Оглавление

Часть I ■ НАЧАЛО 34
1 ■ Введение в Entity Framework Core 36
2 ■ Выполнение запроса к базе данных 63
3 ■ Изменение содержимого базы данных 102
4 ■ Использование EF Core в бизнес-логике 139
5 ■ Использование EF Core в веб-приложениях ASP.NET Core 175
6 ■ Советы и техники, касающиеся чтения и записи данных с EF Core 215

Часть II ■ ОБ ENTITY FRAMEWORK В ДЕТАЛЯХ 250
7 ■ Настройка нереляционных свойств 252
8 ■ Конфигурирование связей 291
9 ■ Управление миграциями базы данных 339
10 ■ Настройка расширенных функций и разрешение конфликтов параллельного доступа 382
11 ■ Углубляемся в DbContext 420

Часть III ■ ИСПОЛЬЗОВАНИЕ ENTITY FRAMEWORK CORE В РЕАЛЬНЫХ ПРИЛОЖЕНИЯХ 462
12 ■ Использование событий сущности для решения проблем бизнес-логики 464
13 ■ Предметно-ориентированное проектирование и другие архитектурные подходы 492
14 ■ Настройка производительности в EF Core 530
15 ■ Мастер-класс по настройке производительности запросов к базе данных 561
16 ■ Cosmos DB, CQRS и другие типы баз данных 595
17 ■ Модульное тестирование приложений, использующих EF Core 634

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 MyFirstEfCoreApp 45
1.8 Настройка приложения MyFirstEfCoreApp 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
2.1.2 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 add migration ... 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 DbContext в одной базе данных ... 350
9.5 Редактирование миграции для обработки сложных ситуаций ... 353
9.5.1 Добавление и удаление методов MigrationBuilder внутри класса миграции ... 354
9.5.2 Добавление команд SQL 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 SQL для создания миграций ... 360
9.6.1 Использование инструментов сравнения баз данных SQL для выполнения миграции ... 361
9.6.2 Написание кода сценариев изменения SQL для миграции базы данных вручную ... 363
9.6.3 Проверка соответствия сценариев изменения SQL модели базы данных EF Core ... 365
9.7 Использование инструмента обратного проектирования EF Core ... 366
9.7.1 Запуск команды обратного проектирования ... 367
9.7.2 Установка и запуск команды обратного проектирования Power Tools ... 368
9.7.3 Обновление классов сущности и DbContext при изменении базы данных ... 368
9.8 Часть 2: применение миграций к базе данных ... 369
9.8.1 Вызов метода Database.Migrate из основного приложения ... 370
9.8.2 Выполнение метода Database.Migrate из отдельного приложения ... 373
9.8.3 Применение миграции EF Core с помощью SQL-сценария ... 373
9.8.4 Применение сценариев изменения SQL 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* ... 382
10.1 DbFunction: использование пользовательских функций с EF Core ... 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 HasDefaultValue для добавления постоянного значения для столбца ... 394
10.3.2 Использование метода HasDefaultValueSql для добавления команды SQL для столбца ... 395
10.3.3 Использование метода HasValueGenerator 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 DbUpdateConcurrencyException ... 411
10.6.4 Проблема с отключенным параллельным обновлением ... 415
Резюме ... 419

11 *Углубляемся в DbContext* ... 420
11.1 Обзор свойств класса DbContext ... 421
11.2 Как EF Core отслеживает изменения ... 421
11.3 Обзор команд, которые изменяют свойство сущности State ... 423
11.3.1 Команда Add: вставка новой строки в базу данных ... 424
11.3.2 Метод Remove: 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 Update ... 426
11.3.5 Метод Attach: начать отслеживание существующего неотслеживаемого класса сущности ... 428
11.3.6 Установка свойства сущности State напрямую ... 428
11.3.7 TrackGraph: обработка отключенных обновлений со связями ... 429
11.4 Метод SaveChanges и как он использует метод ChangeTracker. DetectChanges ... 431
11.4.1 Как метод SaveChanges находит все изменения состояния ... 432
11.4.2 Что делать, если метод ChangeTracker.DetectChanges занимает слишком много времени ... 432
11.4.3 Использование состояния сущностей в методе SaveChanges ... 437
11.4.4 Перехват изменений свойства State с использованием события ... 441
11.4.5 Запуск событий при вызове методов SaveChanges и SaveChangesAsync ... 444
11.4.6 Перехватчики EF Core ... 445
11.5 Использование команд SQL в приложении EF Core ... 445
11.5.1 Методы FromSqlRaw/FromSqlInterpolated: использование SQL в запросе EF Core ... 447
11.5.2 Методы ExecuteSqlRaw и ExecuteSqlInterpolated: выполнение команды без получения результата ... 448
11.5.3 Метод Fluent API ToSqlQuery: отображение классов сущностей в запросы ... 448
11.5.4 Метод Reload: используется после команд ExecuteSql ... 450
11.5.5 GetDbConnection: выполнение собственных команд SQL ... 450
11.6 Доступ к информации о классах сущностей и таблицам базы данных ... 452
11.6.1 Использование context.Entry(entity).Metadata для сброса первичных ключей ... 452
11.6.2 Использование свойства context.Model для получения информации о базе данных ... 455
11.7 Динамическое изменение строки подключения DbContext ... 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: сработали 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* ... 530
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 определить, что 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 Select 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 AsNoTracking к запросам с доступом только на чтение 543
14.4.5 Использование асинхронной версии команд EF Core 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 SQL-кода в LINQ-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 SaveChanges 552
14.6.2 Антипаттерн: слишком большая нагрузка на метод DetectChanges 553
14.6.3 Антипаттерн: HashSet<T> не используется для навигационных свойств коллекции 554
14.6.4 Антипаттерн: использование метода Update, когда нужно изменить только часть сущности 555
14.6.5 Антипаттерн: проблема при запуске – использование одного большого DbContext 555
14.7 Паттерны производительности: масштабируемость доступа к базе данных 556
14.7.1 Использование пулов для снижения затрат на создание нового DbContext 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 Параметры DbContext приложения передаются в конструктор 640
17.2.2 Настройка параметров DbContext приложения через OnConfiguring ... 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 Настройка строки подключения к базе данных, которая будет использоваться для модульного теста 647
17.5.2 Создание базы данных для каждого тестового класса для параллельного запуска тестов в xUnit 649
17.5.3 Убеждаемся, что схема базы данных актуальна, а база данных пуста 651
17.5.4 Имитация настройки базы данных, которую обеспечит миграция EF Core 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 Код теста с методом ChangeTracker.Clear в отключенном состоянии 667
17.10.2 Код теста с несколькими экземплярами DbContext в отключенном состоянии 668
17.11 Перехват команд, отправляемых в базу данных 669
17.11.1 Использование расширения параметра LogTo для фильтрации и перехвата сообщений журналов EF Core 669
17.11.2 Использование метода ToQueryString для отображения сгенерированного SQL-кода из LINQ-запроса 672
Резюме 673

Приложение А. Краткое введение в LINQ 675
Предметный указатель 686

DOT
NET
.RU

Вступительное слово от сообщества

В современном мире разработки программного обеспечения сложно обойтись без работы с массивами данных. Как следствие актуальным является вопрос использования различных хранилищ данных. Исторически реляционные базы данных имели широкое применение, и, конечно, платформа .NET не могла не предоставлять свои инструменты для работы с ними.

Перед вами подробное руководство по одному из таких инструментов – Entity Framework Core. EF Core стал почти стандартом при разработке .NET-приложений, использующих реляционные базы данных, и большое число разработчиков успешно применяют его в своих проектах. EF Core позволяет легко начать работу и быстро реализовать простые сценарии по взаимодействию с базами данных. В первой части книги описываются основы фреймворка и вся необходимая информация, чтобы начать работать. Вместе с тем EF Core обеспечивает поддержку и более сложных сценариев. Во второй части более подробно раскрываются внутренние механизмы фреймворка и приводятся сведения о тонкой настройке EF Core, а в третьей части рассматриваются задачи, возникающие при использовании EF Core в реальных приложениях.

Книга будет интересна как новичкам, так и более опытным разработчикам, уже знакомым с основами EF Core. Автор подробно описал, как использовать EF Core, а также затронул большое количество смежных тем, которые помогут понять место фреймворка в экосистеме разработки на платформе .NET. Если у вас уже есть опыт работы с предыдущей версией Entity Framework 6, то вы найдете большое количество сравнений и описание отличий в поведении фреймворков. А если вам интересны нереляционные базы данных, то на примере Cosmos DB вы сможете узнать, как EF Core позволяет работать с такими хранилищами (включая разработку приложения с использованием CQRS-подхода).

Отдельная благодарность автору за «прикладные» главы книги. EF Core (как и многие другие ORM) прост в использовании, но применение его в сложных сценариях может повлечь за собой проблемы производительности. Автор посвятил этой проблеме отдельную главу, подробно разобрал основные проблемы производительности, методы их диагностики и решения. Также в книге затронуты вопросы проектирования (с использованием популярного подхода предметно-ориентированного проектирования, DDD) и тестирования приложений.

В итоге получилась всесторонняя книга о EF Core (и не только), которую можно смело рекомендовать всем, кто работает с базами данных на платформе .NET. Команда DotNet.Ru с удовольствием работала над переводом книги и благодарит автора за отличный материал. Приятного чтения!

Над переводом работали представители сообщества DotNet.Ru:

Игорь Лабутин;	Сергей Бензенко;	Дмитрий Жабин;	Радмир Тагиров;
Рустам Сафин;	Илья Лазарев;	Вадим Мингажев;	Алексей Ростов;
Евгений Буторин;	Андрей Беленцов;	Виталий Илюхин;	Анатолий Кулаков.

Отзывы на книгу «Entity Framework Core в действии»

Наиболее полный справочник по EF Core, который есть или когда-либо будет.

– *Стивен Бирн*, Intel Corporation

Полное руководство по EF Core. Это самый практичный способ улучшить свои навыки по работе с EF Core с примерами из реальной жизни.

– *Пол Браун*, Diversified Services Network

Я твердо верю, что любой, кто использует EF Core, найдет в этой книге что-то полезное для себя.

– *Энн Эпштейн*, Headspring

Остается для меня полезным ресурсом при работе с Entity Framework.

– *Фостер Хейнс*, J2 Interactive

Предисловие

Приходилось ли вам когда-нибудь работать над приложением, которое не использует данные и требует средств взаимодействия с хранилищем данных? За несколько десятилетий работы в качестве разработчика программного обеспечения каждое приложение, над которым я работала или помогала в работе другим, зависело от чтения и записи данных в хранилище определенного типа. Когда я стала индивидуальным предпринимателем в 1990-х г., то придумала для своей компании название Data Farm. Я определенно фанат данных.

За последние несколько десятилетий корпорация Microsoft прошла множество итераций фреймворков для доступа к хранящимся в базе данным. Если вы какое-то время работали в этой сфере, то, возможно, помните DAO и RDO, ADO и ADO.NET. В 2006 году Microsoft поделилась первыми версиями тогда еще не названного Entity Framework (EF) на основе работы, проделанной в Microsoft Research на закрытой встрече в TechEd. Я была одной из немногих, кого пригласили на эту встречу. Я впервые увидела инструмент объектно-реляционного отображения (Object Relational Mapper – ORM), библиотеку, цель которой – освободить разработчиков от излишней рутинной работы по созданию подключений и команд путем написания SQL-запросов, преобразования результатов запроса в объекты и преобразования изменений объекта в SQL, чтобы сохранить их в базе данных.

Многие из нас беспокоились, что это очередной фреймворк для доступа к хранящимся в базе данным, от которого Microsoft откажется в ближайшее время, заставив нас изучать еще один в будущем. Но история доказала, что мы ошибались. Пятнадцать лет спустя Microsoft по-прежнему инвестирует в Entity Framework, который превратился в кросс-платформенный Entity Framework Core с открытым исходным кодом и продолжает оставаться основной библиотекой Microsoft для доступа к данным для разработчиков .NET.

За 15 лет существования и развития EF эволюционировал и .NET. Возможности EF и EF Core стали более обширными, но в то же время, когда дело доходит до создания современных программных систем, эта библиотека стала умнее и понимает, когда нужно просто не мешать разра-

ботчику. Мы можем настраивать отображения для поддержки хранения со сложной схемой базы данных. Как специалист по предметно-ориентированному проектированию, я была очень довольна тем вниманием, которое команда уделила тому, чтобы позволить EF Core сохранять тщательно спроектированные сущности, объекты значений и агрегаты, которые от природы не наделены знанием о схеме базы данных.

