

УДК 004.4
ББК 32.372
С50

Под редакцией сообщества .NET разработчиков DotNet.Ru

Смит Дж. П.
С50 Entity Framework Core в действии / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2023. – 690 с.: ил.

ISBN 978-5-93700-114-6

Entity Framework радикально упрощает доступ к данным в приложениях .NET. Этот простой в использовании инструмент объектно-реляционного отображения (ORM) позволяет писать код базы данных на чистом C#. Он автоматически отображает классы в таблицы базы данных, разрешает запросы со стандартными командами LINQ и даже генерирует SQL-код за вас.

Данная книга научит вас писать код для беспрепятственного взаимодействия с базой данных при работе с приложениями .NET. Следуя соответствующим примерам из обширного опыта автора книги, вы быстро перейдете от основ к продвинутым методам. Помимо новейших функциональных возможностей EF, в книге рассматриваются вопросы производительности, безопасности, рефакторинга и модульного тестирования.

Издание предназначено разработчикам .NET, знакомым с реляционными базами данных.

УДК 004.4
ББК 32.372

Original English language edition published by Manning Publications USA, USA. Copyright © 2021 by Manning Publications. Russian-language edition copyright © 2023 DMC Press. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-6172-9836-3 (англ.)
ISBN 978-5-93700-114-6 (рус.)

© Manning Publications, 2021
© Перевод, оформление, издание, ДМК Пресс, 2022

Содержание

Предисловие	21
Введение	23
Благодарности	25
Об этой книге	26
Об авторе	32
Об изображении на обложке	33

Часть I НАЧАЛО	34
-----------------------------	-----------

1 Введение в Entity Framework Core	36
1.1 Что вы узнаете из этой книги	37
1.2 Мой «момент озарения»	38
1.3 Несколько слов для разработчиков EF6.x	40
1.4 Обзор EF Core	40
1.4.1 Недостатки инструментов объектно-реляционного отображения	41
1.5 Что насчет нереляционных (NoSQL) баз данных?	42
1.6 Ваше первое приложение, использующее EF Core	42
1.6.1 Что нужно установить	43
1.6.2 Создание собственного консольного приложения .NET Core с помощью EF Core	44
1.7 База данных, к которой будет обращаться MyFirstEfCoreApplication	45
1.8 Настройка приложения MyFirstEfCoreApplication	47
1.8.1 Классы, которые отображаются в базу данных: Book и Author	47
1.8.2 DbContext	48
1.9 Заглянем под капот EF Core	49
1.9.1 Моделирование базы данных	50
1.9.2 Чтение данных	51
1.9.3 Обновление	54
1.10 Этапы разработки EF Core	57
1.11 Стоит ли использовать EF Core в своем следующем проекте?	58
1.11.1 .NET – это программная платформа будущего, и она будет быстрой! ...	58
1.11.2 Открытый исходный код и открытые сообщения	59
1.11.3 Мультиплатформенные приложения и разработка	59
1.11.4 Быстрая разработка и хорошие функциональные возможности	59
1.11.5 Хорошая поддержка	60
1.11.6 Всегда высокая производительность	60
1.12 Когда не следует использовать EF Core?	61
Резюме	61

2	Выполнение запроса к базе данных	63
2.1	Закладываем основу: наш сайт по продаже книг	64
2.1.1	Реляционная база данных приложения Book App	64
2.1.2	Другие типы связей, не описанные в этой главе	67
2.1.3	База данных – все таблицы	68
2.1.4	Классы, которые EF Core отображает в базу данных	70
2.2	Создание DbContext	72
2.2.1	Определение DbContext приложения: EfCoreContext	72
2.2.2	Создание экземпляра DbContext приложения	73
2.2.3	Создание базы данных для своего приложения	74
2.3	Разбираемся с запросами к базе данных	75
2.3.1	Доступ к свойству DbContext приложения	76
2.3.2	Серия команд LINQ / EF Core	76
2.3.3	Команда выполнения	76
2.3.4	Два типа запросов к базе данных	77
2.4	Загрузка связанных данных	78
2.4.1	Немедленная загрузка: загрузка связей с первичным классом сущности	78
2.4.2	Явная загрузка: загрузка связей после первичного класса сущности	81
2.4.3	Выборочная загрузка: загрузка определенных частей первичного класса сущности и любых связей	82
2.4.4	Отложенная загрузка: загрузка связанных данных по мере необходимости	83
2.5	Использование вычисления на стороне клиента: адаптация данных на последнем этапе запроса	85
2.6	Создание сложных запросов	88
2.7	Знакомство с архитектурой приложения Book App	92
2.8	Добавляем сортировку, фильтрацию и разбиение на страницы	93
2.8.1	Сортировка книг по цене, дате публикации и оценкам покупателей	94
2.8.2	Фильтрация книг по году публикации, категориям и оценкам покупателей	95
2.8.3	Другие параметры фильтрации: поиск текста по определенной строке	96
2.8.4	Разбиение книг на страницы в списке	98
2.9	Собираем все вместе: объединение объектов запроса	99
	Резюме	100
3	Изменение содержимого базы данных	102
3.1	Представляем свойство сущности State	103
3.2	Создание новых строк в таблице	103
3.2.1	Самостоятельное создание отдельной сущности	104
3.2.2	Создание книги с отзывом	105
3.3	Обновление строк базы данных	109
3.3.1	Обработка отключенных обновлений в веб-приложении	111
3.4	Обработка связей в обновлениях	117
3.4.1	Основные и зависимые связи	118
3.4.2	Обновление связей «один к одному»: добавляем PriceOffer в книгу	119
3.4.3	Обновление связей «один ко многим»: добавляем отзыв в книгу	123
3.4.4	Обновление связи «многие ко многим»	127
3.4.5	Расширенная функция: обновление связей через внешние ключи	132
3.5	Удаление сущностей	133
3.5.1	Мягкое удаление: использование глобального фильтра запросов, чтобы скрыть сущности	133
3.5.2	Удаление только зависимой сущности без связей	135

