Entity Framework Core Базовый

Моделирование сущностей в EF Core

Introduction

Науринский Юрий
.NET Software Engineer

yurii.naurynskyi

yurii.naurynskyi

Entity Framework Core

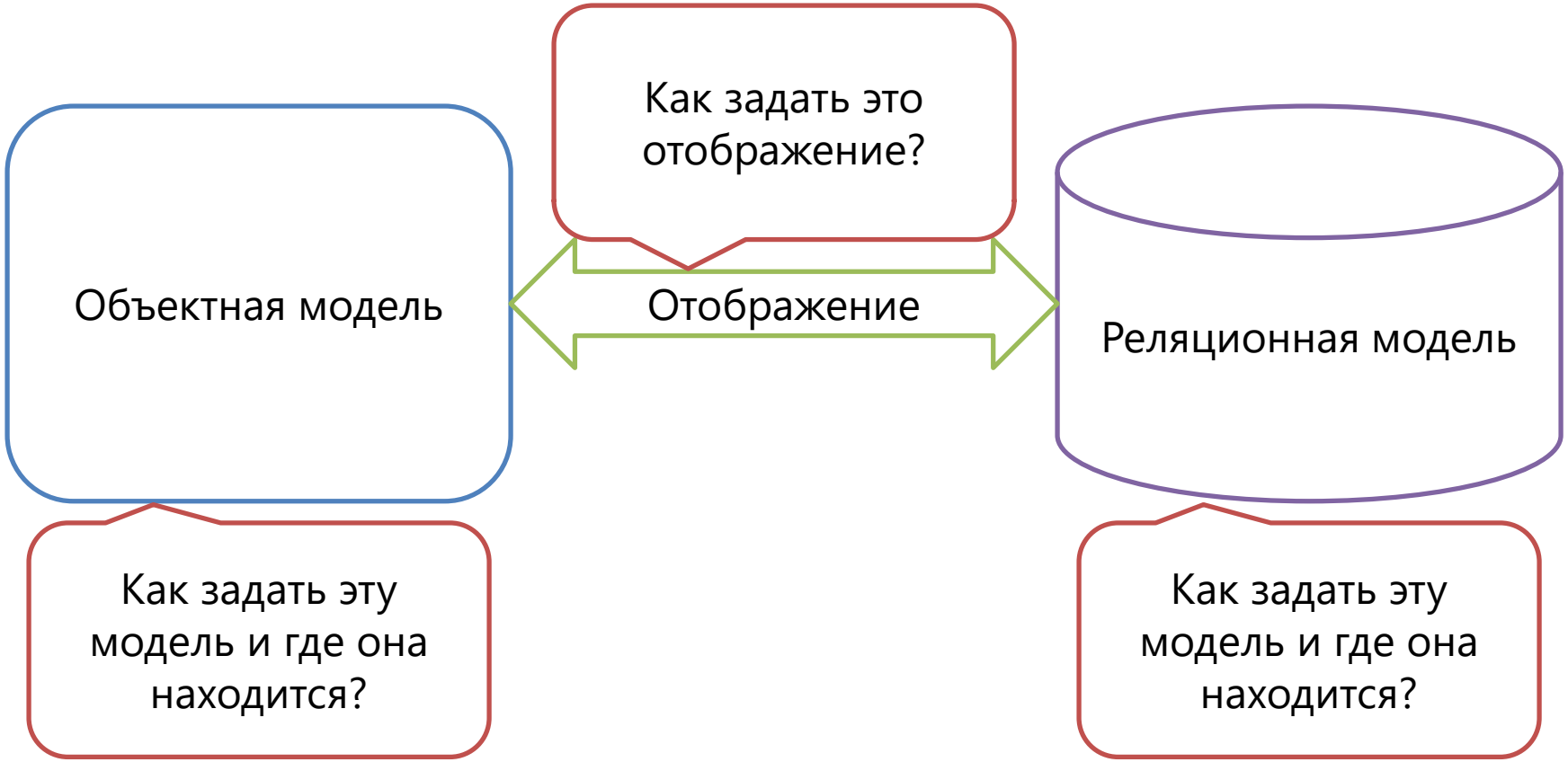
Тема

Моделирование сущностей в EF Core

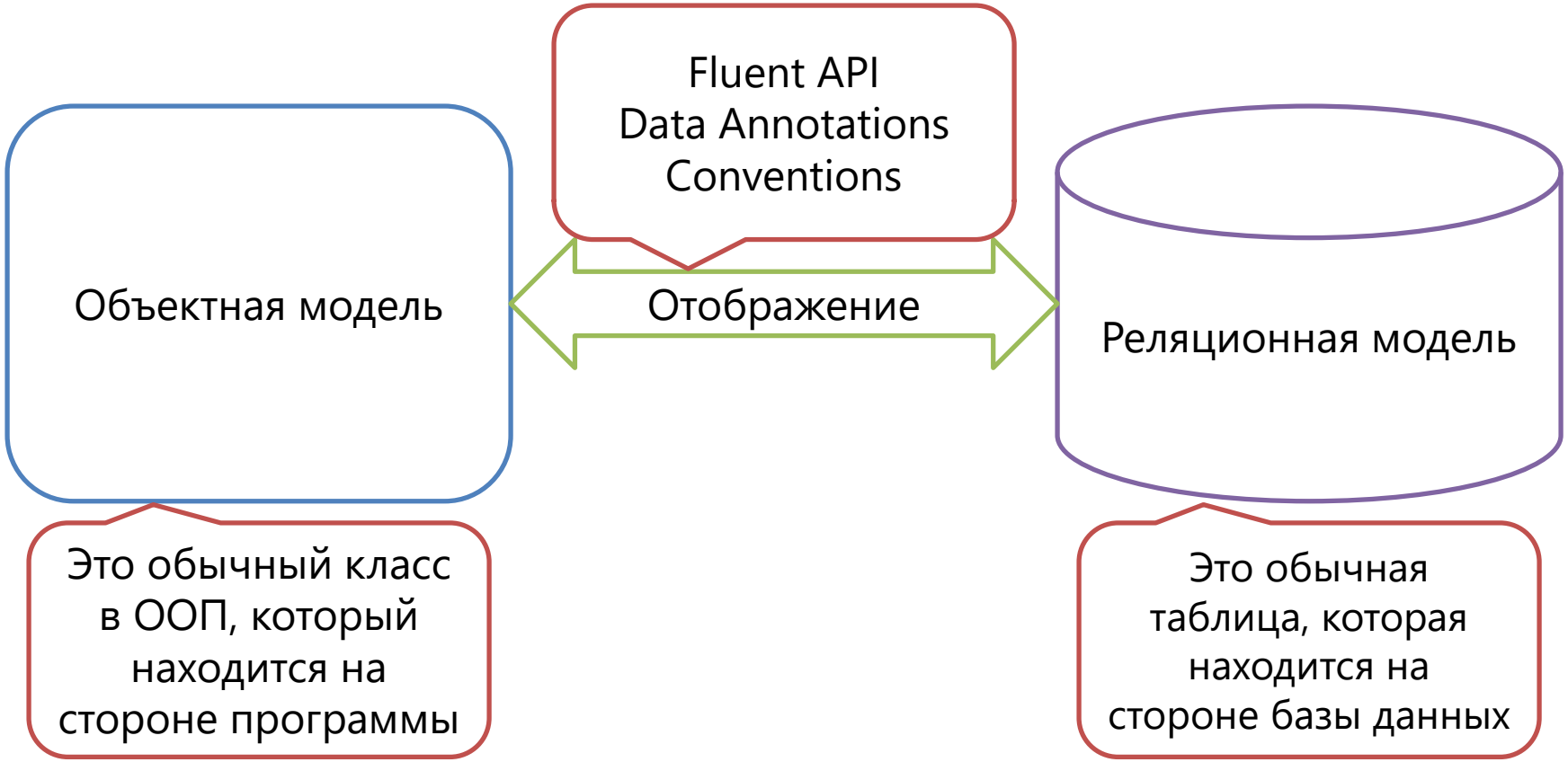
План урока

1. Связь между концепциями ORM и EF Core
2. Конфигурация модели в EF Core
3. Fluent API &
4. Data Annotations
5. Conventions
6. IEntityTypeConfiguration<T>
7. Ключи
8. Индексы
9. Специфические особенности EF Core
10. Отношения между сущностями в EF Core
11. Демо