Будучи одним из первых пользователей, в тесном сотрудничестве с командой EF еще до первого выпуска этой библиотеки я написала четыре книги по Entity Framework в период с 2008 по 2011 г. Хотя мне и в самом деле нравится писать, в конце концов я обнаружила, что мне также нравится создавать видео, поэтому я сосредоточила свои усилия на создании и публикации курсов по EF Core и другим темам в качестве автора на сайте Pluralsight. Я по-прежнему пишу статьи, но больше не пишу книг, поэтому очень счастлива, что Джон П. Смит нашел способ сотрудничества с издательством Manning и написал эту книгу.

Когда Джон опубликовал первое издание «*Entity Framework Core в действии*», я узнала в нем родственную душу, «любопытного кота», который приложил все возможные усилия в своем стремлении понять, как работает EF Core. Точно так же серьезно он относится к изложению этой информации, гарантируя, что его читатели не потеряют нить повествования и получат реальные знания. Поскольку я продолжала создавать учебные ресурсы для тех, кто предпочитает учиться по видео, мне было приятно порекомендовать работу Джона тем, кто ищет заслуживающее доверия издание по EF Core. Обновить содержимое книги, чтобы привести ее в соответствие с новейшей версией EF Core 5, – нелегкая задача. Джон снова заработал мое уважение (и уважение многих других людей), когда на свет появилось издание, которое вы сейчас держите в руках.

Благодаря этой книге вы получаете три книги в одной. Во-первых, Джон подскажет вам основы и даже создаст несколько простых приложений, использующих EF Core. Когда вы освоитесь, можно будет подробнее изучить использование EF Core на среднем уровне, применяя связи, миграции и управление, выходящее за рамки стандартного поведения EF Core. Наконец, придет время использовать EF Core в реальных приложениях, решая такие важные задачи, как производительность и архитектура. Тщательные исследования Джона и его собственный опыт работы с крупными программными приложениями делают его квалифицированным и заслуживающим доверия гидом.

— ДЖУЛИ ЛЕРМАН

Джули Лерман известна как ведущий эксперт по Entity Framework и EF Core за пределами Microsoft. Она является автором серии книг Programming Entity Framework и десятков курсов на сайте Pluralsight.com. Джули обучает компании, как проводить модернизацию программного обеспечения. Ее можно встретить на конференциях, посвященных программному обеспечению в разных уголках света, где она выступает с докладами по EF, предметно-ориентированному программированию и другим темам.

Введение

Любой разработчик программного обеспечения должен привыкать к необходимости изучения новых библиотек или языков, но для меня это обучение было немного экстремальным. Я перестал заниматься программированием в 1988 г., когда перешел в технический менеджмент, и не возвращался к нему до 2009 г. – перерыв в 21 год. Сказать, что ландшафт изменился, – не сказать ничего; я чувствовал себя ребенком в рождественское утро с таким количеством прекрасных подарков, что не мог взять их все.

Поначалу я совершал все ошибки, присущие новичку, например я думал, что объектно-ориентированное программирование – это использование наследования, однако это не так. Но я изучил новый синтаксис и новые инструменты (вау!) и наслаждался объемом информации, который мог получить в интернете. Я решил сосредоточиться на стеке Microsoft, в основном по причине того, что по нему было доступно большое количество документации. В то время это был хороший выбор, но с выходом .NET Core с открытым исходным кодом и многоплатформенным подходом я понял, что это был отличный выбор.

Первые приложения, над которыми я работал в 2009 г., оптимизировали и отображали потребности здравоохранения с географической точки зрения, особенно с точки зрения расположения лечебных центров. Эта задача требовала сложной математики (этим занималась моя жена) и серьезной работы с базами данных. Я прошел через ADO.NET и LINQ to SQL. В 2013 г. я переключился на Entity Framework (EF), когда EF 5 поддержал пространственные (географические) типы SQL, а затем перешел на EF Core сразу после его выпуска.

За прошедшие годы я часто использовал EF Core и в клиентских проектах, и для создания библиотек с открытым исходным кодом. Помимо этой книги, я много писал о EF Core в собственном блоге (www.thereformedprogrammer.net). Оказывается, мне нравится брать сложные идеи и пытаться сделать так, чтобы их легче было понять другим. Надеюсь, мне удастся сделать это и в данной книге.

«*Entity Framework Core в действии*» охватывает все функции EF Core 5.0 со множеством примеров и кодом, который вы можете за-

пустить. Кроме того, я включил сюда много паттернов и практик, которые помогут вам создать надежный и поддающийся рефакторингу код. В третьей части книги, которая называется «Использование Entity Framework Core в реальных приложениях», показаны создание и доставка реальных приложений. И у меня есть не одна, а три главы о настройке производительности EF Core, поэтому у вас под рукой множество методов повышения производительности, когда ваше приложение работает не так хорошо, как вам нужно.

Одними из самых приятных для написания глав были главы, посвященные тому, как EF Core работает внутри (главы 1, 6 и 11), и настройке производительности приложения (главы 14, 15 и 16). Лично я многому научился, используя модульную монолитную архитектуру (глава 13) и создавая полноценное приложение с помощью Cosmos DB (глава 16). Попутно я стараюсь представить плюсы и минусы каждого используемого мной подхода, поскольку не верю, что в программном обеспечении есть такое понятие, как «серебряная пуля». Есть лишь ряд компромиссов, которые мы, будучи разработчиками, должны учитывать при выборе того, как реализовать что-либо.

Благодарности

Хотя бóльшую часть работы над книгой проделал я, мне очень помогли и другие люди, и я хочу поблагодарить их всех.

Спасибо моей жене, доктору Хоноре Смит, за то, что она терпела, как я три четверти года сидел перед компьютером, и за то, что вернула меня к программированию. Я люблю ее до потери сознания. Еще одна особая благодарность моему большому другу JC за помощь и поддержку.

Работать с Manning Publications было восхитительно. Это был надежный и всеобъемлющий процесс, трудоемкий, но продуманный, обеспечивающий в итоге отличный продукт. Команда была просто замечательная, и я хочу перечислить значимых лиц в хронологическом порядке, начиная с Брайана Сойера, Брекина Эли, Марины Майклс, Джоэля Котарски, Рейханы Марканович, Йосипа Мараса, Хизер Такер, Александара Драгосавлевича и многих других, кто помогал в выпуске книги. Марина Майклс была моим основным контактным лицом во время первого издания, и, очевидно, я не доставил ей слишком много проблем, поскольку она любезно согласилась мне помочь со вторым изданием.

Кроме того, мне очень помогла загруженная команда EF Core. Помимо ответов на многочисленные вопросы, которые были подняты в репозитории EF Core на GitHub, они проверили несколько глав. Особого упоминания заслуживают Артур Викерс и Шай Роянски за рецензирование некоторых глав. Остальные члены команды перечислены в алфавитном порядке: Андрей Свирид, Брайс Лэмбсон, Джереми Ликнесс, Маурициий Марковски и Смит Пател.

Я также хотел бы поблагодарить Жюльена Похи, технического корректора, и рецензентов: Эла Пезевски, Анну Эпштейн, Фостера Хейнса, Хари Хальса, Янека Лопеса, Джеффа Ноймана, Джоэля Клермона, Джона Роудса, Мауро Кверчиоли, Пола Г. Брауна, Раушана Джа, Рикардо Переса, Шона Лэма, Стивена Бирна, Сумита К Сингха, Томаса Гета, Томаса Оверби Хансена и Уэйна Мэзер. Ваши предложения помогли сделать эту книгу лучше.

Об этой книге

Книга «*Entity Framework Core в действии*» посвящена быстрому и правильному написанию кода работы с базой данных с помощью EF Core, чтобы в конечном итоге обеспечить высокую производительность. Чтобы помочь с такими аспектами, как «быстро и правильно», я включил сюда большое количество примеров со множеством советов и приемов. Попутно я немного расскажу, как EF Core работает изнутри, потому что эта информация поможет вам, когда что-то работает не так, как, по вашему мнению, должно работать.

У Microsoft неплохая документация, но в ней нет места для подробных примеров. В этой книге я постараюсь дать вам хотя бы один пример по каждой функции, о которой я рассказываю, а в репозитории на GitHub часто можно будет найти модульные тесты (см. раздел «О коде», где есть ссылки), которые тестируют функцию несколькими способами. Иногда чтение модульного теста может показать, что происходит, гораздо быстрее, нежели чтение текста в книге, поэтому считайте модульные тесты полезным ресурсом.

Кому адресована эта книга?

Эта книга предназначена как для разработчиков программного обеспечения, которые никогда раньше не использовали EF, так и для опытных разработчиков EF Core, а также для всех, кто хочет знать, на что способен EF Core. Я предполагаю, что вы знакомы с разработкой в .NET на C# и имеете хоть какое-то представление о том, что такое реляционная база данных. Не нужно быть экспертом в C#, но если вы новичок, возможно, вам будет трудно читать некоторые части кода, поскольку я не объясняю C#. Книга начинается с основных команд EF Core, которые должны быть доступны большинству программистов на C#, но начиная со второй части темы становятся более сложными по мере углубления в функции EF Core.

Как устроена эта книга

Я попытался построить маршрут, который начинается с основ (часть I), после чего мы переходим к деталям (часть II), а заканчивается он полезными инструментами и методами (часть III). Я не предполагаю, что вы прочитаете эту книгу от корки до корки, особенно справочный раздел из второй части, но хотя бы беглое чтение первых шести глав поможет вам понять основы, которые я использую позже.

Часть I «Начало»:

- глава 1 знакомит вас с суперпростым консольным приложением, использующим EF Core, чтобы вы могли увидеть все части EF Core в действии. Кроме того, я привожу обзор того, как работает EF Core и для чего можно его использовать;
- в главе 2 рассматривается запрос (чтение данных) к базе данных. Я расскажу о связях между данными, хранящимися в базе, и о том, как загрузить эти связанные данные с помощью EF Core;
- в главе 3 мы переходим к изменению данных в базе: добавлению новых данных, обновлению существующих данных и их удалению;
- в главе 4 рассматриваются различные способы построения надежной бизнес-логики, использующей EF Core для доступа к базе данных. *Бизнес-логикой* называется код, реализующий бизнес-правила или рабочий процесс для конкретной бизнес-задачи, которую решает ваше приложение;
- глава 5 посвящена созданию приложения ASP.NET Core, использующего EF Core. Она объединяет код, разработанный в главах 2, 3 и 4, для создания веб-приложения. Помимо этого, я рассказываю о развертывании веб-приложения и доступе к размещенной базе данных;
- глава 6 охватывает широкий круг тем. Большинство из них содержит описание одного из аспектов EF Core в сочетании со способами использования этой функции в коде.

Часть II «Об Entity Framework Core в деталях»:

- в главе 7 рассматривается настройка нереляционных свойств – свойств, содержащих значение, например `int`, `string`, `DateTime` и т. д.;
- в главе 8 рассматривается настройка связей между классами, например классом `Book`, связанным с одним или несколькими классами `Author`. Кроме того, она включает в себя специальные методы отображения, например отображение нескольких классов в одну таблицу;
- в главе 9 описаны все способы изменения структуры базы данных при использовании EF Core, а также рассматриваются проблемы, возникающие, когда вам нужно изменить структуру базы данных, используемой работающим приложением;
- в главе 10 рассматриваются расширенные функции сопоставления и вся область обнаружения и обработки конфликтов параллельного доступа;

- в главе 11 подробно рассмотрено, как работает DbContext EF Core, с подробным описанием того, что различные методы и свойства делают внутри DbContext-приложения.

Часть III «Использование Entity Framework Core в реальных приложениях»:

- в главе 12 представлены два подхода к отправке сообщений расширенным методам `SaveChanges` и `SaveChangesAsync`. Эти подходы предоставляют еще один способ объединения нескольких обновлений в одно транзакционное обновление базы данных;
- в главе 13 рассматривается применение предметно-ориентированного проектирования (DDD) к классам, отображаемым в базу данных с помощью EF Core, а также описывается еще один архитектурный подход, используемый в версии приложения Book App из части III;
- в главе 14 перечислены все проблемы, которые могут повлиять на производительность доступа к базе данных, и обсуждается, что с ними делать;
- глава 15 представляет собой рабочий пример настройки производительности приложения EF Core. Я беру исходный запрос на отображение приложения Book App, разработанного в части I, и применяю три уровня настройки производительности;
- в главе 16 для дальнейшей настройки приложения Book App используется Cosmos DB, что раскрывает сильные и слабые стороны этой базы данных и провайдера EF Core для нее. В конце главы рассказывается, что нужно делать при переходе с одного типа базы данных на другой;
- глава 17 посвящена модульному тестированию приложений, использующих EF Core. Кроме того, я создал пакет NuGet, который вы можете использовать для упрощения своих модульных тестов.

Приложение:

- приложение A знакомит с языком LINQ, используемым в EF Core. Это приложение полезно для тех, кто незнаком с LINQ или хочет быстро вспомнить его.