3.5.3	Удаление основной сущности, у которой есть связи	135
3.5.4	Удаление книги с зависимыми связями	136
Резюме		137

4	Использование EF Core в бизнес-логике	139
4.1	Вопросы, которые нужно задать, и решения, которые нужно принять, прежде чем начать писать код	140
4.1.1	Три уровня сложности кода бизнес-логики	141
4.2	Пример сложной бизнес-логики: обработка заказа на приобретение книги.....	143
4.3	Использование паттерна проектирования для реализации сложной бизнес-логики.....	144
4.3.1	Пять правил по созданию бизнес-логики, использующей EF Core	144
4.4	Реализация бизнес-логики для обработки заказа	146
4.4.1	Правило 1: бизнес-логика требует определения структуры базы данных	147
4.4.2	Правило 2: ничто не должно отвлекать от бизнес-логики	148
4.4.3	Правило 3: бизнес-логика должна думать, что работает с данными в памяти	149
4.4.4	Правило 4: изолируйте код доступа к базе данных в отдельный проект	152
4.4.5	Правило 5: бизнес-логика не должна вызывать метод EF Core, SaveChanges.....	153
4.4.6	Собираем все вместе: вызов бизнес-логики для обработки заказов ...	156
4.4.7	Размещение заказа в приложении Book App.....	157
4.4.8	Плюсы и минусы паттерна сложной бизнес-логики	159
4.5	Пример простой бизнес-логики: ChangePriceOfferService	159
4.5.1	Мой подход к проектированию простой бизнес-логики	160
4.5.2	Пишем код класса ChangePriceOfferService	160
4.5.3	Плюсы и минусы этого паттерна бизнес-логики	161
4.6	Пример валидации: добавление отзыва в книгу с проверкой	162
4.6.1	Плюсы и минусы этого паттерна бизнес-логики.....	163
4.7	Добавление дополнительных функций в обработку вашей бизнес-логики	163
4.7.1	Валидация данных, которые вы записываете в базу	164
4.7.2	Использование транзакций для объединения кода бизнес-логики в одну логическую атомарную операцию	168
4.7.3	Использование класса RunnerTransact2WriteDb	172
Резюме		173

5	Использование EF Core в веб-приложениях ASP.NET Core	175
5.1	Знакомство с ASP.NET Core	176
5.2	Разбираемся с архитектурой приложения Book App.....	176
5.3	Внедрение зависимостей	177
5.3.1	Почему нужно знать, что такое внедрение зависимостей, работая с ASP.NET Core	179
5.3.2	Базовый пример внедрения зависимостей в ASP.NET Core	179
5.3.3	Жизненный цикл сервиса, созданного внедрением зависимостей	180
5.3.4	Особые соображения, касающиеся приложений Blazor Server.....	182
5.4	Делаем DbContext приложения доступным, используя внедрение зависимостей	182
5.4.1	Предоставление информации о расположении базы данных	183

