# Отчет по Архитектурной Трансформации: Неделя 8 — Персистентность Данных и Реляционная Алгебра в Экосистеме Enterprise

## 1. Введение и Контекстуальный Анализ

Переход квалифицированного Unity-разработчика в сферу backend-инженерии, как обозначено в исходном плане обучения 1, достигает своей критической точки на восьмой неделе. Если предыдущие фазы фокусировались на синтаксических конструкциях C# и управлении потоками исполнения, то восьмая неделя знаменует собой фундаментальный сдвиг парадигмы: переход от **Stateful** (состояние в памяти) к **Stateless** (состояние в базе данных) архитектуре.

В среде Unity разработчик привык оперировать объектами, живущими в оперативной памяти на протяжении всей игровой сессии. Сохранение данных (Persistence) часто воспринимается как вспомогательная задача, решаемая через сериализацию JSON-файлов, PlayerPrefs или локальные SQLite базы данных, работающие в том же процессе. В Enterprise-системах, таких как проектируемый сервис "Календарь", база данных является центральным источником истины (Source of Truth), а приложение — лишь эфемерным обработчиком транзакций.

### 1.1. Анализ Технологического Стека

Выбор технологий для восьмой недели обусловлен требованиями современных высоконагруженных систем:

* **PostgreSQL:** Выбран в качестве СУБД не просто как хранилище, а как вычислительный узел. Поддержка JSONB, геометрических типов и типов диапазонов (range types) делает его идеальным кандидатом для календарных систем, где критична проверка временных коллизий.2
* **Entity Framework Core 8 (EF Core):** Используется не просто как маппер, а как Unit of Work. В версии.NET 8 EF Core получил существенные оптимизации производительности и новые возможности маппинга JSON и массивов, что позволяет строить гибридные (реляционно-документные) схемы.4

### 1.2. Сравнительная Характеристика Парадигм

Ниже приведена таблица, демонстрирующая ментальный сдвиг, необходимый разработчику на этой неделе.

| **Характеристика** | **Подход Unity (GameDev)** | **Подход Backend (Enterprise.NET)** | **Архитектурное следствие для Недели 8** |
| --- | --- | --- | --- |
| **Жизненный цикл данных** | Данные загружаются в RAM при старте сцены. | Данные подгружаются из БД по требованию (Lazy/Eager). | Необходимость минимизации выборки (Select) и проблема N+1.5 |
| **Конкурентность** | Однопоточная (в основном). Конфликты редки. | Тысячи параллельных запросов к одной записи. | Внедрение оптимистичной блокировки (xmin).6 |
| **Сложные типы** | Сериализация целого класса в JSON строку. | Структурированные типы БД (Arrays, JSONB, Ranges). | Использование специфичных возможностей PostgreSQL через Npgsql.7 |
| **Сохранение** | SaveGame() — полный дамп состояния. | Транзакции ACID. Изменение дельты. | Понимание ChangeTracker и транзакционных границ. |

## 2. Глобальный Промт для Ментора (GEM-Bot)

Данный промт разработан для загрузки в контекст AI-ассистента (Gemini/ChatGPT) с целью эмуляции работы с Senior Backend Architect. Он активирует режим строгой проверки архитектурных решений, специфичных для перехода с Unity на.NET Backend.

**Инструкция для пользователя:** Скопируйте текст ниже в чат с ботом в начале недели.

**Role:** You are a Principal Backend Architect specializing in.NET 8, Entity Framework Core, and PostgreSQL optimization. You are mentoring a Senior Unity Developer transitioning to Enterprise Backend engineering.

**Context:** The mentee understands C# deeply (memory, threading) but lacks "Database Thinking". They act as if data is always in RAM. We are in Week 8 of the roadmap: "Data Access Layer".

**Pedagogical Strategy:**

1. **Stop "Memory Thinking":** Whenever the mentee proposes loading a list to filter it with LINQ-to-Objects, correct them immediately. Force them to use IQueryable and Server-Side Evaluation.
2. **PostgreSQL Supremacy:** Do not accept generic SQL solutions. Demand the use of Postgres-specific features: jsonb for flexible settings, tsrange for booking intervals, and xmin for concurrency.
3. **Ruthless Performance Review:** Flag any potential N+1 query problems. Require AsNoTracking for all read operations. Explain the cost of Include vs SplitQuery.
4. **Comparison:** Constantly contrast scenarios with Unity. Example: "In Unity, you serialize the whole UserProfile. In EF Core, if you only need the Email, projecting the whole entity is network suicide."

**Immediate Task:** We are building the Data Layer for a "Calendar Service". Your goal is to guide the implementation of a conflict-free booking system using Database Exclusion Constraints, not C# validation logic. Await the first theoretical question.

## 3. План Обучения: День 1 — Архитектура Контекста и Подключение

Первый день посвящен фундаменту. Ошибка Unity-разработчиков — восприятие базы данных как простого хранилища файлов. Наша задача — представить БД как сложный механизм, требующий правильной конфигурации пула соединений и управления жизненным циклом.