О коде

Мне кажется, что я действительно что-то знаю только в том случае, если я написал код для использования этой функции или возможности, поэтому вам доступен репозиторий GitHub на странице http://mng.bz/XdlG.

ПРИМЕЧАНИЕ Я настоятельно рекомендую клонировать код с вышеуказанной страницы. У копии репозитория, указанной на странице книг Manning, имеется проблема с веткой `Part3` из-за длинных имен каталогов.

Этот репозиторий содержит код приложений, которые я показываю в книге, и модульные тесты, которые я запускал, чтобы убедиться, что все, о чем я говорю в книге, правильно. У этого репозитория три ветки:

- `master`, охватывающая первую часть книги (главы 1–6);
- `Part2`, охватывающая вторую часть книги (главы 7–11);
- `Part3`, охватывающая третью часть книги (главы 12–17).

Чтобы запустить любое из приложений, сначала необходимо прочитать файл Readme на странице http://mng.bz/yYjG в репозитории GitHub. Файл Readme каждой ветки состоит из трех основных разделов:

- *что нужно установить для запуска примеров приложений*, где указаны приложения для разработки, версия .NET и требования к базе данных для запуска приложений из репозитория GitHub (эта информация одинакова для всех веток);
- *что можно запустить в этой ветке*, где указано, какое приложение (приложения) можно запустить в выбранной вами ветке репозитория GitHub;
- *как найти и запустить модульные тесты*, где говорится, где находятся модульные тесты и как их запустить.

По мере прохождения трех частей книги вы можете выбирать каждую ветку, чтобы получить доступ к коду, предназначенному именно для этой части. Также обратите внимание на связанные модульные тесты, сгруппированные по главам и функциям.

ПРИМЕЧАНИЕ В главе 17, посвященной модульному тестированию, я использовал созданную мной библиотеку. Эта библиотека, которую можно найти на странице https://github.com/JonPSmith/EfCore.TestSupport, – обновленная версия созданной мной библиотеки EfCore.TestSupport для первого издания данной книги. Теперь здесь используются новые функции, доступные в EF Core 5. Это библиотека с открытым исходным кодом (лицензия MIT), поэтому вы можете использовать пакет NuGet под названием EfCore.TestSupport (версия 5 и новее) в собственных модульных тестах.

Соглашения об оформлении программного кода

Примеры кода в этой книге и их вывод написаны моноширинным шрифтом и часто сопровождаются аннотациями. Примеры намеренно делаются максимально простыми, потому что они не представляют собой части для повторного использования, которые можно вставить в ваш код. Образцы кода урезаны, чтобы вы могли сосредоточиться на проиллюстрированном принципе.

Эта книга содержит множество примеров исходного кода, как в пронумерованных листингах, так и в самом тексте. В обоих случаях исходный код отформатирован с использованием моноширинного шрифта,

`как этот`, чтобы отделить его от остального текста. Кроме того, иногда используется **`жирный шрифт`**, чтобы выделить код, который изменился по сравнению с предыдущими шагами в главе, например когда в существующую строку кода добавляется новая функция.

Во многих случаях исходный код был переформатирован; мы добавили разделители строк и переделали отступы, чтобы уместить их по ширине книжных страниц. Также из многих листингов, описываемых в тексте, мы убрали комментарии. Некоторые листинги сопровождают аннотации, выделяющие важные понятия.

Исходный код примеров из этой книги доступен для скачивания из репозитория GitHub (http://mng.bz/XdlG).

Автор онлайн

Приобретая книгу «*EF Core в действии*», вы получаете бесплатный доступ на приватный веб-форум издательства Manning Publications, где можно оставить отзывы о книге, задать технические вопросы и получить помощь от авторов и других пользователей. Чтобы получить доступ к форуму, откройте в браузере страницу https://livebook.manning.com/book/entity-framework-core-in-action-second-edition. На странице https://livebook.manning.com/#!/discussion можно получить дополнительную информацию о форумах Manning и правилах поведения на них.

Издательство Manning обязуется предоставить своим читателям место встречи, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Но со стороны автора отсутствуют какие-либо обязательства уделять форуму какое-то определенное внимание – его присутствие на форуме остается добровольным (и неоплачиваемым). Мы предлагаем задавать автору стимулирующие вопросы, чтобы его интерес не угас!

Форум и архивы предыдущих дискуссий будут оставаться доступными, пока книга продолжает издаваться.

Онлайн-ресурсы

Полезные ссылки на документацию Microsoft и код:

- *документация Microsoft по EF Core* – https://docs.microsoft.com/en-us/ef/core/;
- *код EF Core* – https://github.com/dotnet/efcore;
- *ASP.NET Core, работа с EF Core* – https://docs.microsoft.com/en-us/aspnet/core/data/;
- *тег EF Core на Stack Overflow* `[entity-framework-core]` – https://stackoverflow.com.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понра-

вилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Джон П. Смит – внештатный разработчик программного обеспечения и программный архитектор, специализирующийся на .NET Core и Azure. Он работает в основном над серверной частью клиентских приложений, обычно используя Entity Framework Core (EF Core) и веб-приложения ASP.NET Core. Джон работает удаленно с клиентами по всему миру, многие из его проектов из Соединенных Штатов. Обычно он помогает с проектированием, настройкой производительности и написанием разделов клиентского приложения.

Джон интересуется определением паттернов и созданием библиотек, которые повышают скорость разработки приложений при использовании EF Core и ASP.NET Core. Его библиотеки были написаны, потому что он нашел некую повторяющуюся часть проекта, над которым работал, которую можно было превратить в полезную библиотеку. Сводку его основных библиотек можно увидеть на странице в GitHub (https://github.com/JonPSmith).

Кроме того, Джон ведет собственный технический блог на странице http://www.thereformedprogrammer.net, где рассматривает темы, связанные с EF Core, ASP.NET Core и различными архитектурными подходами. Самая популярная статья в его блоге посвящена улучшенной системе авторизации ASP.NET Core; см. http://mng.bz/ao2z. Он выступал на нескольких конференциях и митапах в Великобритании.

Об изображении на обложке

Иллюстрация на обложке книги называется «Жена франкского торговца». Она взята из книги Томаса Джефферисса «Коллекция платьев разных народов, древних и современных» (четыре тома), Лондон, изданной между 1757 и 1772 г. На титульном листе указано, что эти иллюстрации представляют собой гравюры на медной пластине ручной раскраски, усиленные гуммиарабиком.

Томаса Джефферисса (1719–1771) называли «географом короля Георга III». Он был английским картографом и ведущим поставщиком карт своего времени. Гравировал и печатал карты для правительства и других официальных организаций, а также выпустил широкий спектр коммерческих карт и атласов, особенно Северной Америки. Его работа в качестве картографа вызвала у него интерес к местным обычаям одежды на землях, которые он исследовал и наносил на карту и которые с блеском представлены в этой коллекции. Очарование далеких стран и путешествия ради удовольствия были относительно новым явлением в конце XVIII века, и коллекции, подобные этой, были популярны, знакомя и туриста, и путешественника, сидящего в кресле, с жителями других стран.

Разнообразие рисунков в томах Джефферисса наглядно свидетельствует об уникальности и индивидуальности народов мира около 200 лет назад. Манеры одеваться сильно изменились с тех пор, а своеобразие каждого региона, такое яркое в то время, давно поблекло. Сейчас бывает трудно различить обитателей разных континентов. Возможно, если смотреть на это оптимистично, мы обменяли культурное и визуальное разнообразие на более разнообразную частную жизнь или более разнообразную интеллектуальную и техническую жизнь.

В то время когда сложно отличить одну компьютерную книгу от другой, мы в Manning высоко ценим изобретательность, инициативу и, конечно, радость от компьютерного бизнеса, используя книжные обложки, основанные на разнообразии жизни в разных регионах два века назад, которое оживает благодаря картинкам из этой коллекции.

Часть I

Начало

Данные используются повсеместно, и каждый год их объем растет петабайтами. Значительная их часть хранится в базах данных. Кроме того, существуют миллионы приложений – в начале 2021 г. насчитывалось 1,2 млрд сайтов, – и большинству из них требуется доступ к данным. И это не говоря об интернете вещей. Поэтому неудивительно, что согласно прогнозу ведущей исследовательской и консалтинговой компании Gartner в 2021 г. глобальные расходы на IT достигнут 3,7 трлн долларов (http://mng.bz/gonl).

Хорошая новость для вас заключается в том, что ваши навыки будут востребованы. Однако плохая состоит в том, что потребность в быстрой разработке приложений неумолимо растет. Эта книга посвящена инструменту, который можно использовать для быстрого написания кода доступа к базам данных: Microsoft Entity Framework Core (EF Core). EF Core предоставляет объектно-ориентированный способ доступа к реляционным и нереляционным (NoSQL) базам данных в среде .NET. Самое замечательное в EF Core и других библиотеках .NET Core заключается в их быстродействии, а также в том, что они могут работать на платформах Windows, Linux и Apple.

В первой части книги мы с вами сразу же погрузимся в код. В главе 1 вы создадите очень простое консольное приложение, а к концу главы 5 – достаточно сложное веб-приложение для продажи книг. В главах 2 и 3 объясняется чтение и запись данных в реляционную базу данных, а в главе 4 рассказывается о написании бизнес-логики. В главе 5 вы будете использовать веб-фреймворк Microsoft ASP.NET Core для создания сайта по продаже книг. Глава 6 расширит ваши познания о внутренней работе EF Core с помощью ряда полезных ме-

тодов решения проблем с базами данных, таких как быстрый способ копирования данных в базу.

В первой части вам предстоит многое изучить, несмотря на то что я опущу здесь несколько тем, полагаясь на используемые по умолчанию настройки EF Core. Тем не менее эта часть должна дать вам четкое представление о том, что может делать этот фреймворк. Последующие части дополнят ваши познания, рассказывая о дополнительных функциях EF Core, с более подробной информацией о его настройках, а некоторые главы посвящены специфическим областям, таким как настройка производительности.

1 Введение в Entity Framework Core

В этой главе рассматриваются следующие темы:

- анатомия приложения EF Core;
- доступ к базе данных и ее обновление с помощью EF Core;
- изучение реального приложения EF Core;
- принятие решения об использовании EF Core в своем приложении.

Entity Framework Core, или *EF Core* – это библиотека, которую разработчики программного обеспечения могут использовать для доступа к базам данных. Есть много способов создать такую библиотеку, но EF Core разработана как инструмент объектно-реляционного отображения (ORM). Эти инструменты занимаются отображением между двумя мирами: реляционной базой данных (с собственным API) и объектно-ориентированным миром классов кода программного обеспечения. Основное преимущество EF Core – предоставить разработчикам программного обеспечения возможность быстрого написания кода доступа к базе данных на языке, который вы, возможно, знаете лучше, чем SQL.

EF Core поддерживает несколько платформ: он может работать в Windows, Linux и Apple. Это делается в рамках платформы .NET Core – отсюда и слово *Core* в названии. .NET 5 охватывает весь спектр настольных компьютеров, сеть, облачные и мобильные приложения, игры, интернет вещей (IoT) и искусственный интеллект (AI), однако эта книга посвящена EF Core.

EF Core – это логическое продолжение более ранней версии Entity Framework, известной как *EF6.x*. EF Core берет свое начало из многолетнего опыта, накопленного в предыдущих версиях, с 4 по 6.x. Он сохранил тот же тип интерфейса, что и EF6.x, но в нем есть существенные изменения, например возможность работы с нереляционными базами данных, чего не было предусмотрено в EF6.x. Я использовал EF5 и EF6 во многих приложениях до появления EF Core, что позволило мне увидеть значительные улучшения EF Core по сравнению с EF6.x как в отношении функциональных возможностей, так и в отношении производительности.

Эта книга предназначена как для разработчиков программного обеспечения, которые уже используют EF Core, так и для тех, кто никогда ранее не работал с Entity Framework, а также опытных разработчиков на EF6.x, которые хотят перейти на EF Core. Я предполагаю, что вы знакомы с разработкой на .NET с использованием C# и имеете представление о том, что такое реляционные базы данных. Возможно, вы не знаете, как писать запросы с помощью языка структурированных запросов (SQL), который используется большинством реляционных баз данных, потому что EF Core может проделать бóльшую часть этой работы за вас. Тем не менее я буду демонстрировать SQL запросы, которые создает EF Core, потому что это помогает понять, что происходит и как это работает. Для использования некоторых расширенных возможностей EF Core требуются знания SQL, но в книге есть большое количество диаграмм, которые помогут вам разобраться.

> **СОВЕТ** Если вы плохо разбираетесь в SQL и хотите узнать больше, предлагаю посетить онлайн-ресурс W3Schools: https://www.w3schools.com/sql/sql_intro.asp. Набор SQL-команд довольно обширен, но запросы, формируемые EF Core, используют лишь небольшое их подмножество (например, SELECT, WHERE и INNER JOIN), поэтому данный ресурс – хорошее место, чтобы получить базовые знания о SQL.