5.4.2	Регистрация DbContext приложения у поставщика внедрения зависимостей	184
5.4.3	Регистрация фабрики DbContext у поставщика внедрения зависимостей	185
5.5	Вызов кода доступа к базе данных из ASP.NET Core	186
5.5.1	Краткое изложение того, как работает паттерн ASP.NET Core MVC, и термины, которые он использует	187
5.5.2	Где находится код EF Core в приложении Book App?	187
5.6	Реализация страницы запроса списка книг	189
5.6.1	Внедрение экземпляра DbContext приложения через внедрение зависимостей	189
5.6.2	Использование фабрики DbContext для создания экземпляра DbContext	191
5.7	Реализация методов базы данных как сервиса внедрения зависимостей	193
5.7.1	Регистрация класса в качестве сервиса во внедрении зависимостей	194
5.7.2	Внедрение ChangePubDateService в метод действия ASP.NET	195
5.7.3	Улучшаем регистрацию классов доступа к базе данных как сервисов ..	196
5.8	Развертывание приложения ASP.NET Core с базой данных	199
5.8.1	Местонахождение базы данных на веб-сервере	200
5.8.2	Создание и миграция базы данных	201
5.9	Использование функции миграции в EF Core для изменения структуры базы данных	201
5.9.1	Обновление рабочей базы данных	202
5.9.2	Заставляем приложение обновить базу данных при запуске	203
5.10	Использование async/await для лучшей масштабируемости	206
5.10.1	Чем паттерн async/await полезен в веб-приложении, использующем EF Core	207
5.10.2	Где использовать async/await для доступа к базе данных?	208
5.10.3	Переход на версии команд EF Core с async/await	208
5.11	Выполнение параллельных задач: как предоставить DbContext	210
5.11.1	Получение экземпляра DbContext для параллельного запуска	211
5.11.2	Запуск фоновой службы в ASP.NET Core	212
5.11.3	Другие способы получения нового экземпляра DbContext	213
	Резюме	213

6	Советы и техники, касающиеся чтения и записи данных с EF Core	215
6.1	Чтение из базы данных	216
6.1.1	Этап ссылочной фиксации в запросе	216
6.1.2	Понимание того, что делает метод AsNoTracking и его разновидности	218
6.1.3	Эффективное чтение иерархических данных	220
6.1.4	Понимание того, как работает метод Include	222
6.1.5	Обеспечение отказоустойчивости загрузки навигационных коллекций	224
6.1.6	Использование глобальных фильтров запросов в реальных ситуациях ..	225
6.1.7	Команды LINQ, требующие особого внимания	230
6.1.8	Использование AutoMapper для автоматического построения запросов с методом Select	232
6.1.9	Оценка того, как EF Core создает класс сущности при чтении данных	235
6.2	Запись данных в базу с EF Core	240
6.2.1	Оценка того, как EF Core записывает сущности или связи	

	в базу данных.....	240
6.2.2	Оценка того, как DbContext обрабатывает запись сущностей и связей	242
6.2.3	Быстрый способ копирования данных со связями	246
6.2.4	Быстрый способ удалить сущность	247
Резюме.....		248

Часть II ОБ ENTITY FRAMEWORK В ДЕТАЛЯХ 250

7	Настройка нереляционных свойств	252
7.1	Три способа настройки EF Core.....	253
7.2	Рабочий пример настройки EF Core.....	254
7.3	Конфигурация по соглашению	257
7.3.1	Соглашения для классов сущностей.....	257
7.3.2	Соглашения для параметров в классе сущности.....	258
7.3.3	Условные обозначения для имени, типа и размера	258
7.3.4	По соглашению поддержка значения NULL для свойства основана на типе .NET	259
7.3.5	Соглашение об именах EF Core определяет первичные ключи	259
7.4	Настройка с помощью аннотаций данных	260
7.4.1	Использование аннотаций из пространства имен System.ComponentModel.DataAnnotations	261
7.4.2	Использование аннотаций из пространства имен System.ComponentModel.DataAnnotations.Schema	261
7.5	Настройка с использованием Fluent API.....	261
7.6	Исключение свойств и классов из базы данных.....	264
7.6.1	Исключение класса или свойства с помощью Data Annotations.....	264
7.6.2	Исключение класса или свойства с помощью Fluent API	265
7.7	Установка типа, размера и допустимости значений NULL для столбца базы данных	266
7.8	Преобразование значения: изменение данных при чтении из базы данных или записи в нее	267
7.9	Различные способы настройки первичного ключа.....	269
7.9.1	Настройка первичного ключа с помощью Data Annotations.....	269
7.9.2	Настройка первичного ключа через Fluent API	270
7.9.3	Настройка сущности как класса с доступом только на чтение	270
7.10	Добавление индексов в столбцы базы данных	271
7.11	Настройка именования на стороне базы данных.....	272
7.11.1	Настройка имен таблиц	273
7.11.2	Настройка имени схемы и группировки схем	273
7.11.3	Настройка имен столбцов базы данных в таблице	274
7.12	Настройка глобальных фильтров запросов	274
7.13	Применение методов Fluent API в зависимости от типа поставщика базы данных.....	275
7.14	Теневые свойства: сокрытие данных столбца внутри EF Core	276
7.14.1	Настройка теневых свойств	277
7.14.2	Доступ к теневым свойствам	277
7.15	Резервные поля: управление доступом к данным в классе сущности.....	278
7.15.1	Создание простого резервного поля, доступного через свойство чтения/записи	279
7.15.2	Создание столбца с доступом только на чтение.....	279
7.15.3	Сокрытие даты рождения внутри класса	280
7.15.4	Настройка резервных полей.....	281
7.16	Рекомендации по использованию конфигурации EF Core	283

