own instances but is used only for inheritance.

The World Wide Wings use case has two data models, explained here:

- The slightly simpler model version 1 (see Figures 2 and 3) is the result of classic relational database design with normalization. The object model is created by reverse engineering.

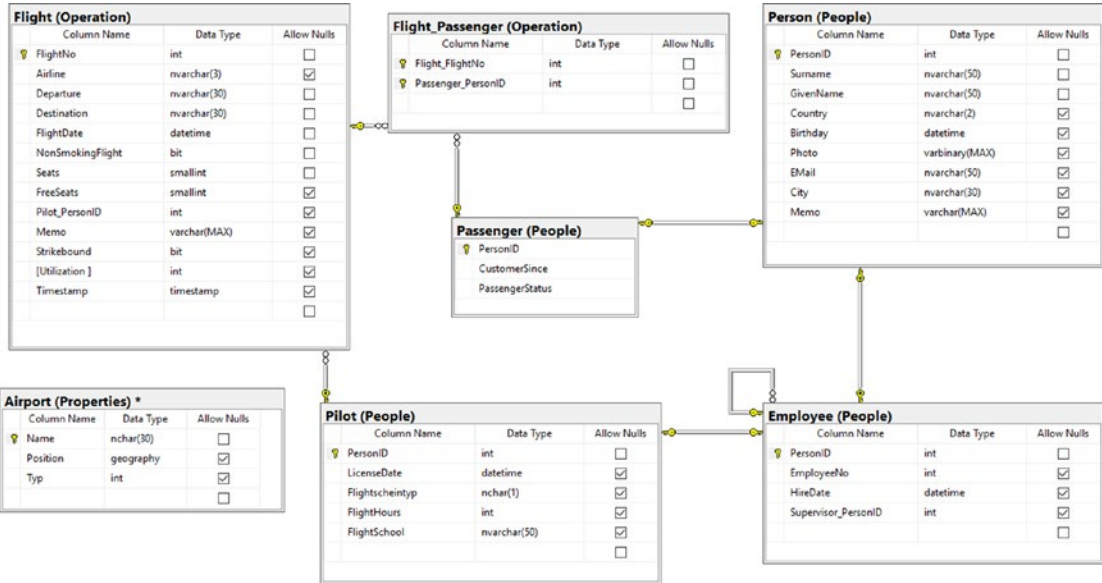


Figure 2. World Wide Wings data model in the simpler version 1

INTRODUCTION

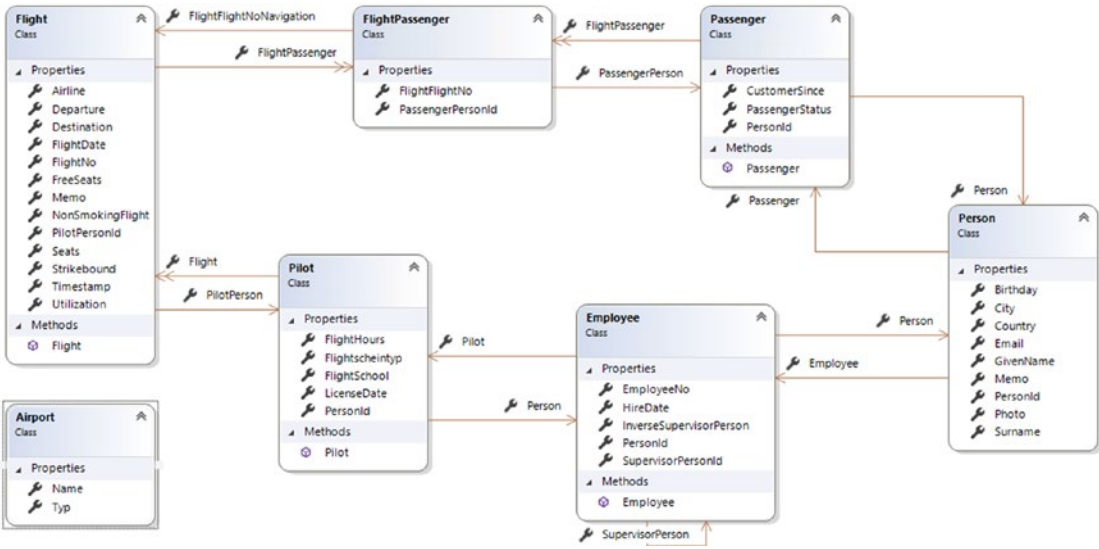


Figure 3. Object model of the World Wide Wings data model in the simpler version 1

- Model version 2 (see Figures 4 and 5) is the result of forward engineering with Entity Framework Core from an object model. In addition, there are other entities (Airline, Persondetail, AircraftType, and AircraftTypeDetail) in this model to show further modeling aspects. In this case, there is an optional copilot for each flight.

In model version 1 there is a separate table for people (called Person), staff, pilots, employee, and passengers. This separation corresponds to the classes in the object model.