В этой главе вы познакомитесь с EF Core на примере небольшого приложения, которое работает с данной библиотекой. Заглянете под ее капот, чтобы увидеть, как команды на языке C# преобразуются в SQL запросы к базе данных. Обзорное представление того, что происходит внутри EF Core, поможет вам при чтении остальной части книги.

1.1 Что вы узнаете из этой книги

Книга содержит введение в библиотеку EF Core, начиная с основ и заканчивая более сложными ее аспектами. Чтобы извлечь максимальную пользу из книги, вы должны уметь разрабатывать приложения на C#, в том числе создавать проекты и загружать NuGet пакеты. Вы узнаете:

- основные принципы использования EF Core для работы с базами данных;
- как использовать EF Core в веб-приложении ASP.NET Core;
- многочисленные способы настройки EF Core, чтобы все работало именно так, как вам нужно;
- как использовать некоторые расширенные возможности баз данных;
- как управлять изменениями в схеме базы данных по мере роста приложения;
- как повысить производительность взаимодействия вашего кода с базой данных;
- и самое главное, как убедиться, что ваш код работает правильно.

На протяжении всей книги я буду создавать простые, но полнофункциональные приложения, чтобы вы могли увидеть, как EF Core работает в реальных ситуациях. Все эти приложения, а также множество советов и приемов, которые я собрал, занимаясь как рабочими, так и своими собственными проектами, доступны в репозитории с примерами.

1.2 Мой «момент озарения»

Прежде чем перейти к сути, позвольте рассказать вам об одном решающем моменте, который возник, когда я работал с Entity Framework, и который наставил меня на путь «поклонника» EF. Именно моя жена вернула меня в программирование после 21-летнего перерыва (это уже отдельная история!).

Моя жена, доктор Хонора Смит, преподает математику в Университете Саутгемптона и специализируется на моделировании систем здравоохранения, уделяя особое внимание расположению медицинских учреждений. Я работал с ней над созданием нескольких приложений для географического моделирования и визуализации для Национальной службы здравоохранения Великобритании, а также в Южной Африке над оптимизацией тестирования на ВИЧ/СПИД.

В начале 2013 г. я решил создать веб-приложение специально для моделирования здравоохранения. Я использовал только что появившиеся ASP.NET MVC4 и EF5, которые поддерживали пространственные типы SQL, обрабатывающие географические данные. Проект был успешным, но это была тяжелая работа. Я знал, что интерфейс будет сложным; это было одностраничное приложение, где использовалась библиотека Backbone.js, но я был удивлен тем, сколько времени мне потребовалось, чтобы выполнить работу на стороне сервера.

Я применил передовые методы работы с программным обеспечением и убедился, что база данных и бизнес-логика соответствуют задаче – моделированию и оптимизации расположения медицинских учреждений. Все было нормально, но я потратил слишком много времени на написание кода для преобразования записей базы данных и бизнес-

логики в форму, подходящую для отображения пользователю. Кроме того, я использовал паттерны «Репозиторий» (Repository) и «Единица работы» (Unit of Work), чтобы скрыть код EF5, и мне постоянно приходилось настраивать области, чтобы репозиторий работал правильно.

В конце проекта я всегда оглядываюсь назад и спрашиваю: «Мог бы я сделать это лучше?» Будучи разработчиком программного обеспечения, я всегда ищу части, которые (а) хорошо работали, (б) повторялись и должны быть автоматизированы или (с) имели постоянные проблемы. На этот раз список выглядел так:

- *работает хорошо* – ServiceLayer, слой в моем приложении, который изолировал/адаптировал нижние уровни приложения от внешнего интерфейса ASP.NET MVC4, работал хорошо (я представлю эту многоуровневую архитектуру в главе 2);
- *повторяемость* – я использовал классы ViewModel, также известные как *объекты передачи данных* (DTO), чтобы представить данные, которые мне нужно было показать пользователю. Все прошло хорошо, но написание кода для копирования таблиц базы данных во ViewModel и DTO было однообразным и скучным делом (о ViewModels и DTO рассказывается в главе 2);
- *наличие постоянных проблем* – паттерны «Репозиторий» и «Единица работы» мне не подошли. На протяжении всего проекта возникали постоянные проблемы (я расскажу о паттерне «Репозиторий» и его альтернативах в главе 13).

В результате своего обзора я создал библиотеку GenericServices (https://github.com/JonPSmith/GenericServices) для использования с EF6.x. Она автоматизировала копирование данных между классами базы данных и классами ViewModel и DTO, устраняя необходимость в использовании вышеуказанных паттернов. Библиотека вроде бы работала хорошо, но, чтобы провести для нее стресс-тест, я решил создать интерфейс на основе одного из примеров баз данных Microsoft: AdventureWorks 2012 Lite. Я создал все приложение с помощью библиотеки для разработки пользовательских интерфейсов за 10 дней!

Entity Framework + правильные библиотеки + правильный подход = быстрая разработка кода доступа к базе данных

Сайт был не очень красивым, но дело не во внешнем виде. Анализируя то, как я использовал паттерны «Репозиторий» и «Единица работы» с EF6.x, я нашел лучший подход. Затем, инкапсулируя этот подход в библиотеку GenericServices, я автоматизировал процесс создания команд Create, Read, Update и Delete (CRUD). Результат позволил мне очень быстро создавать приложения – определенно это был «момент озарения», и я «подсел» на EF.

С тех пор я создал новые библиотеки, работающие с EF Core, которые, как я обнаружил, значительно ускоряют разработку 90 % обращений к базе данных. Я работаю разработчиком по контракту, и эти библиотеки с открытым исходным кодом, доступные и вам, автома-

тизируют некоторые стандартные требования, позволяя сосредоточиться на более сложных темах, таких как понимание потребностей клиента, написание пользовательской бизнес-логики и настройка производительности там, где это необходимо. Я расскажу об этих библиотеках в последующих главах.

1.3 Несколько слов для разработчиков EF6.x

> **ЧТОБЫ СЭКОНОМИТЬ ВРЕМЯ** Если вы не работали с Entity Framework 6.x, то можете пропустить этот раздел.

Если вы работали с EF6.x, то многое из EF Core будет вам знакомо. Чтобы помочь вам сориентироваться в этой книге, я добавил примечания по EF6.

> **EF6** Обращайте внимание на эти примечания в ходе чтения. Они указывают на те моменты, где EF Core отличается от EF6.x. Также обязательно просмотрите аннотации в конце каждой главы, которые указывают на наиболее серьезные отличия между EF6 и EF Core.

Кроме того, я дам вам один совет из своего путешествия по EF Core. Я хорошо знаю EF6.x, но эти знания стали проблемой, когда я начал использовать EF Core. Я применял к проблемам подход на базе EF6.x и не заметил, что у EF Core появились новые способы их решения. В большинстве случаев подходы схожи, а где-то нет.

Мой совет вам как существующему разработчику EF6.x – подходите к EF Core как к новой библиотеке, которая была написана для имитации EF6.x, но учтите, что она работает иначе. Таким образом, вы будете готовы к новым способам работы в EF Core.

1.4 Обзор EF Core

Можно использовать EF Core как инструмент объектно-реляционного отображения (далее: ORM), который работает с реляционной базой данных и миром классов и программного кода .NET. Смотрите табл. 1.1.

Таблица 1.1. Как используется EF Core для работы с базой данных в .NET

Реляционная база данных	Программное обеспечение .NET
Таблица	Класс .NET
Столбцы таблицы	Свойства/поля класса
Записи/строки в таблице	Элементы в коллекциях .NET, например `List`
Первичные ключи: уникальная запись/строка	Уникальный экземпляр класса
Внешние ключи: определение связи	Ссылка на другой класс
SQL-оператор, например `WHERE`	Запросы на языке LINQ, например `Where(p => …`

1.4.1 Недостатки инструментов объектно-реляционного отображения

Создать хороший инструмент ORM сложно. Хотя EF6.x или EF Core могут показаться простыми в использовании, иногда «магия» EF Core может застать вас врасплох. Позвольте упомянуть две проблемы, о которых следует знать, прежде чем подробно рассматривать, как работает EF Core.

Первая проблема – это *объектно-реляционное несоответствие*. Серверы баз данных и объектно-ориентированное программное обеспечение используют разные принципы; базы данных используют первичные ключи для определения уникальности строки, тогда как экземпляры классов .NET по умолчанию считаются уникальными по их ссылке. EF Core обрабатывает бóльшую часть несоответствия за вас, но классы .NET получают первичные и внешние ключи, которые являются дополнительными данными, необходимыми только для базы данных. Программная версия классов не нуждается в этих дополнительных свойствах, а базе данных они нужны.

Вторая проблема заключается в том, что инструмент ORM – в особенности такой всеобъемлющий, как EF Core, – является противоположностью первой проблемы. EF Core настолько хорошо «скрывает» базу данных, что иногда можно забыть о том, что находится внутри нее. Эта проблема может привести к написанию кода, который будет хорошо работать в C#, но не подходит для базы данных. Один из примеров – вычисляемое свойство, возвращающее полное имя человека, путем объединения свойств `FirstName` и `LastName` в классе, например

```
public string FullName => $"{FirstName} {LastName}";
```

Это свойство является правильным в C#, но это же свойство вызовет исключение, если вы попытаетесь выполнить фильтрацию или сортировку по нему, потому что EF Core требуется столбец FullName в таблице, чтобы можно было применить SQL-команду `WHERE` или `ORDER` на уровне базы данных.

Вот почему в этой главе я показываю, как EF Core работает изнутри и какой SQL он создает. Чем больше вы понимаете, что делает EF Core, тем лучше вы будете подготовлены для написания правильного кода и, что более важно, будете знать, что делать, если ваш код не работает.

ПРИМЕЧАНИЕ На протяжении всей книги я использую подход «Сделайте так, чтобы это работало, но будьте готовы сделать это быстрее, если нужно». EF Core позволяет быстро осуществлять разработку, но я знаю, что из-за EF Core или из-за того, что я плохо его использовал, производительность моего кода доступа к базе данных может быть недостаточно хорошей для определенной бизнес-потребности. В главе 5 рассказывается, как изолировать EF Core, чтобы можно было выполнить настройку с минимальными побочными эффектами, а в главе 15

показано, как найти и улучшить код базы данных, который работает недостаточно быстро.

1.5 Что насчет нереляционных (NoSQL) баз данных?

Нельзя говорить о реляционных базах данных, не упомянув нереляционные базы данных, также известные в обиходе как NoSQL (http://mng.bz/DW63). И реляционные, и нереляционные базы данных играют важную роль в современных приложениях. Я использовал Microsoft SQL Server (реляционная база данных) и Azure Tables (нереляционная база данных) в одном приложении для удовлетворения двух бизнес-требований.

EF Core работает и с реляционными, и нереляционными базами данных – в отличие от EF6.x, которая была разработана только для реляционных баз данных. Большинство команд EF Core, описанных в этой книге, применимы к обоим типам баз данных, но есть некоторые различия на уровне базы данных, которые не учитывают некоторые более сложные команды для обеспечения масштабируемости и производительности.

В EF Core 3.0 добавлен провайдер базы данных Azure, Cosmos DB, о котором я расскажу в главе 16. В этой главе я указываю на различия между реляционной базой данных и Cosmos DB. Я был удивлен тем, что обнаружил. Теперь, когда в EF Core были внесены изменения, чтобы его можно было использовать для работы с нереляционными базами данных, я ожидаю, что для этого типа БД будут написаны и другие провайдеры.

> **ПРИМЕЧАНИЕ** У Cosmos DB и других нереляционных баз данных есть много сильных сторон по сравнению с реляционными базами данных. Например, гораздо проще иметь несколько копий нереляционных БД по всему миру, что позволяет пользователю быстрее получить доступ, а если центр обработки данных выйдет из строя, то другие копии могут принять на себя нагрузку. Но у нереляционных баз данных также есть некоторые ограничения по сравнению с базами данных SQL; прочтите главу 16, где приводится углубленный анализ преимуществ и ограничений Cosmos DB.

1.6 Ваше первое приложение, использующее EF Core

В этой главе мы начнем с простого примера, чтобы можно было сосредоточиться на том, что делает EF Core, а не код. Мы напишем не-

большое консольное приложение MyFirstEfCoreApp, которое обращается к простой базе данных. Задача приложения – вывести список книг и обновить его в прилагаемой базе данных. На рис. 1.1 показан вывод консоли.