7.16.1	Сначала используйте конфигурацию «По соглашению»	284
7.16.2	По возможности используйте Data Annotations	284
7.16.3	Используйте Fluent API для всего остального	284
7.16.4	Автоматизируйте добавление команд Fluent API по сигнатурам класса или свойства	285
Резюме		289

8	Конфигурирование связей	291
8.1	Определение терминов, относящихся к связям	292
8.2	Какие навигационные свойства нам нужны?	293
8.3	Настройка связей	294
8.4	Настройка связей по соглашению	295
8.4.1	Что делает класс классом сущности?	295
8.4.2	Пример класса сущности с навигационными свойствами	295
8.4.3	Как EF Core находит внешние ключи по соглашению	296
8.4.4	Поддержка значения null у внешних ключей: обязательные или необязательные зависимые связи	297
8.4.5	Внешние ключи: что произойдет, если не указать их?	298
8.4.6	Когда подход «По соглашению» не работает?	300
8.5	Настройка связей с помощью аннотаций данных	300
8.5.1	Аннотация ForeignKey	300
8.5.2	Аннотация InverseProperty	301
8.6	Команды Fluent API для настройки связей	302
8.6.1	Создание связи «один к одному»	303
8.6.2	Создание связи «один ко многим»	306
8.6.3	Создание связей «многие ко многим»	307
8.7	Управление обновлениями навигационных свойств коллекции	310
8.8	Дополнительные методы, доступные во Fluent API	312
8.8.1	OnDelete: изменение действия при удалении зависимой сущности	313
8.8.2	IsRequired: определение допустимости значения null для внешнего ключа	316
8.8.3	HasPrincipalKey: использование альтернативного уникального ключа ...	318
8.8.4	Менее используемые параметры в связях Fluent API	319
8.9	Альтернативные способы отображения сущностей в таблицы базы данных	320
8.9.1	Собственные типы: добавление обычного класса в класс сущности	320
8.9.2	Таблица на иерархию (TPH): размещение унаследованных классов в одной таблице	326
8.9.3	Таблица на тип (TPT): у каждого класса своя таблица	331
8.9.4	Разбиение таблицы: отображение нескольких классов сущностей в одну и ту же таблицу	333
8.9.5	Контейнер свойств: использование словаря в качестве класса сущности	335
Резюме		337

9	Управление миграциями базы данных	339
9.1	Как устроена эта глава	340
9.2	Сложности изменения базы данных приложения	340
9.2.1	Какие базы данных нуждаются в обновлении	341
9.2.2	Миграция, которая может привести к потере данных	342
9.3	Часть 1: знакомство с тремя подходами к созданию миграции	342
9.4	Создание миграции с помощью команды EF Core add migration	344
9.4.1	Требования перед запуском любой команды миграции EF Core	346