Как связаны концепции ORM и EF Core



Связь между концепциями ORM и EF Core



Конфигурация модели в EF Core

Fluent API	Самый мощный метод конфигурации
	Не требует модификации описываемых сущностей
	Самый высокий приоритет
	Самый многословный метод конфигурации
Data Annotations	Требует применения специальных атрибутов в объектной модели
	Требует модификации описываемых сущностей
	Средний приоритет
Conventions	Неявно заданные разработчиками библиотеки правила, как отображать объектную модель на реляционную и наоборот
	Не выражены явно в коде
	Самый низкий приоритет
	Удобные, если их знаешь
	Неудобные, если их не знаешь

Conventions в EF Core

Конвенции самого класса сущности	Имя таблицы – имя свойства DbSet
	Класс должен быть публичным
	Класс НЕ должен быть статическим
	Всегда необходима возможность создания класса через конструктор (автоматически исключает абстрактные классы)
Конвенции свойств класса	Имя свойства – имя колонки в таблице
	В качестве колонок учитываются все **публичные** свойства класса с **любым** модификатором доступа для метода **set**
	Свойства только для чтения могут быть учтены только при использовании Fluent API
Конвенции типов данных	Примитивные .NET типы данных транслируются в соответствующие типы на стороне базы данных
	Исключения: string, Guid, DateTime

Conventions в EF Core

Конвенции размеров данных	Размер данных в базе соответствует размеру примитивного типа в .NET
	Кроме string – размер максимальный
Конвенция трансляции null типов	Все примитивные типы и структуры по умолчанию NOT NULL
	Кроме строки, которая может быть NULL
	Оператор ? (Nullable<T>where T : struct) учитывается
Конвенции первичных ключей	Свойство является первичным ключом, если название или **Id** или **{ClassName}Id.**
	Возможно определить только один первичный ключ для сущности!

System.ComponentModel.DataAnnotations

Key	Первичный ключ сущности
Timestamp	Временная метка Используется для транзакционной обработки запросов Всегда тип byte[]
ConcurrencyCheck	Маркирует свойство, теперь оно включено в транзакционную обработку
Required	Делает колонку NOT NULL
MinLength & MaxLength	Задают длину массива или строки
StringLength	Задает максимальную длину строки

Entity Framework Core Базовый

System.ComponentModel.DataAnnotations.Schema

Table	Название таблицы и её схема
Column	Название колонки
	Позиция колонки
	Тип колонки
ForeignKey	Маркирует свойство как использующееся в качестве внешнего ключа
DatabaseGenerated	Computed – БД генерирует значение, когда колонка вставлена или обновлена
	Identity – БД генерирует значение, когда колонка вставлена
	None – БД никогда не генерирует значение
NotMapped	Класс никак не транслируется в таблицу в БД
	Свойство никак не транслируется в колонку в БД
InverseProperty	Маркирует это свойство как соответствующее другому свойству из связанной сущности

Fluent API

Варианты конфигурации	`OnModelCreating(ModelBuilder modelBuilder)`
	`IEntityTypeConfiguration<T>`
Используется ModelBuilder	Специальный класс, который предоставляет методы для конфигурации сущностей и отношений между ними

IEntityTypeConfiguration<T>

Использует EntityTypeBuilder<T>	Предназначен для конфигурации конкретной сущности, в этом отличие от ModelBuilder
	Вызывается как `modelBuilder.Entity<T>`
Варианты применения в OnModelCreating	`modelBuilder.ApplyConfiguration(new Configuration<T>());`
	`modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());` (неопределенный порядок выполнения конфигураций)

Какую конфигурацию модели использовать

Ловушка конвенций	Практически всегда лучше максимально явно прописывать все сущности, их свойства и отношения между ними
	Для учебных проектов – особенно полезно
Data Annotations	Так как они полностью покрываются с помощью Fluent API, их использование малоприменимо
Максимальная ясность и прозрачность – Fluent API	Если проект с малым количеством сущностей – конфигурирование в OnModelCreating
	Иначе - использование IEntityTypeConfiguration<T>