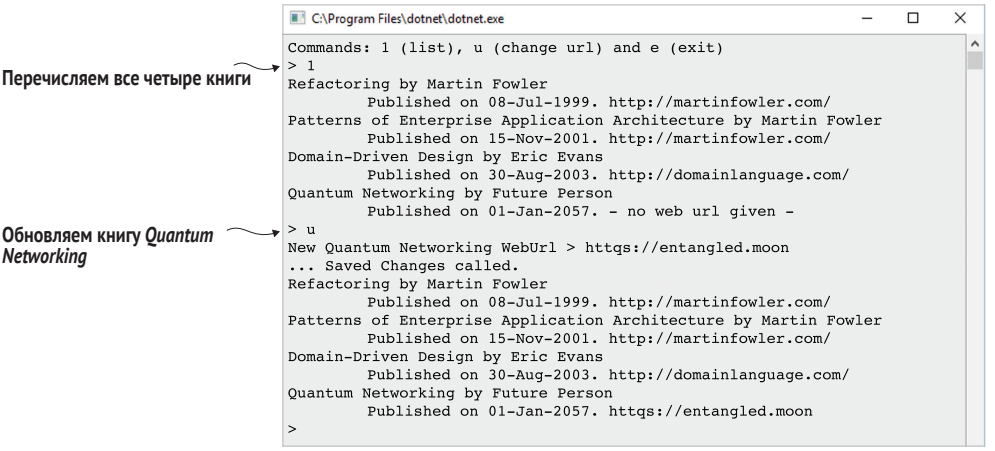


Рис. 1.1 Консольное приложение предоставляет команду, которая использует запрос EF Core для чтения и отображения всех книг из вашей базы данных, а также команду для обновления. Эти две команды показывают, как EF Core работает изнутри

Это приложение не получит никаких призов за свой интерфейс или сложность, но это хороший способ начать работу, особенно потому, что я хочу показать вам, как EF Core работает изнутри, чтобы понять, о чем пойдет речь в последующих главах.

Пример приложения можно скачать из репозитория Git на странице http://mng.bz/XdlG. Вы можете посмотреть код и запустить приложение. Для этого вам понадобятся инструменты разработчика.

1.6.1 Что нужно установить

У Microsoft есть два редактора для работы с приложениями .NET Core: Visual Studio и Visual Studio Code (сокращенно VS Code). Visual Studio немного проще в использовании, и если вы новичок в .NET, то предлагаю использовать его. Его можно скачать на сайте www.visualstudio.com. Существует множество версий, в том числе и бесплатная Community версия, но необходимо прочитать лицензию, чтобы убедиться, что вы соответствуете требованиям: www.visualstudio.com/vs/community.

При установке Visual Studio в Windows обязательно включите функцию кросс-платформенной разработки (Cross-Platform Development) .NET Core и функцию хранения и обработки данных (Data storage and processing), которые находятся в разделе **Other Toolsets** (Другие наборы инструментов) на этапе **Install Workloads** (Установка рабочих нагрузок). Выбрав функцию кросс-платформенной разработки, вы

также установите в своей системе комплект для разработки программного обеспечения .NET Core, который необходим для создания приложений на .NET. Посетите страницу http://mng.bz/2x0T для получения дополнительной информации.

Если вы хотите использовать бесплатный редактор VS Code, то можете скачать его на странице https://code.visualstudio.com. Вам нужно будет выполнить дополнительные настройки в системе, например установить последнюю версию .NET Core SDK и SQL Server Express LocalDB. Как я уже сказал, если вы новичок в .NET, то предлагаю использовать Visual Studio для Windows, так как она может многое настроить за вас.

Одна из версий Visual Studio работает на компьютере с Apple Macintosh, а версии VS Code работают в Windows, на Mac и в Linux. Если вы хотите запустить какое-либо приложение или модульный тест, то на вашем компьютере должен быть установлен SQL Server. Возможно, вам потребуется изменить имя сервера в строках подключения.

Можно запускать модульные тесты с помощью встроенного в Visual Studio обозревателя тестов, доступного из меню **Test**. Если вы используете VS Code, то средство запуска тестов здесь также имеется, но необходимо настроить задачи сборки и тестирования в файле VS Code tasks.json, что позволяет запускать все тесты посредством команды **Task** > **Test**.

1.6.2 Создание собственного консольного приложения .NET Core с помощью EF Core

Я знаю, что многим разработчикам нравится создавать собственные приложения, потому что написание самого кода означает, что вы точно знаете, в чем дело. В этом разделе показано, как создать консольное приложение .NET MyFirstEfCoreApp с помощью Visual Studio.

СОЗДАНИЕ КОНСОЛЬНОГО ПРИЛОЖЕНИЯ .NET CORE

В Visual Studio есть отличный набор руководств, а на странице http://mng.bz/e56z можно найти пример создания консольного приложения C#.

> **СОВЕТ** Чтобы узнать, какую версию .NET использует ваше приложение, выберите **Project** > **MyFirstEfCoreApp Properties** в главном меню; на вкладке **Application** (Приложение) показана целевая платформа. Некоторые версии EF Core требуют определенной версии .NET Core.

ДОБАВЛЯЕМ БИБЛИОТЕКУ EF CORE В ПРИЛОЖЕНИЕ

Установить библиотеку NuGet можно различными способами. Более наглядный – использовать диспетчер пакетов NuGet; руководство

можно найти на странице http://mng.bz/pVeG. Для этого приложения вам понадобится пакет EF Core для той базы данных, к которой приложение будет подключаться. В данном случае выбираем NuGet пакет Microsoft.EntityFrameworkCore.SqlServer, потому что приложение будет использовать SQL Server для разработки, который был установлен при установке Visual Studio.

Также нужно обратить внимание на номер версии пакета NuGet, который вы собираетесь установить. EF Core устроен таким образом, что у каждого основного выпуска есть свой номер. Например, номер версии 5.1.3 означает основную версию (Major) 5, с второстепенным выпуском (Minor) 1 и патчем (исправлением ошибок – Patch) версии 3. Часто в разных проектах нужно загружать разные пакеты EF Core. Например, может потребоваться загрузить Microsoft.EntityFrameworkCore в слой доступа к данным и Microsoft.EntityFrameworkCore.SqlServer в веб-приложение. Если нужно это делать, старайтесь использовать пакеты NuGet с одинаковыми версиями Major.Minor. Patch. Если полного совпадения добиться невозможно, следите, чтобы совпадали хотя бы версии Major.Minor.

Скачивание и запуск примера приложения из репозитория Git

Есть два варианта скачивания и запуска консольного приложения MyFirstEfCoreApp из репозитория Git: Visual Studio и VS Code. Можно найти еще одно руководство по Visual Studio, «Открыть проект из репозитория», на странице http://mng.bz/OE0n. Репозиторий для этой книги находится на странице http://mng.bz/XdlG.

Обязательно выберите правильную ветку. В репозитории Git есть ветки, позволяющие переключаться между разными версиями кода. Для этой книги я создал три основные ветки: master, которая содержит код из первой части (главы 1–6); Part2, содержащую код из второй части (главы 7–11); и Part3, которая содержит код из третьей части (главы 12–17).

По умолчанию репозиторий будет открыт в ветке master, чтобы те, кто не привыкли к Git, могли сразу приступить к работе. Файл Readme в каждой ветке содержит дополнительную информацию о том, что нужно установить и что можно запустить.

1.7 База данных, к которой будет обращаться MyFirstEfCoreApp

EF Core предназначен для доступа к базе данных, но откуда берется эта база? EF Core предоставляет вам два варианта: EF Core может создать ее за вас, используя подход «*Сначала код*» (*code-first*), или вы можете предоставить существующую базу данных, созданную вами за пределами EF Core. Этот подход называется «*Сначала база данных*»

(*database-first*). В первой части книги используется первый вариант, потому что этот подход применяется многими разработчиками.

> **EF6** В EF6 можно использовать EDMX / конструктор базы данных для визуального проектирования своей базы данных. Этот вариант известен как «*Сначала проектирование*» (*design-first*). EF Core не поддерживает данный подход в какой бы то ни было форме, и планов его добавлять нет.

В этой главе мы не будем изучать, как создается база данных. Чтобы приложение MyFirstEfCoreApp работало, код самостоятельно создаст базу данных и добавит тестовые данные, если базы данных нет.

> **ПРИМЕЧАНИЕ** В своем коде я использую базовую команду EF Core, предназначенную для модульного тестирования, чтобы создать базу данных, потому что это просто и быстро. В главе 5 рассказывается, как правильно создать базу данных с EF Core, а в главе 9 полностью описана проблема создания и изменения структуры базы данных, известной как *схема* базы данных.

Для приложения MyFirstEfCoreApp я создал простую базу данных, показанную на рис. 1.2, всего с двумя таблицами:

- таблицей Books с информацией о книгах;
- таблицей Author с авторами.

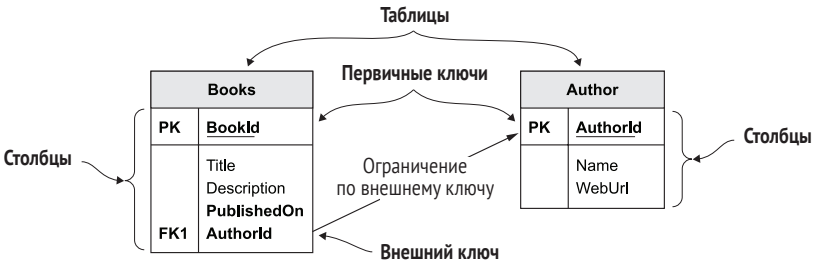
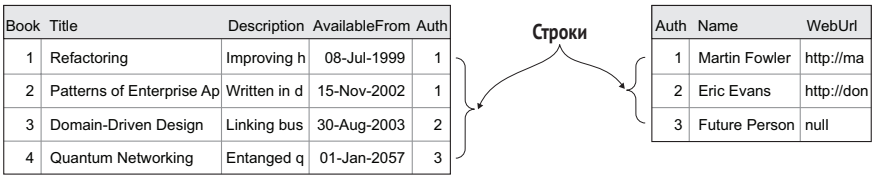


Рис. 1.2 Пример нашей реляционной базы данных с двумя таблицами: Books и Author

> **РАСШИРЕННОЕ ПРИМЕЧАНИЕ** В этом примере я позволил EF Core дать таблицам имена, используя параметры конфигурации по умолчанию. Имя таблицы Books взято из свойства `DbSet<Book> Books`, показанного на рис. 1.5. Для имени таблицы Author на рис. 1.5 нет свойства `DbSet<T>`, поэтому EF Core использует имя класса.

На рис. 1.3 показано содержимое базы данных. В ней всего четыре книги, у первых двух один и тот же автор: Мартин Фаулер.



Book	Title	Description	AvailableFrom	Auth
1	Refactoring	Improving h	08-Jul-1999	1
2	Patterns of Enterprise Ap	Written in d	15-Nov-2002	1
3	Domain-Driven Design	Linking bus	30-Aug-2003	2
4	Quantum Networking	Entanged q	01-Jan-2057	3

Auth	Name	WebUrl
1	Martin Fowler	http://ma
2	Eric Evans	http://don
3	Future Person	null

Рис. 1.3 Содержимое базы данных, где показано четыре книги, у двух из которых один и тот же автор

1.8 Настройка приложения MyFirstEfCoreApp

После создания и настройки консольного приложения .NET можно приступить к написанию кода EF Core. Перед созданием кода доступа к любой базе данных нужно написать две основные части:

- классы, которые EF Core должен отображать в таблицы вашей базы данных;
- DbContext – это основной класс, который вы будете использовать для настройки базы данных и получения доступа к ней.

1.8.1 Классы, которые отображаются в базу данных: Book и Author

EF Core отображает классы в таблицы базы данных. Следовательно, нужно создать класс, который будет определять таблицу базы данных или согласуется с ней, если база данных у вас уже есть. Существует множество правил и конфигураций (они описаны в главах 7 и 8), но на рис. 1.4 показан типичный формат класса, отображаемого в таблицу базы данных.

В листинге 1.1 показан другой класс, который вы будете использовать: `Author`. У него та же структура, что и у класса `Book` на рис. 1.4, с первичным ключом, что соответствует соглашению об именовании `<ClassName>Id` (см. раздел 7.3.5). У класса `Book` также есть навигационное свойство типа `Author` и свойство типа `int` с именем `AuthorId`, соответствующее первичному ключу `Author`. Эти два свойства сообщают EF Core, что вам нужна связь класса `Book` с классом `Author` и что свойство `AuthorId` нужно использовать как внешний ключ для связи двух таблиц в базе данных.