9.4.2	Запуск команды <i>add migration</i>	347
9.4.3	Заполнение базы данных с помощью миграции	348
9.4.4	Миграции и несколько разработчиков	349
9.4.5	Использование собственной таблицы миграций, позволяющей использовать несколько <i>DbContext</i> в одной базе данных	350
9.5	Редактирование миграции для обработки сложных ситуаций	353
9.5.1	Добавление и удаление методов <i>MigrationBuilder</i> внутри класса миграции	354
9.5.2	Добавление команд <i>SQL</i> в миграцию	355
9.5.3	Добавление собственных команд миграции	357
9.5.4	Изменение миграции для работы с несколькими типами баз данных	358
9.6	Использование сценариев <i>SQL</i> для создания миграций	360
9.6.1	Использование инструментов сравнения баз данных <i>SQL</i> для выполнения миграции	361
9.6.2	Написание кода сценариев изменения <i>SQL</i> для миграции базы данных вручную	363
9.6.3	Проверка соответствия сценариев изменения <i>SQL</i> модели базы данных <i>EF Core</i>	365
9.7	Использование инструмента обратного проектирования <i>EF Core</i>	366
9.7.1	Запуск команды обратного проектирования	367
9.7.2	Установка и запуск команды обратного проектирования <i>Power Tools</i>	368
9.7.3	Обновление классов сущности и <i>DbContext</i> при изменении базы данных	368
9.8	Часть 2: применение миграций к базе данных	369
9.8.1	Вызов метода <i>Database.Migrate</i> из основного приложения	370
9.8.2	Выполнение метода <i>Database.Migrate</i> из отдельного приложения	373
9.8.3	Применение миграции <i>EF Core</i> с помощью <i>SQL</i> -сценария	373
9.8.4	Применение сценариев изменения <i>SQL</i> с помощью инструмента миграций	375
9.9	Миграция базы данных во время работы приложения	375
9.9.1	Миграция, которая не содержит критических изменений	377
9.9.2	Работа с критическими изменениями, когда вы не можете остановить приложение	378
	Резюме	380

10 Настройка расширенных функций и разрешение конфликтов параллельного доступа

10.1	<i>DbFunction</i> : использование пользовательских функций с <i>EF Core</i>	383
10.1.1	Настройка скалярной функции	384
10.1.2	Настройка табличной функции	386
10.1.3	Добавление кода пользовательской функции в базу данных	387
10.1.4	Использование зарегистрированной пользовательской функции в запросах к базе данных	388
10.2	Вычисляемый столбец: динамически вычисляемое значение столбца	389
10.3	Установка значения по умолчанию для столбца базы данных	392
10.3.1	Использование метода <i>HasDefaultValue</i> для добавления постоянного значения для столбца	394
10.3.2	Использование метода <i>HasDefaultValueSql</i> для добавления команды <i>SQL</i> для столбца	395
10.3.3	Использование метода <i>HasValueGenerator</i> для назначения генератора значений свойству	396
10.4	Последовательности: предоставление чисел в строгом порядке	397

10.5	Помечаем свойства, созданные базой данных	398
10.5.1	Помечаем столбец, создаваемый при добавлении или обновлении	399
10.5.2	Помечаем значение столбца как установленное при вставке новой строки	400
10.5.3	Помечаем столбец/свойство как «обычное»	401
10.6	Одновременные обновления: конфликты параллельного доступа	402
10.6.1	Почему конфликты параллельного доступа так важны?	403
10.6.2	Возможности решения конфликтов параллельного доступа в EF Core	404
10.6.3	Обработка исключения <code>DbUpdateConcurrencyException</code>	411
10.6.4	Проблема с отключенным параллельным обновлением	415
	Резюме	419

11

	Углубляемся в DbContext	420
11.1	Обзор свойств класса <code>DbContext</code>	421
11.2	Как EF Core отслеживает изменения	421
11.3	Обзор команд, которые изменяют свойство сущности <code>State</code>	423
11.3.1	Команда <code>Add</code> : вставка новой строки в базу данных	424
11.3.2	Метод <code>Remove</code> : удаление строки из базы данных	425
11.3.3	Изменение класса сущности путем изменения данных в нем	425
11.3.4	Изменение класса сущности путем вызова метода <code>Update</code>	426
11.3.5	Метод <code>Attach</code> : начать отслеживание существующего неотслеживаемого класса сущности	428
11.3.6	Установка свойства сущности <code>State</code> напрямую	428
11.3.7	<code>TrackGraph</code> : обработка отключенных обновлений со связями	429
11.4	Метод <code>SaveChanges</code> и как он использует метод <code>ChangeTracker.DetectChanges</code>	431
11.4.1	Как метод <code>SaveChanges</code> находит все изменения состояния	432
11.4.2	Что делать, если метод <code>ChangeTracker.DetectChanges</code> занимает слишком много времени	432
11.4.3	Использование состояния сущностей в методе <code>SaveChanges</code>	437
11.4.4	Перехват изменений свойства <code>State</code> с использованием события	441
11.4.5	Запуск событий при вызове методов <code>SaveChanges</code> и <code>SaveChangesAsync</code>	444
11.4.6	Перехватчики EF Core	445
11.5	Использование команд SQL в приложении EF Core	445
11.5.1	Методы <code>FromSqlRaw/FromSqlInterpolated</code> : использование SQL в запросе EF Core	447
11.5.2	Методы <code>ExecuteSqlRaw</code> и <code>ExecuteSqlInterpolated</code> : выполнение команды без получения результата	448
11.5.3	Метод <code>Fluent API ToSqlQuery</code> : отображение классов сущностей в запросы	448
11.5.4	Метод <code>Reload</code> : используется после команд <code>ExecuteSql</code>	450
11.5.5	<code>GetDbConnection</code> : выполнение собственных команд SQL	450
11.6	Доступ к информации о классах сущностей и таблицах базы данных	452
11.6.1	Использование <code>context.Entry(entity).Metadata</code> для сброса первичных ключей	452
11.6.2	Использование свойства <code>context.Model</code> для получения информации о базе данных	455
11.7	Динамическое изменение строки подключения <code>DbContext</code>	456
11.8	Решение проблем, связанных с подключением к базе данных	457
11.8.1	Обработка транзакций базы данных с использованием стратегии выполнения	458
11.8.2	Изменение или написание собственной стратегии исполнения	460
	Резюме	460