Вывод – максимальное использование Fluent API & IEntityTypeConfiguration<T>

Ключи

Первичный ключ	Используется для уникальной идентификации сущности
	Имя в БД – PK_{ClassName}
	Может быть составным
	Конфигурируется всеми тремя способами
	Возможна как автоматическая генерация на стороне БД, так и полностью ручное управление
Альтернативный ключ	Альтернативный вариант идентификации сущности, используется в связях между сущностями
	Имя в БД – AK_{ClassName}_{PropertyName}
	Может быть составным
	Не стоит использовать только для уникальности
	Конфигурируется через Fluent API

Индексы

Специальный объект базы данных, который создаётся для улучшения производительности поиска в БД	Data Annotations
	Fluent API
Уникальность	По умолчанию не уникальный
	Может требовать уникальность
Свойства индексов	Возможно переопределить имя индекса
	Индекс может быть составным
	Индексы можно фильтровать по какому-либо условию
	По умолчанию для MS SQL Server базы данных для всех Nullable частей составного уникального индекса добавляется фильтр IS NOT NULL

Специфические особенности EF Core

Генерация значений	Теневые свойства	Последовательности	Резервные поля
Пользовательские конвертация и сравнение сущностей	Разделение таблиц	Собственные типы сущностей	Сущности без ключа

Генерация значений

Значение по умолчанию

- Срабатывает при вставке колонки в таблицу
- Константное значение
- Вычисляемое с помощью SQL значения

Вычисляемое значение

- Хранимое
  - Вычисляется каждый раз при обновлении сущности
- Виртуальное
  - Вычисляется каждый раз при чтении колонки

Конфигурируются через Fluent API

Для первичного ключа

- Возможно указать что значение генерируется при вставке колонки в таблицу

В общем случае явное указание генерации каких-либо значений зависит от провайдера БД

Можно отключить генерацию значений, которая задана через конвенции

- Например с помощью атрибута DatabaseGenerated

Теневые свойства

Значения, которые существуют на стороне реляционной модели, но которых нет в объектной

- Обычно используется для того, чтобы задать отношения между сущностями
- Или, например, когда в таблице есть служебные данные, которые не имеют отношения к основной логике таблицы

Конфигурируются через FluentAPI

Возможно получить их значение через EF Core

Автоматически создается теневое свойство, когда есть отношение между сущностями, но нет явно заданного внешнего ключа

- Создаётся в зависимой сущности
- Автоматическое имя – {DependentPropertyName}{PrincipalPropertyName}

Последовательности

Последовательность уникальных чисел	В основном используются в реляционных БД	Не зависят от какой-то определенной таблицы
Конфигурируются через Fluent API	Зависят от провайдера БД	Перед применением необходимо удостовериться что используемый провайдер БД поддерживает последовательности

Резервные поля

Позволяют использовать поля вместо свойств в объектной модели

Используются для нарушения инкапсуляции, когда EF Core должен иметь возможность читать/записывать в поля, которые скрыты за модификаторами доступа

- Например, можно создать свойство только для чтения, с помощью данной особенности

По умолчанию EF Core пишет в резервное поле и читает из него

- Начиная с EF Core 3.0

Конфигурируются через Data Annotations & Fluent API

Следует крайне осторожно использовать данную особенность

- Неявно усложняет объектную модель
- Вносят дополнительную логику, которой возможно не место в объектной модели

Пользовательские конвертация и сравнение сущностей

Пользовательская конвертация сущностей позволяет добавить свой конвертер для отображения пользовательских типов данных на реляционную модель

- Задаются как соотношение между ModelClrType – тип в .NET и ProviderClrType – тип БД
- Необходимо задать конвертацию в обе стороны
- Есть широкий список встроенных конвертаций

Пользовательское сравнение сущностей используется в Change Tracking для сравнения пользовательских типов, которые используют свою конвертацию

- Как правильно сравнить сущности?
  - Необходимо переопределить Equals & GetHashCode если используется класс
  - Структуры используют структурное сравнение

Для наиболее корректной работы – лучше всего использовать иммутабельные сущности

- Если это невозможно – использовать класс ValueComparer

Разделение таблиц

Возможно создать несколько объектных моделей, которые используют разные свойства реляционной модели

- Иными словами возможно создать несколько классов, которые делят между собой строки одной и той же таблицы
- Возможно различное конфигурирование этих разных классов
- Возможно различные чтение/запись разных классов
- Обязательно нужно задать первичный ключ в каждом классе и он обязан совпадать с одной и той же колонкой в таблице