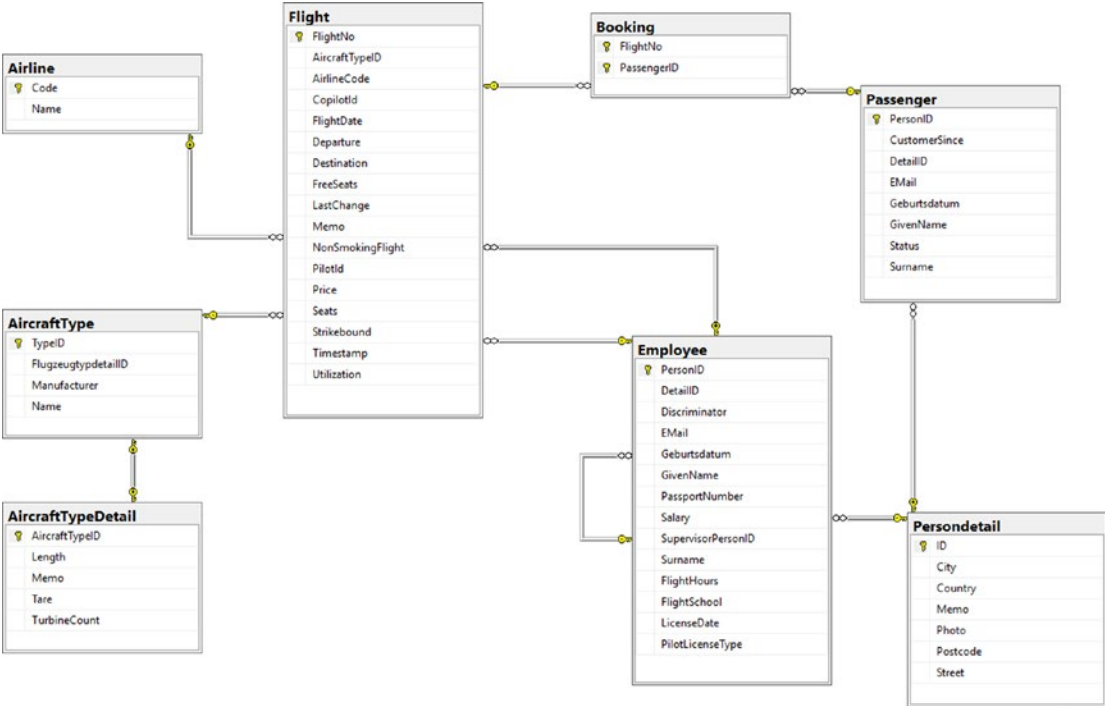


Figure 4. World Wide Wings data model in the more complex version 2

INTRODUCTION

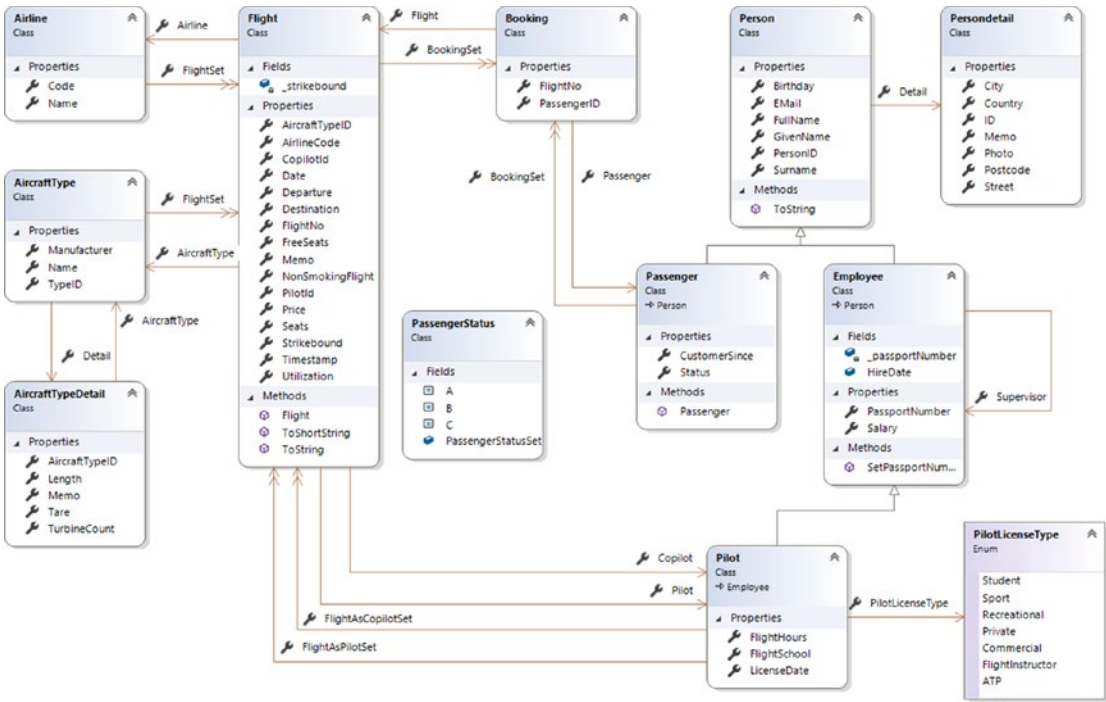


Figure 5. Object model for the World Wide Wings data model in the more complex version 2

Note The object models that were created in this book for the data models do not represent an ideal object model because Entity Framework Core does not support some mapping capabilities, such as N:M mapping, yet.

The object model for the data schema of World Wide Wings version 6.1 (Figure 3) was automatically generated by the Entity Framework Core from the database (through reverse engineering); I deliberately did not change it, even if some of the generated names are ugly.

In model version 2, there are only the Passenger and Employee tables for these four entities. Entity Framework Core is currently somewhat limited and does not support table per type mapping (a separate table for each class). Therefore, the table Passenger also includes all the characteristics of Person. In addition to the Person properties, the

Employee table includes the properties of the Employee and Pilot entities. In the table, a Discriminator column distinguishes between records that are an employee and those that are a pilot. Entity Framework Core mixes the concepts of table by concrete type (TPC) and table by hierarchy (TPH). The developer has no definite influence on the inheritance mapping in Entity Framework Core 1.x/2.0. The classic Entity Framework offers more options here.

The following are the extra dependencies in model version 2:

- A Flight belongs to Airline (there will be only World Wide Wings and its subsidiary Never Come Back Airline in this book).
- There a Copilot entity here, but it is optional.
- A Flight can optionally have an AircraftType object assigned. AircraftType must have an AircraftTypeDetail object.
- Each Person and therefore each Pilot and Passenger must own a Persondetail object.