Часть III ИСПОЛЬЗОВАНИЕ ENTITY FRAMEWORK CORE В РЕАЛЬНЫХ ПРИЛОЖЕНИЯХ 462

12	Использование событий сущности для решения проблем бизнес-логики.....	464
12.1	Использование событий для решения проблем бизнес-логики	465
12.1.1	Пример использования событий предметной области	465
12.1.2	Пример событий интеграции	467
12.2	Определяем, где могут быть полезны события предметной области и интеграции.....	468
12.3	Где можно использовать события с EF Core?	468
12.3.1	Плюс: следует принципу разделения ответственностей	470
12.3.2	Плюс: делает обновления базы данных надежными	470
12.3.3	Минус: делает приложение более сложным	470
12.3.4	Минус: усложняет отслеживание потока исполнения кода	471
12.4	Реализация системы событий предметной области с EF Core	472
12.4.1	Создайте несколько классов событий предметной области, которые нужно будет вызвать	473
12.4.2	Добавьте код в классы сущностей, где будут храниться события предметной области.....	474
12.4.3	Измените класс сущности, чтобы обнаружить изменение, при котором вызывается событие	475
12.4.4	Создайте обработчики событий, соответствующие событиям предметной области	475
12.4.5	Создайте диспетчер событий, который находит и запускает правильный обработчик событий.....	477
12.4.6	Переопределите метод SaveChanges и вставьте вызов диспетчера событий перед вызовом этого метода	479
12.4.7	Зарегистрируйте диспетчер событий и все обработчики событий ...	480
12.5	Внедрение системы событий интеграции с EF Core	482
12.5.1	Создание сервиса, который обменивается данными со складом	484
12.5.2	Переопределение метода SaveChanges для обработки события интеграции	485
12.6	Улучшение события предметной области и реализаций событий интеграции	486
12.6.1	Обобщение событий: запуск до, во время и после вызова метода SaveChanges	487
12.6.2	Добавление поддержки асинхронных обработчиков событий	488
12.6.3	Несколько обработчиков событий для одного и того же события	489
12.6.4	Последовательности событий, в которых одно событие запускает другое	490
	Резюме.....	491

13	Предметно-ориентированное проектирование и другие архитектурные подходы	492
13.1	Хорошая программная архитектура упрощает создание и сопровождение приложения.....	493
13.2	Развивающаяся архитектура приложения Book App	494
13.2.1	Создание модульного монолита для обеспечения реализации принципов разделения ответственностей	495
13.2.2	Использование принципов предметно-ориентированного проектирования в архитектуре и в классах сущностей	497

13.2.3	Применение чистой архитектуры согласно описанию Роберта Мартина	498
13.3	Введение в предметно-ориентированное проектирование на уровне класса сущности	498
13.4	Изменение сущностей приложения Book App, чтобы следовать предметно-ориентированному проектированию	499
13.4.1	Изменение свойств сущности Book на доступ только для чтения	500
13.4.2	Обновление свойств сущности Book с помощью методов в классе сущности	502
13.4.3	Управление процессом создания сущности Book	503
13.4.4	Разбор различий между сущностями и объектом-значением	505
13.4.5	Минимизация связей между классами сущностей	505
13.4.6	Группировка классов сущностей	506
13.4.7	Принимаем решение, когда бизнес-логику не следует помещать внутрь сущности	508
13.4.8	Применение паттерна «Ограниченный контекст» к DbContext приложения	510
13.5	Использование классов сущностей в стиле DDD в вашем приложении	511
13.5.1	Вызов метода доступа AddPromotion с помощью паттерна «Репозиторий»	512
13.5.2	Вызов метода доступа AddPromotion с помощью библиотеки GenericServices	515
13.5.3	Добавление отзыва в класс сущности Book через паттерн «Репозиторий»	517
13.5.4	Добавление отзыва в класс сущности Book с помощью библиотеки GenericServices	518
13.6	Обратная сторона сущностей DDD: слишком много методов доступа	519
13.7	Решение проблем с производительностью в DDD-сущностях	520
13.7.1	Добавить код базы данных в свои классы сущностей	521
13.7.2	Сделать конструктор Review открытым и написать код для добавления отзыва вне сущности	523
13.7.3	Использовать события предметной области, чтобы попросить обработчик событий добавить отзыв в базу данных	523
13.8	Три архитектурных подхода: работали ли они?	524
13.8.1	Модульный монолит, реализующий принцип разделения ответственностей с помощью проектов	524
13.8.2	Принципы DDD как в архитектуре, так и в классах сущностей	526
13.8.3	Чистая архитектура согласно описанию Роберта С. Мартина	527
	Резюме	528