Листинг 1.1 Класс Author из приложения MyFirstEfCoreApp

```
public class Author
{
    public int AuthorId { get; set; }
    public string Name { get; set; }
    public string WebUrl { get; set; }
}
```

Содержит первичный ключ строки Author в БД. Обратите внимание, что у внешнего ключа из класса Book то же имя

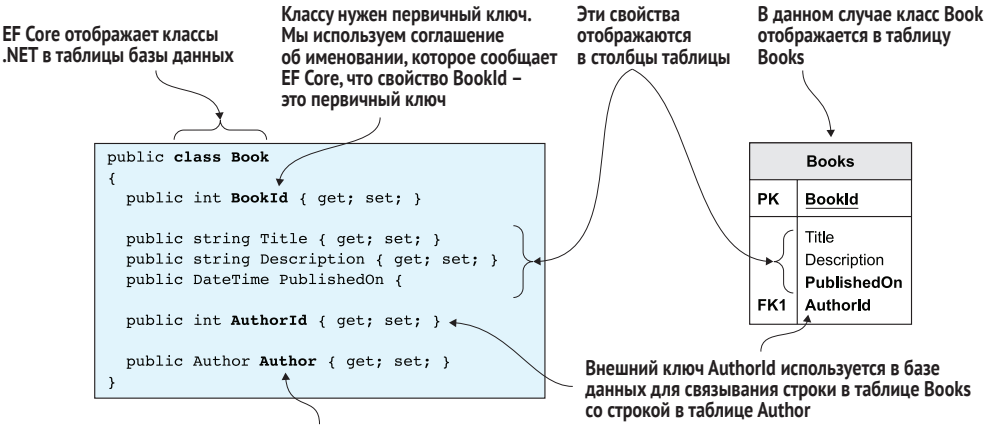


Рис. 1.4 Класс .NET Book слева отображается в таблицу базы данных Books справа. Это типичный способ создания приложения с несколькими классами, которые отображаются в таблицы базы данных

1.8.2 DbContext

Другая важная часть приложения – это DbContext, создаваемый вами класс, который наследует от класса EF Core DbContext. Этот класс содержит необходимую информацию, чтобы настроить отображение в базу данных. Этот класс используется в коде для доступа к базе данных (см. раздел 1.9.2). На рис. 1.5 показан класс DbContext, `AppDbContext`, используемый консольным приложением MyFirstEfCoreApp.

В нашем небольшом приложении все решения по моделированию принимает EF Core, используя набор соглашений. Есть много различных способов сообщить EF Core, что такое модель базы данных, и эти команды могут быть сложными. Потребуется материал из глав 7 и 8 и немного сведений из главы 10, чтобы охватить все варианты, доступные вам как разработчику.

Кроме того, используется стандартный подход для настройки доступа к базе данных в консольном приложении: переопределение метода `OnConfiguring` внутри DbContext и предоставление всей информации, необходимой EF Core для определения типа и местоположения базы данных. Недостатком такого подхода является то, что имеется фиксированная строка подключения, и это затрудняет разработку и модульное тестирование.

Для веб-приложений ASP.NET Core эта проблема серьезнее, потому что одновременно нужен доступ к локальной базе данных для тестирования и к базе данных, развернутой в промышленном окружении. В главе 2, когда вы начнете создавать веб-приложение ASP.NET Core, вы познакомитесь с другим подходом, который позволит изменять строку подключения к базе данных (см. раздел 2.2.2).

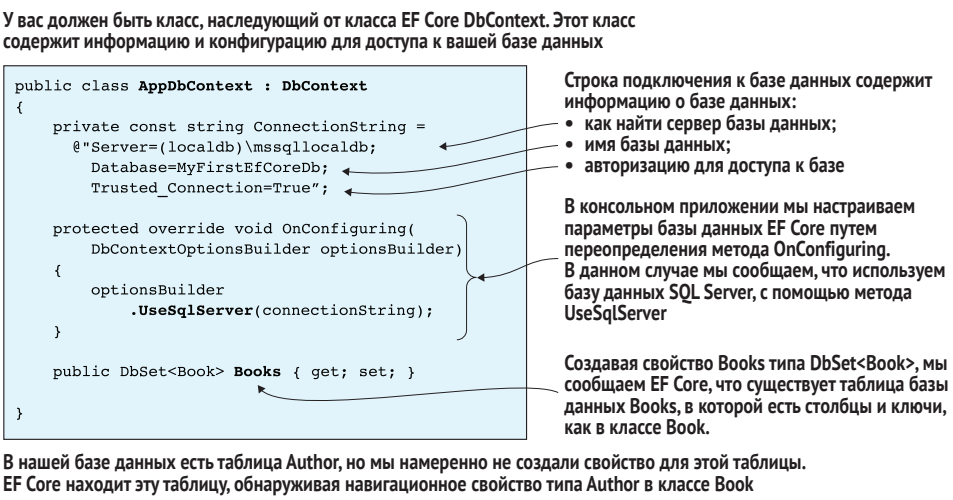


Рис. 1.5 Две основные части DbContext, созданные для консольного приложения MyFirstEfCoreApp. Сначала настройка параметров базы данных определяет, какой тип базы данных использовать и где ее можно найти. Далее свойство (или свойства) `DbSet<T>` сообщает EF Core, какие классы нужно отобразить в базу данных

1.9 Заглянем под капот EF Core

Теперь, когда мы запустили приложение MyFirstEfCoreApp, стоит разобраться, как работает библиотека EF Core. Акцент делается не на коде приложения, а на том, что происходит внутри библиотеки при чтении и записи данных в базу данных. Моя цель – предоставить вам ментальную модель того, как EF Core обращается к базе данных. Эта модель должна помочь вам, когда вы углубитесь в бесчисленное множество команд, описанных в оставшейся части книги.

Действительно ли нужно знать внутреннее устройство EF Core, чтобы использовать его?

Можно использовать библиотеку EF Core, не беспокоясь о том, как он работает. Но знание того, что происходит внутри, поможет понять, почему различные команды работают именно так. Кроме того, вы будете лучше вооружены, когда нужно будет отладить код доступа к базе данных.

Следующие страницы содержат множество пояснений и диаграмм, чтобы показать вам, что происходит внутри EF Core. EF Core «скрывает» базу данных, чтобы вы как разработчик могли легко писать код доступа к базе данных, что хорошо работает на практике. Но, как я уже сказал, знание того, как работает EF Core, может помочь вам, если вы хотите сделать что-то посложнее или если что-то работает не так, как ожидалось.

1.9.1 Моделирование базы данных

Прежде чем вы сможете что-либо делать с базой данных, EF Core должен пройти процесс, который я называю *моделированием базы данных*. Моделирование – способ EF Core определить, как выглядит база данных, глядя на классы и другие данные конфигурации. Затем EF Core использует полученную модель при всех обращениях к базе данных.

Процесс моделирования запускается при первом создании класса DbContext, в данном случае `AppDbContext` (показанного на рис. 1.5). У него есть одно свойство `DbSet<Book>` – это способ доступа кода к базе данных.

На рис. 1.6 представлен общий вид процесса, используемого EF Core для моделирования базы данных. В последующих главах вы познакомитесь с рядом команд, которые позволят вам более точно настроить базу данных, а пока вы будете использовать конфигурации по умолчанию.

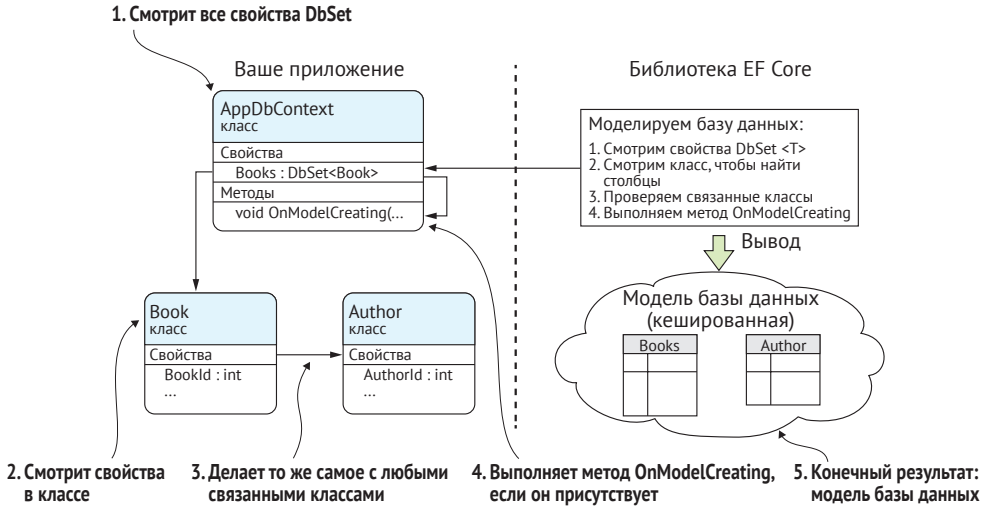


Рис. 1.6 На рисунке показано, как EF Core создает модель базы данных, в которую отображаются ваши классы. Сначала он смотрит на классы, которые вы определили с помощью свойств `DbSet<T>`, а потом смотрит на все ссылки на другие классы. Используя эти классы, EF Core может построить модель базы данных по умолчанию. Но далее он выполняет метод `OnModelCreating` в DbContext, который можно переопределить, чтобы добавить свои конкретные команды и настроить базу данных так, как вы хотите

На рис. 1.6 показаны этапы моделирования, которые EF Core использует в классе `AppDbContext`. Это происходит при первом создании экземпляра класса. (После этого модель кешируется, поэтому последующие экземпляры создаются быстро.) Ниже представлено более подробное описание процесса:

- EF Core просматривает DbContext приложения и находит все публичные свойства `DbSet<T>`. На основе этих данных он определяет начальное имя для одной найденной таблицы: Books;

- EF Core просматривает все классы, на которые ссылаются в `DbSet<T>`, и смотрит их свойства для определения имен столбцов, типов и т. д., а также ищет специальные атрибуты класса и/или свойств, обеспечивающие дополнительную информацию о моделировании;
- EF Core ищет любые классы, на которые ссылаются классы `DbSet<T>`. В нашем случае у класса `Book` есть ссылка на класс `Author`, поэтому EF Core сканирует и его. Он выполняет тот же поиск по свойствам класса `Author`, что и в случае с классом `Book` на этапе 2, а также принимает имя класса `Author` в качестве имени таблицы;
- напоследок EF Core вызывает виртуальный метод `OnModelCreating` внутри DbContext. В этом простом приложении вы не переопределяете метод `OnModelCreating`, но если бы вы это сделали, то могли бы предоставить дополнительную информацию через Fluent API, чтобы выполнить настройку моделирования;
- EF Core создает внутреннюю модель базы данных на основе собранной информации. Эта модель базы данных кешируется, поэтому последующий доступ будет быстрее. Затем она используется для выполнения всех обращений к базе данных.

Возможно, вы заметили, что на рис. 1.6 базы данных нет. Это объясняется тем, что когда EF Core создает свою внутреннюю модель, он не смотрит на базу данных. Я подчеркиваю этот факт, чтобы показать, насколько важно построить хорошую модель нужной вам базы данных; в противном случае могут возникнуть проблемы, если существует несоответствие между тем, как по мнению EF Core должна выглядеть база данных, и фактической базой.

В своем приложении вы можете использовать EF Core для создания базы данных, и в этом случае шансы на несоответствие равны нулю. Тем не менее если вам нужна хорошая и эффективная база данных, стоит позаботиться о том, чтобы создать правильное представление нужной базы данных в своем коде, чтобы созданная база работала нормально. Варианты создания, обновления и управления структурой базы данных – обширная тема, подробно описанная в главе 9.

1.9.2 Чтение данных

Теперь у вас есть доступ к базе данных. У консольного приложения есть команда `l (list)`, которая читает данные из базы и выводит информацию в терминал. На рис. 1.7 показан результат запуска консольного приложения с командой l.

В следующем листинге показан код, который вызывается для вывода списка всех книг с авторами в консоль.

Листинг 1.2 Код для перечисления всех книг и вывода их в консоль

```
public static void ListAll()
{
    using (var db = new AppDbContext())
```

Создается DbContext приложения, через который выполняются все обращения к базе данных

```
    {
        foreach (var book in
            db.Books.AsNoTracking()
            .Include(book => book.Author))
        {
            var webUrl = book.Author.WebUrl == null
                ? "- no web URL given -"
                : book.Author.WebUrl;
            Console.WriteLine(
                $"{book.Title} by {book.Author.Name}");
            Console.WriteLine(" " +
                "Published on " +
                $"{book.PublishedOn:dd-MMM-yyyy}" +
                $". {webUrl}");
        }
    }
}
```

Читаются данные обо всех книгах. AsNoTracking указывает на то, что доступ к данным осуществляется в режиме «только для чтения»

Метод Include делает так, чтобы с каждой книгой загружалась информация об авторе. Смотрите главу 2 для получения дополнительной информации

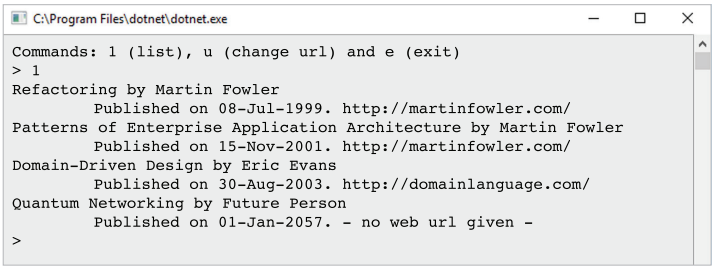


Рис. 1.7 Вывод консольного приложения при перечислении содержимого базы данных

EF Core использует язык Language Integrated Query (LINQ) от Microsoft для передачи нужных команд и обычные классы .NET для хранения данных. Запрос из листинга 1.2 не включает никаких методов LINQ, но позже вы увидите много примеров с LINQ.