Обычно используется для инкапсуляции данных, также для решения некоторых проблем с производительностью

ConcurrencyToken обязан быть во всех классах, которые используют эту таблицу

Собственные типы сущностей

Используются для моделирования неотъемлемых частей сущности

- Собственные типы сущности не могут существовать без главного типа
- Всегда отношение один к одному к главной сущности
- Всегда зависимы по отношению к главной сущности
- Неявно создаётся первичный ключ, который совпадает с первичным ключом главной сущности
- Если необходимо использовать коллекцию значений собственного типа – необходимо явно задать первичный ключ собственному типу
- Могут быть вложенными

Конфигурируются через Data Annotations & Fluent API

По умолчанию собственный тип находится в той же таблице, что и главный тип

- Можно сконфигурировать так, чтобы собственный тип сущности находился в отдельной таблице

По умолчанию загружаются вместе с главной сущностью

- Даже если находятся в другой таблице

Сущности без ключа

Используются в основном как:

- Результат сырого запроса к БД
- Результат выполнения запросов определенных в модели данных
- Сущность для конфигурации View
- Используется для таблиц, у которых нет первичного ключа

Конфигурируются через Data Annotations & Fluent API

Так как у них нет ключа – любые операции с ключом для них недоступны

Недоступны любые CUD манипуляции

Отношения между сущностями

One-to-one

Отношение между главной и зависимой сущностью как один к одному

- Одной главной сущности соответствует одна зависимая и наоборот
- В данном случае грань между главной и зависимой сущностью размывается
- Обычно представлено как ссылка на главную сущность у зависимой и наоборот

Зависимая сущность может быть как и обязательной, так и опциональной

One-to-many

Отношение между главной и зависимой сущностью как один ко многим

- Одной главной сущности соответствует множество зависимых
- Обычно представлено как коллекция зависимых сущностей в главной сущности

Зависимых сущностей может быть сколько угодно, вплоть до того, что их может и не быть

Many-to-many

Отношение между равноправными сущностями как многие ко многим

- Самый сложный вариант отношений
- Множеству одних сущностей соответствует множество других
- Обычно предоставлено как
  - Две таблицы сущностей (не содержат ключей, которые относятся к промежуточной таблице)
  - Промежуточная таблица
  - Промежуточная таблица содержит в себе два ключа, которые относятся к обеим таблицам сущностей

Моделируется в том числе как два отношения One-to-many, где с одной стороны таблица сущностей, с другой стороны промежуточная таблица

Сущностей в промежуточной таблице может быть сколько угодно, вплоть до того, что их может и не быть

Общий совет насчёт отношений между сущностями

Лучше всегда явно прописывать отношения между сущностями через Fluent API и НЕ полагаться на конвенции

Демо

Полезные ссылки

- https://docs.microsoft.com/en-us/ef/core/modeling/
- https://www.learnentityframeworkcore.com/configuration/data-annotation-attributes
- https://www.entityframeworktutorial.net/code-first/dataannotation-in-code-first.aspx
- https://docs.microsoft.com/en-us/ef/core/modeling/entity-types
- https://docs.microsoft.com/en-us/ef/core/modeling/entity-properties
- https://docs.microsoft.com/en-us/ef/core/modeling/generated-properties
- https://docs.microsoft.com/en-us/ef/core/modeling/shadow-properties
- https://docs.microsoft.com/en-us/ef/core/modeling/sequences
- https://docs.microsoft.com/en-us/ef/core/modeling/backing-field
- https://docs.microsoft.com/en-us/ef/core/modeling/value-conversions
- https://docs.microsoft.com/en-us/ef/core/modeling/value-comparers?tabs=ef5
- https://docs.microsoft.com/en-us/ef/core/modeling/table-splitting
- https://docs.microsoft.com/en-us/ef/core/modeling/owned-entities
- https://docs.microsoft.com/en-us/ef/core/modeling/keyless-entity-types

Информационный видеосервис для разработчиков программного обеспечения

TestProvider Pearson CyberBionic systematics Microsoft Partner Windows Azure EBA

Проверка знаний

TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!

Науринский Юрий
.NET Software Engineer

Entity Framework Core