14 Настройка производительности в EF Core

14.1	Часть 1: решаем, какие проблемы с производительностью нужно исправлять	531
14.1.1	Фраза «Не занимайтесь настройкой производительности на ранних этапах» не означает, что нужно перестать думать об этом	531
14.1.2	Как работает медленно и требует настройки производительности?	532
14.1.3	Затраты на поиск и устранение проблем с производительностью	534
14.2	Часть 2: методы диагностики проблем с производительностью	535
14.2.1	Этап 1. Получить хорошее общее представление, оценив опыт пользователей	536
14.2.2	Этап 2. Найти весь код доступа к базе данных, связанный с оптимизируемой функцией	537
14.2.3	Этап 3. Проверить SQL-код, чтобы выявить низкую производительность	538

14.3	Часть 3: методы устранения проблем с производительностью	540
14.4	Использование хороших паттернов позволяет приложению хорошо работать	541
14.4.1	Использование метода <i>Select</i> для загрузки только нужных столбцов	542
14.4.2	Использование разбиения по страницам и/или фильтрации результатов поиска для уменьшения количества загружаемых строк	542
14.4.3	Понимание того, что отложенная загрузка влияет на производительность базы данных.....	543
14.4.4	Добавление метода <i>AsNoTracking</i> к запросам с доступом только на чтение	543
14.4.5	Использование асинхронной версии команд <i>EF Core</i> для улучшения масштабируемости	544
14.4.6	Поддержание кода доступа к базе данных изолированным/слабосвязанным.....	544
14.5	Антипаттерны производительности: запросы к базе данных	545
14.5.1	Антипаттерн: отсутствие минимизации количества обращений к базе данных	545
14.5.2	Антипаттерн: отсутствие индексов для свойства, по которому вы хотите выполнить поиск.....	547
14.5.3	Антипаттерн: использование не самого быстрого способа загрузки отдельной сущности	547
14.5.4	Антипаттерн: перенос слишком большой части запроса данных на сторону приложения	548
14.5.5	Антипаттерн: вычисления вне базы данных	549
14.5.6	Антипаттерн: использование неоптимального <i>SQL</i> -кода в <i>LINQ</i> -запросе	550
14.5.7	Антипаттерн: отсутствие предварительной компиляции часто используемых запросов	550
14.6	Антипаттерны производительности: операции записи.....	552
14.6.1	Антипаттерн: неоднократный вызов метода <i>SaveChanges</i>	552
14.6.2	Антипаттерн: слишком большая нагрузка на метод <i>DetectChanges</i>	553
14.6.3	Антипаттерн: <i>HashSet<T></i> не используется для навигационных свойств коллекции.....	554
14.6.4	Антипаттерн: использование метода <i>Update</i> , когда нужно изменить только часть сущности.....	555
14.6.5	Антипаттерн: проблема при запуске – использование одного большого <i>DbContext</i>	555
14.7	Паттерны производительности: масштабируемость доступа к базе данных	556
14.7.1	Использование пулов для снижения затрат на создание нового <i>DbContext</i> приложения	557
14.7.2	Добавление масштабируемости с незначительным влиянием на общую скорость	557
14.7.3	Повышение масштабируемости базы данных за счет упрощения запросов.....	558
14.7.4	Вертикальное масштабирование сервера базы данных	558
14.7.5	Выбор правильной архитектуры для приложений, которым требуется высокая масштабируемость	559
Резюме	559

15 Мастер-класс по настройке производительности запросов к базе данных..... 561

15.1	Настройка тестового окружения и краткое изложение четырех подходов к повышению производительности	562
------	---	-----