ПРИМЕЧАНИЕ *Изучение LINQ будет иметь важное значение для вас, поскольку EF Core использует команды LINQ для доступа к базе данных.* В приложении содержится краткое введение в LINQ. Также доступно множество онлайн-ресурсов; см. http://mng.bz/YqBN.

Две строки кода, выделенные жирным шрифтом в листинге 1.2, обеспечивают доступ к базе данных. Теперь посмотрим, как EF Core использует этот код для доступа к базе данных и возврата необходимых книг с указанием их авторов. На рис. 1.8 показано, как выполняется запрос к базе данных.

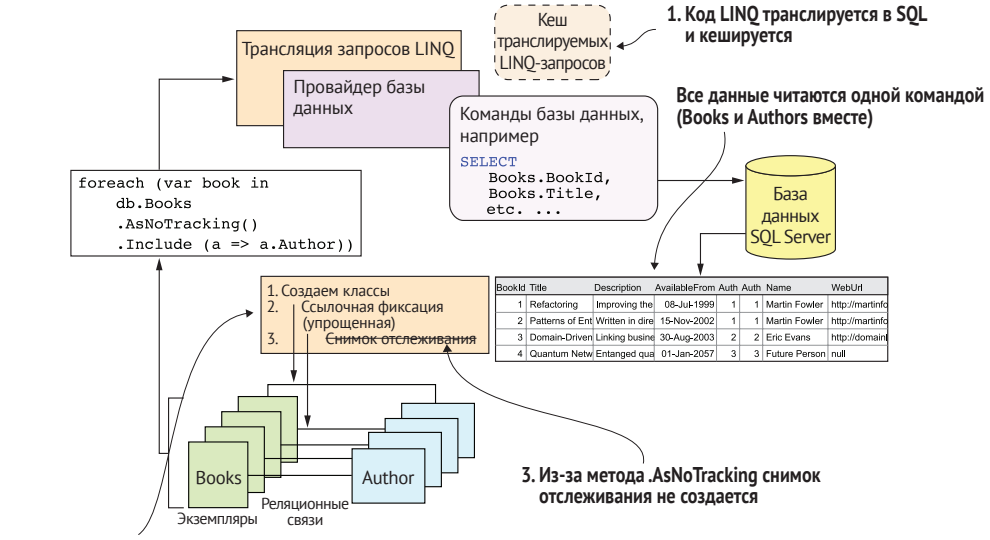


Рис. 1.8 Как EF Core выполняет запрос к базе данных

Процесс чтения данных из базы данных выглядит следующим образом:

- запрос `db.Books.AsNoTracking().Include(book => book.Author)` обращается к свойству `DbSet <Book>` в DbContext приложения и добавляет `.Include(book => book.Author)` в конце, чтобы запросить загрузку связанных данных из `Author`. Далее провайдер базы данных выполняет преобразование в SQL-команду для доступа к базе данных. Полученный в результате SQL запрос кешируется, чтобы избежать затрат на повторную трансляцию, если тот же запрос к базе данных будет использоваться снова.

EF Core пытается максимально эффективно использовать доступ к базе данных. В данном случае он объединяет две таблицы, которые ему нужно прочитать, Books и Author, в одну большую таблицу, чтобы можно было выполнять работу за один запрос к базе данных. В следующем листинге показан SQL запрос, созданный EF Core и провайдером базы данных.

Листинг 1.3 Команда SQL, созданная для чтения данных из таблиц Books и Author

```
SELECT [b].[BookId],
[b].[AuthorId],
[b].[Description],
[b].[PublishedOn],
```

```
[b].[Title],
[a].[AuthorId],
[a].[Name],
[a].[WebUrl]
FROM [Books] AS [b]
INNER JOIN [Author] AS [a] ON
[b].[AuthorId] = [a].[AuthorId]
```

После того как провайдер базы данных прочитает данные, EF Core передает их, используя процесс, который (а) создает экземпляры классов .NET и (б) использует ссылки реляционной базы данных, которые называются *внешними ключами*, для правильного связывания классов .NET по ссылке. Это называется ссылочная фиксация (relational fixup). Поскольку мы добавили метод `AsNoTracking`, используется упрощенная фиксация по соображениям скорости.

ПРИМЕЧАНИЕ Различия между упрощенной фиксацией с использованием метода `AsNoTracking` и обычной ссылочной фиксацией обсуждаются в разделе 6.1.2.

Результат – набор экземпляров классов .NET со свойством `Author` класса `Book`, связанным с классом `Author`, содержащим информацию об авторе. В этом примере у двух книг один автор, Мартин Фаулер, поэтому есть два экземпляра класса `Author`, каждый из которых содержит одну и ту же информацию о Мартине Фаулере.

Поскольку этот код включает команду `AsNoTracking`, EF Core знает, как подавить создание *снимка отслеживания изменений данных* (*tracking snapshot*). Снимки отслеживания используются для обнаружения изменений данных, как будет показано в примере редактирования столбца базы данных WebUrl в разделе 1.9.3. Поскольку этот запрос с доступом только на чтение, отключение моментального снимка отслеживания делает команду быстрее.

1.9.3 Обновление

Теперь нужно использовать вторую команду update (`u`) в MyFirstEfCoreApp, чтобы обновить столбец WebUrl в таблице Author книги «*Quantum Networking*». Как показано на рис. 1.9, сначала перечисляются все книги, чтобы показать, что у последней книги нет URL-адреса автора. Затем выполняется команда `u`, которая запрашивает новый URL-адрес автора для последней книги, «*Quantum Networking*». Вы вводите новый URL-адрес `httqs://entangled.moon` (это книга о вымышленном будущем, так почему бы не использовать вымышленный URL-адрес!), и после обновления команда снова перечисляет все книги, показывая, что URL-адрес автора изменился (красным обведено место, где должен быть указан URL-адрес, и адрес, который появился после).

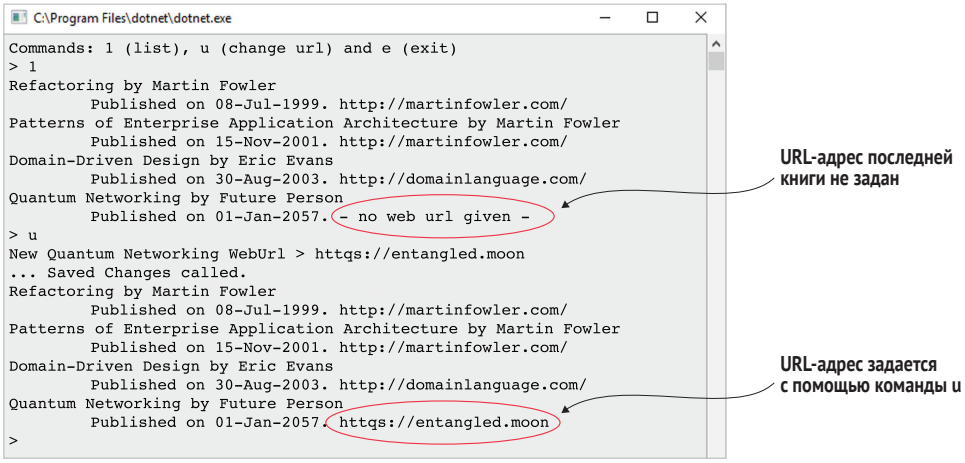


Рис. 1.9 На этом рисунке показано обновление в действии. Первая команда – l (list), которая показывает каждую книгу, имя ее автора и URL-адрес на следующей строке. Затем вы нажимаете u (update), что позволяет обновить URL-адрес автора последней книги. Команда обновления вызывает команду list, чтобы было видно, что обновление прошло успешно

Код для обновления столбца WebUrl в таблице Author, связанной с книгой «*Quantum Networking*», показан здесь:

Листинг 1.4. Код для обновления столбца WebUrl

```
public static void ChangeWebUrl()
{
    Console.Write("New Quantum Networking WebUrl > ");
    var newWebUrl = Console.ReadLine();

    using (var db = new AppDbContext())
    {
        var singleBook = db.Books
            .Include(book => book.Author)
            .Single(book => book.Title == "Quantum Networking");

        singleBook.Author.WebUrl = newWebUrl;
        db.SaveChanges();
        Console.WriteLine("... SavedChanges called.");
    }
    ListAll();
}
```

Считывается из консоли новый URL-адрес

Загружается информация об авторе вместе с книгой

Выбирается книга с названием Quantum Networking

Чтобы внести изменения в базу данных, изменяются значения, которые были прочитаны

Метод SaveChanges сообщает EF Core о необходимости проверки всех изменений считанных данных и записи этих изменений в базу

Отображается вся информация о книгах

На рис. 1.10 показано, что происходит внутри библиотеки EF Core. Этот пример намного сложнее, чем предыдущий пример чтения данных, поэтому позвольте дать вам несколько советов по поводу того, что искать.

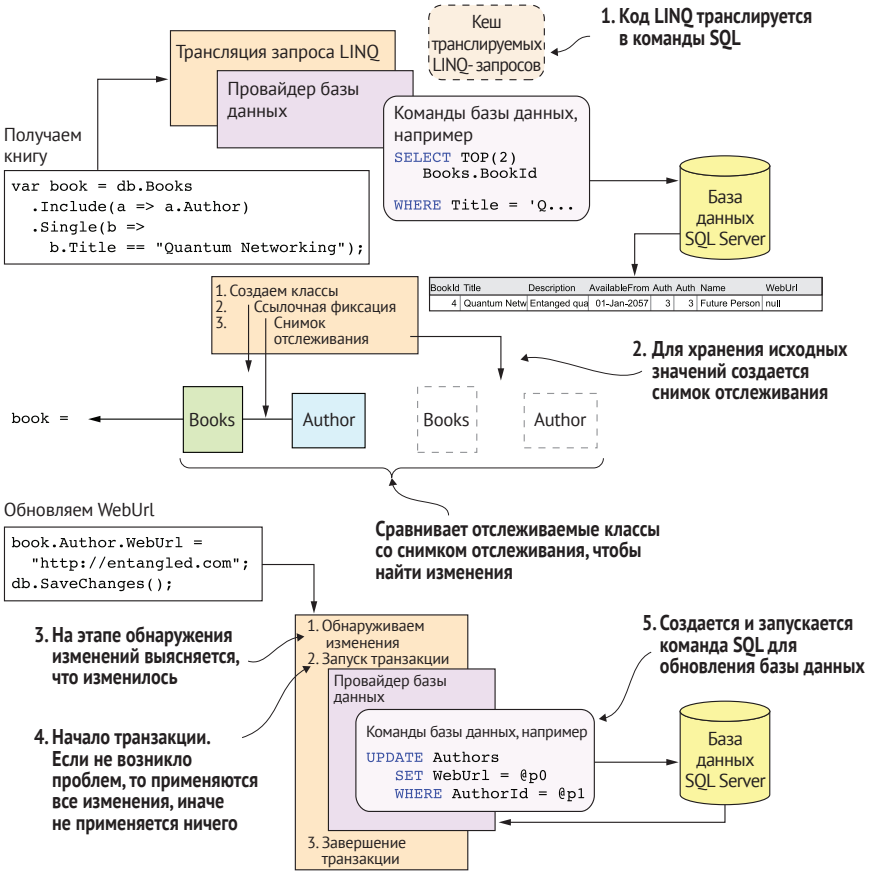


Рис. 1.10 На этом рисунке показано, что делает EF Core, когда вы обновляете свойство `WebUrl` и просите EF Core записать его в базу данных. Это довольно сложный рисунок, но если начать сверху и следовать за пронумерованным текстом, понять его будет легче. Все начинается с чтения, чтобы получить нужную книгу и автора. (Обратите внимание, что в этом процессе присутствует снимок отслеживания; см. этап 2.) Затем, когда вы обновляете `WebUrl` и вызываете метод `SaveChanges`, EF Core создает и выполняет правильную SQL-команду для обновления столбца WebUrl в правильной строке

Во-первых, этап чтения в верхней части диаграммы аналогичен примеру чтения и поэтому должен быть вам знаком. В данном случае запрос загружает конкретную книгу, используя ее заголовок в качестве фильтра. Важным изменением является пункт 2: создается снимок отслеживания.