In this book, both data models are used, partly in modified form, to show certain scenarios (for example, database schema migrations).

Application Types in This Book

In this book, the examples are for the most time shown via a text-based console in console applications because this allows me to focus on database access. When using graphical user interfaces such as WPF, Windows Forms, ASP.NET Web Forms, or ASP.NET MVC, the representation is decoupled by data binding, which means I would always need to show a second listing so you could understand that the data access was actually delivered. I simulate user inputs in the console examples by writing variables at the beginning of the program code.

I have provided training and consultancy on data access for many years and have learned that console editions are didactically the best tool for teaching because otherwise the listings are large and thus inferior.

Of course, console output is not common practice in 99 percent of software development, but graphical user interfaces are covered in other books, and data binding typically has no impact on the form of data access. Where data access is relevant, this book will also show data binding examples.

Helper Functions for Console Output

I will show the screen output on the console using the standard method `Console.WriteLine()` in several places; in addition, I use auxiliary routines that generate colored screen output. Listing 1 shows these auxiliary routines in the class CUI from `ITV_DemoUtil.dll` for a better understanding.

Listing 1. Class CUI with Subroutines for Screen Output to the Console

```

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

namespace ITVisions
{
    /// <summary>
    /// Helper utilities for Console UIs
    /// (C) Dr. Holger Schwichtenberg 2002-2018
    /// </summary>
    public static class CUI
    {
        public static bool IsDebug = false;
        public static bool IsVerbose = false;

        #region Print only under certain conditions
        public static void PrintDebug(object s)
        {
            PrintDebug(s, System.Console.ForegroundColor);
        }

        public static void PrintVerbose(object s)
        {
            PrintVerbose(s, System.Console.ForegroundColor);
        }
        #endregion
    }
}

```

```
#region Print with predefined colors
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 PrintStep(object s)
{
    Print(s, ConsoleColor.Cyan);
}

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

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

INTRODUCTION

```
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 Print with selectable color
```

```

public static void Print(object s, ConsoleColor frontcolor, ConsoleColor?
backcolor = null)
{
    PrintInternal(s, frontcolor, backcolor);
}

```

```

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

```

```

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

```

#endregion

#region Print with additional data

```

/// <summary>
/// Print with 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>
/// Print with time
/// </summary>
public static void PrintWithTime(object s, ConsoleColor c = ConsoleColor.
White)

```

INTRODUCTION

```
{
    CUI.Print(DateTime.Now.Second + "." + DateTime.Now.Millisecond + ":" + s);
}

private static long count;
/// <summary>
/// Print with counter
/// </summary>
private static void PrintWithCounter(object s, ConsoleColor frontcolor,
ConsoleColor? backcolor = null)
{
    count += 1;
    s = $"{count:0000}: {s}";
    CUI.Print(s, frontcolor, backcolor);
}

#endregion

#region internal helper routines
private static void PrintDebugOrVerbose(object s, ConsoleColor
frontcolor, ConsoleColor? backcolor = null)
{
    count += 1;
    s = $"{count:0000}: {s}";
    Print(s, frontcolor, backcolor);
    Debug.WriteLine(s);
    Trace.WriteLine(s);
    Trace.Flush();
}

/// <summary>
/// Output to console, trace and file
/// </summary>
/// <param name="s"></param>
[DebuggerStepThrough()]
```

```

private static void PrintInternal(object s, ConsoleColor? frontcolor =
null, ConsoleColor? backcolor = null)
{
    if (s == null) return;

    if (HttpContext.Current != null)
    {
        try
        {
            if (frontcolor != null)
            {
                HttpContext.Current.Response.Write("<span style='color:" +
                frontcolor.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 (frontcolor != null)
            {
                HttpContext.Current.Response.Write("</span>");
            }
        }
        catch (Exception)
        {
        }
    }
    else
    {
        object x = 1;
        lock (x)
        {

```