15.2	Хороший LINQ: использование выборочного запроса	565
15.3	LINQ + пользовательские функции: добавляем SQL в код LINQ	568
15.4	SQL + Dapper: написание собственного SQL-кода	570
15.5	LINQ + кеширование: предварительное вычисление частей запроса, которое занимает много времени	573
15.5.1	Добавляем способ обнаружения изменений, влияющих на кешированные значения	574
15.5.2	Добавление кода для обновления кешированных значений	577
15.5.3	Добавление свойств в сущность Book с обработкой параллельного доступа	581
15.5.4	Добавление системы проверки и восстановления в систему событий ...	587
15.6	Сравнение четырех подходов к производительности с усилиями по разработке	589
15.7	Повышение масштабируемости базы данных	591
	Резюме	593

16 *Cosmos DB, CQRS и другие типы баз данных*..... 595

16.1	Различия между реляционными и нереляционными базами данных	596
16.2	Cosmos DB и ее провайдер для EF Core	597
16.3	Создание системы CQRS с использованием Cosmos DB	598
16.4	Проектирование приложения с архитектурой CQRS с двумя базами данных	601
16.4.1	Создание события, вызываемого при изменении сущности Book.....	602
16.4.2	Добавление событий в метод сущности Book	603
16.4.3	Использование библиотеки EfCore.GenericEventRunner для переопределения BookDbContext	605
16.4.4	Создание классов сущностей Cosmos и DbContext	605
16.4.5	Создание обработчиков событий Cosmos	607
16.5	Структура и данные учетной записи Cosmos DB	610
16.5.1	Структура Cosmos DB с точки зрения EF Core	610
16.5.2	Как CosmosClass хранится в Cosmos DB	611
16.6	Отображение книг через Cosmos DB	613
16.6.1	Отличия Cosmos DB от реляционных баз данных	614
16.6.2	Основное различие между Cosmos DB и EF Core: миграция базы данных Cosmos	617
16.6.3	Ограничения поставщика базы данных EF Core 5 для Cosmos DB	618
16.7	Стоило ли использование Cosmos DB затраченных усилий? Да!	621
16.7.1	Оценка производительности системы CQRS с двумя базами данных в приложении Book App	622
16.7.2	Исправление функций, с которыми поставщик баз данных EF Core 5 для Cosmos DB не справился	626
16.7.3	Насколько сложно было бы использовать эту систему CQRS с двумя базами данных в своем приложении?	629
16.8	Отличия в других типах баз данных	630
	Резюме	632

17 *Модульное тестирование приложений, использующих EF Core*..... 634

17.1	Знакомство с настройкой модульного теста	637
17.1.1	Окружение тестирования: библиотека модульного тестирования xUnit	638
17.1.2	Созданная мной библиотека для модульного тестирования приложений, использующих EF Core	639

17.2	Подготовка DbContext приложения к модульному тестированию	640
17.2.1	<i>Параметры DbContext приложения передаются в конструктор</i>	640
17.2.2	<i>Настройка параметров DbContext приложения через OnConfiguring ...</i>	641
17.3	Три способа смоделировать базу данных при тестировании приложений EF Core	643
17.4	Выбор между базой данных того же типа, что и рабочая, и базой данных SQLite in-memory	644
17.5	Использование базы данных промышленного типа в модульных тестах	647
17.5.1	<i>Настройка строки подключения к базе данных, которая будет использоваться для модульного теста</i>	647
17.5.2	<i>Создание базы данных для каждого тестового класса для параллельного запуска тестов в xUnit</i>	649
17.5.3	<i>Убеждаемся, что схема базы данных актуальна, а база данных пуста</i>	651
17.5.4	<i>Имитация настройки базы данных, которую обеспечит миграция EF Core</i>	655
17.6	Использование базы данных SQLite in-memory для модульного тестирования	656
17.7	Создание заглушки или имитации базы данных EF Core	659
17.8	Модульное тестирование базы данных Cosmos DB	662
17.9	Заполнение базы данных тестовыми данными для правильного тестирования кода	664
17.10	Решение проблемы, когда один запрос к базе данных нарушает другой этап теста	665
17.10.1	<i>Код теста с методом <code>ChangeTracker.Clear</code> в отключенном состоянии</i>	667
17.10.2	<i>Код теста с несколькими экземплярами DbContext в отключенном состоянии</i>	668
17.11	Перехват команд, отправляемых в базу данных	669
17.11.1	<i>Использование расширения параметра <code>LogTo</code> для фильтрации и перехвата сообщений журналов EF Core</i>	669
17.11.2	<i>Использование метода <code>ToQueryString</code> для отображения сгенерированного SQL-кода из LINQ-запроса</i>	672
	Резюме	673
	Приложение А. Краткое введение в LINQ	675
	Предметный указатель	686