Изменение происходит на этапе обновления в нижней половине диаграммы. Здесь видно, что EF Core сравнивает загруженные данные со снимком отслеживания, чтобы обнаружить изменения. Он видит, что было изменено только свойство `WebUrl`, и создает команду SQL, чтобы обновить только столбец WebUrl в нужной строке таблицы Author.

Я описал большинство шагов, а вот подробное описание того, как обновляется столбец WebUrl:

1. Приложение использует LINQ-запрос для поиска конкретной книги с информацией об авторе. EF Core превращает запрос в команду SQL для чтения строк, где Title имеет значение *Quantum Networking*, возвращая экземпляры обоих классов `Book` и `Author`, и проверяет, что была найдена только одна строка.
2. LINQ-запрос не включает метод `.AsNoTracking`, который использовался в предыдущих версиях чтения данных, поэтому запрос считается *отслеживаемым*. Следовательно, EF Core создает снимок загруженных данных.
3. Затем код изменяет свойство `WebUrl` в классе книги `Author`. При вызове метода `SaveChanges` на этапе обнаружения изменений сравниваются все классы, которые были возвращены из отслеживаемого запроса со снимком отслеживания. Так можно определить, что изменилось – в данном случае только свойство `WebUrl` класса `Author` с первичным ключом 3.
4. При обнаружении изменения EF Core начинает *транзакцию*. Каждое обновление базы данных выполняется как *атомарная единица*: если в базе данных происходит несколько изменений, они либо все успешные, либо все терпят неудачу. Это важный факт, потому что реляционная база данных может перейти в ненадлежащее состояние, если будет применена только часть изменений.
5. Запрос на обновление преобразуется провайдером базы данных в SQL-команду, которая выполняет обновление. Если команда была выполнена успешно, транзакция фиксируется, и происходит выход из метода `SaveChanges`; в противном случае возникает исключение.

1.10 Этапы разработки EF Core

EF Core и .NET Core прошли долгий путь с момента первого выпуска. Корпорация Microsoft усердно работала над улучшением производительности .NET Core, при этом добавляя дополнительные функциональные возможности до такой степени, что .NET 5 может заменить существующую платформу .NET Framework 4.8.

На рис. 1.11 показана история основных выпусков EF Core до настоящего момента. Номера версий EF Core следуют за номером версии NET Core. Обратите внимание, что выпуски в верхней части рисунка – это выпуски *с долгосрочной поддержкой* (LTS). Это означает, что выпуск поддерживается в течение трех лет после первого выпуска. Основные выпуски ожидаются каждый год, а LTS-релизы – каждые два года.

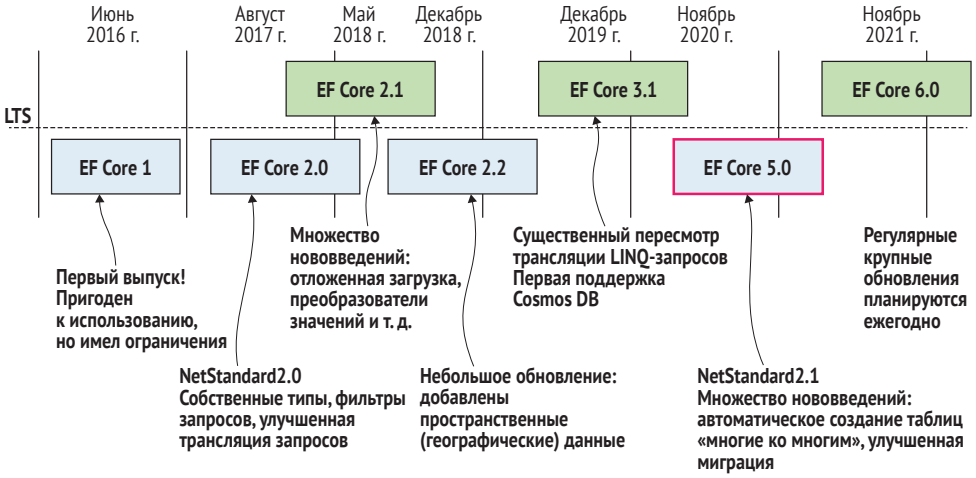


Рис. 1.11 На этом рисунке показана разработка EF Core, которая идет параллельно с разработкой NET, платформы для разработчиков с открытым исходным кодом. Версия EF Core 5 выделена, потому что эта книга охватывает все функциональные возможности EF Core до EF Core 5 включительно

1.11 Стоит ли использовать EF Core в своем следующем проекте?

Теперь, когда у вас есть краткий обзор того, что такое EF Core и как он работает, встает вопрос: стоит ли начинать использовать EF Core в своем проекте? Для всех, кто планирует перейти на EF Core, ключевой вопрос заключается в следующем: «Достаточно ли EF Core превосходит библиотеку доступа к данным, которую я использую сейчас, чтобы использовать ее в моем следующем проекте?» Изучение и внедрение любой новой библиотеки, особенно такой сложной, как EF Core, связано с затратами, поэтому это правильный вопрос. Вот мой взгляд на EF Core и .NET Core в целом.

1.11.1 .NET – это программная платформа будущего, и она будет быстрой!

Microsoft давно и упорно работала над улучшением производительности .NET Core, при этом добавляя дополнительные функциональные возможности. Ориентация на производительность перебросила веб-приложения на Microsoft ASP.NET Core с примерно 250-го места (ASP.NET MVC) на примерно 10–40-е места (ASP.NET Core) (в зависимости от нагрузки); см. http://mng.bz/Gxaq. Аналогичный, но меньший прирост производительности был добавлен в EF Core.

Microsoft заявила, что .NET 5 заменит существующую платформу .NET Framework 4.8, однако вспышка COVID-19 сорвала эти планы, и теперь .NET 6 заменит .NET Framework 4.8. Но здесь есть ясный сигнал: если вы начинаете новый проект, а у .NET 5 и EF Core есть функциональные возможности, необходимые вашему проекту, то переход на EF Core означает, что вы не останетесь за бортом.

1.11.2 Открытый исходный код и открытые сообщения

За многие годы Microsoft изменилась. Вся ее работа над .NET Core – это открытый исходный код, и множество внешних специалистов занимаются исправлением ошибок и добавлением нововведений, поэтому у вас может быть прямой доступ к коду, если он вам нужен.

Кроме того, впечатляет уровень открытости сообщений о том, что происходит в .NET Core и других продуктах. Команда EF Core, например, еженедельно выпускает обновления о том, что она делает, предоставляя множество ранних предварительных просмотров новых выпусков и делая регулярные ночные сборки EF Core доступными для всех. Команда серьезно относится к отзывам, а все работы и дефекты отображаются на страницах проблем репозитория EF Core.

1.11.3 Мультиплатформенные приложения и разработка

Как я сказал в начале главы, EF Core поддерживает несколько платформ; вы можете выполнять разработку и запускать приложения EF Core в Windows, Linux и Apple. Этот факт означает, что можно запускать приложения Microsoft на дешевых системах Linux. Также вполне возможна разработка на разных платформах. Артур Викерс, который является одним из ведущих инженеров команды EF Core, решил перейти с Windows на Linux в качестве основной платформы разработки. О его опыте можно прочитать на странице http://mng.bz/zxWa.

1.11.4 Быстрая разработка и хорошие функциональные возможности

Моя основная работа – разработчик по контракту. В типичном приложении, управляемом данными, я пишу большое количество кода доступа к базе данных, причем некоторые его части сложны. С EF Core я могу писать код доступа очень быстро и сделать это так, чтобы он был легким для понимания и рефакторинга, если он слишком медленный. Это основная причина, по которой я использую EF Core.

В то же время мне нужен инструмент объектно-реляционного отображения со множеством функциональных возможностей, чтобы я мог создать базу данных так, как я хочу, не сталкиваясь со слишком большим количеством препятствий в EF Core. Конечно, некоторые вещи, такие как создание общих табличных выражений SQL, исклю-

чены, но использование чистого SQL в некоторых случаях позволяет обходить подобные ограничения, если мне это нужно.

1.11.5 Хорошая поддержка

По EF Core есть хорошая документация (https://docs.microsoft.com/en-us/ef/core/index), и, конечно же, у вас есть эта книга, в которой собрана документация с подробными объяснениями и примерами, а также паттерны и практики, которые сделают вас отличным разработчиком. В интернете полно блогов об EF Core, в том числе и мой на странице https://www.thereformedprogrammer.net. А для вопросов и ошибок всегда есть Stack Overflow; http://mng.bz/0mDx.

Другая часть поддержки – это инструменты разработки. Microsoft, кажется, изменила фокус, предоставив поддержку нескольких платформ, но еще создала бесплатную кросс-платформенную среду разработки под названием VS Code, а также сделала свой основной инструмент разработки, Visual Studio (Windows и Mac), бесплатным для индивидуальных разработчиков и предприятий малого бизнеса; в разделе «Использование» в нижней части страницы по адресу www.visualstudio.com/vs/community подробно описаны все условия. Привлекательное предложение.

1.11.6 Всегда высокая производительность

Ах да, проблема с производительностью базы данных. Послушайте, я не буду говорить, что EF Core сразу же обеспечит невероятную производительность доступа к базе данных с красивым SQL и быстрым получением, внесением и обработкой данных для последующего их использования или хранения в базе. Это цена, которую вы платите за быструю разработку кода доступа к данным; вся эта «магия» внутри EF Core не может сравниться с написанным вручную SQL, но, возможно, вы удивитесь, насколько эффективной она может быть. Смотрите главу 15, где я по шагам настраиваю производительность приложения.

Но есть много возможностей улучшить производительность своих приложений. В моих приложениях я обнаружил, что только около 5–10 % запросов являются ключевыми, которым нужна ручная настройка. Главы 14 и 15 посвящены настройке производительности, как и часть главы 16. Эти главы показывают, что можно многое сделать для повышения производительности доступа к базе данных с помощью EF Core.

Также нет причин, по которым нельзя перейти на чистый SQL для доступа к базе данных. Это замечательно: быстро создать приложение с помощью EF Core, а затем преобразовать несколько мест, где EF Core не обеспечивает хорошую производительность, в команды на чистом SQL, используя ADO.NET или Dapper.

1.12 Когда не следует использовать EF Core?

Я явный сторонник EF Core, но не буду использовать его в клиентском проекте, если это не имеет смысла. Итак, рассмотрим несколько моментов, когда можно решить *не* использовать EF Core.

Первый очевиден: поддерживает ли он базу данных, которую вы хотите использовать? Список поддерживаемых баз данных можно найти на странице https://docs.microsoft.com/en-us/ef/core/providers.

Второй фактор – это необходимый уровень производительности. Если вы пишете, скажем, небольшой REST-совместимый сервис или бессерверную систему, то я не уверен, что EF Core – подходящий вариант; вы можете использовать быструю, но требовательную ко времени разработки библиотеку, потому что у вас не так много обращений к базе данных. Но если у вас большое приложение, со множеством скучных административных API и важными API для клиентов, вам может подойти гибридный подход. (Смотрите главу 15, где приводится пример приложения, написанного с использованием EF Core и Dapper.)

Кроме того, EF Core не очень подходит для пакетных (bulk) команд. Обычно такие задачи, как массовая загрузка больших объемов данных и удаление всех строк в таблице, можно реализовать быстрее, используя чистый SQL. Но несколько расширений EF Core для пакетных CRUD-операций (некоторые с открытым исходным кодом, а некоторые платные) могут помочь; попробуйте выполнить поиск по запросу «пакетная загрузка в EF Core» («EF Core bulk loading»), чтобы найти возможные библиотеки.

Резюме

- EF Core – инструмент объектно-реляционного отображения (ORM), использующий язык LINQ для определения запросов к базе данных и возврата данных в связанные экземпляры классов .NET.
- EF Core разработан для быстрого и интуитивно понятного написания кода для доступа к базе данных. У нее есть множество функциональных возможностей, соответствующих многим требованиям.
- Были показаны различные примеры того, что происходит внутри EF Core. Эти примеры помогут вам понять, что могут делать команды EF Core, описанные в последующих главах.
- Есть много веских причин рассмотреть возможность использования EF Core: за ним стоит большой опыт, у него хорошая поддержка, и он работает на нескольких платформах.

Для читателей, знакомых с EF6.x:

- ищите в книге примечания по EF6. В них отмечены различия между подходом на базе EF Core и подходом на базе EF6.x. Также не за-

бывайте про резюме в конце каждой главы, где указаны основные изменения EF Core в этой главе;

- рассматривайте EF Core как новую библиотеку, которая была написана для имитации EF6.x, но работает по-другому. Такой образ мышления поможет вам определить улучшения EF Core, меняющие способ доступа к базе данных;
- EF Core больше не поддерживает подход EDMX / конструктор баз данных, который использовался в более ранних версиях EF.