### 3.1. Теоретический Блок

#### Тема 1.1: DbContext как Паттерн Unit of Work

В Unity менеджеры данных (DataManagers) часто являются синглтонами. В ASP.NET Core DbContext проектируется как Scoped сервис. Критически важно понять, почему внедрение DbContext в Singleton-сервис приведет к "Memory Leaks" и "Concurrency Issues" (поскольку DbContext не потокобезопасен).1

* **Механика:** DbContext накапливает изменения (ChangeTracker). Если контекст живет слишком долго, трекер раздувается, замедляя каждое обращение к данным.
* **Пулинг (DbContext Pooling):** В высоконагруженных системах создание объекта контекста может быть накладным. Механизм AddDbContextPool позволяет переиспользовать объекты контекста, сбрасывая их состояние, что напоминает паттерн Object Pooling в Unity.9

#### Тема 1.2: Провайдер Npgsql и Конфигурация

PostgreSQL требует специфической настройки. Стандартные соглашения именования в C# (PascalCase) конфликтуют с конвенциями Postgres (snake\_case).

* **Snake Case Rewriter:** Использование EFCore.NamingConventions для автоматического маппинга имен таблиц и колонок.
* **DateOnly/TimeOnly:** В.NET 8 и EF Core 8 появилась нативная поддержка типов DateOnly и TimeOnly. Это критично для календаря (дни рождения, время начала рабочего дня). Ранее приходилось использовать DateTime с игнорированием времени, что вело к ошибкам часовых поясов. Npgsql 8.0 мапит их напрямую на date и time.4

#### Тема 1.3: Управление Миграциями в Команде

Работа с миграциями в команде отличается от локальной разработки.

* **Idempotency:** Генерация идемпотентных SQL-скриптов для CI/CD пайплайнов. Скрипт должен уметь накатываться на базу в любом состоянии, не ломая её.
* **Model Snapshot:** Как EF понимает, что изменилось? Разбор файла ModelSnapshot и его роль в вычислении дельты изменений.4

### 3.2. Промт для Глубокого Исследования (Теория)

**Запрос к AI:** "Проанализируй внутреннее устройство DbContext в EF Core 8.

1. Объясни механизм работы ChangeTracker: как именно он определяет изменения (Snapshot vs Proxy)? В чем накладные расходы каждого подхода?
2. Сравни AddDbContext и AddDbContextPool. В каких сценариях пулинг контекста может навредить (например, сохранение состояния между запросами)?
3. Подготовь пример конфигурации Npgsql с использованием UseSnakeCaseNamingConvention и маппингом типов DateOnly/TimeOnly.
4. Объясни проблему 'Captive Dependency' при внедрении Scoped Context в Singleton Service и предложи решение через IServiceScopeFactory."

### 3.3. Практическое Задание: "Инициализация Инфраструктуры"

**Цель:** Создать изолированный слой доступа к данным (Infrastructure Layer) для сервиса Календаря, отделив его от API.

1. **Создание Проекта:** Развернуть решение с проектами Calendar.Domain (POCO классы), Calendar.Infrastructure (EF Core) и Calendar.Api.
2. **Доменная Модель:** Реализовать сущности:
   * User: Id (Guid), UserName, Email, Birthday (DateOnly - используем новую фичу.NET 8).11
   * Event: Id, Title, Description, StartAt, EndAt, UserId.
3. **Конфигурация EF:**
   * Подключить Npgsql.EntityFrameworkCore.PostgreSQL.
   * Реализовать ApplicationDbContext.
   * Настроить Connection String в appsettings.Development.json (использовать Docker Container для Postgres).
4. **Middleware Проверки:** Написать middleware или IHostedService, который при старте приложения проверяет доступность БД и применяет миграции (context.Database.MigrateAsync()). *Примечание: В продакшене миграции часто выносятся в отдельный шаг CI/CD, но для dev-среды автомиграция допустима.*

## 4. План Обучения: День 2 — Продвинутый Маппинг и JSONB

Unity-разработчики часто злоупотребляют сериализацией, сохраняя сложные графы объектов как строки. В Postgres для этого есть jsonb — бинарный JSON, который можно индексировать. EF Core 8 вывел работу с JSON на новый уровень.

### 4.1. Теоретический Блок

#### Тема 2.1: JSONB и Паттерн "Владеемые Типы" (Owned Types)

До EF Core 7/8 работа с JSON требовала ручной конвертации строк. Теперь мы можем мапить POCO-объекты напрямую в JSON-колонки.

* **ToJson():** Метод конфигурации, позволяющий хранить вложенные объекты или коллекции объектов в одной колонке jsonb, при этом сохраняя возможность LINQ-запросов к их свойствам.7
* **Индексация:** Разница между json и jsonb в PostgreSQL. Как создать GIN-индекс для ускорения поиска по ключам внутри JSON-документа.13

#### Тема 2.2: Массивы Примитивов (Arrays)

В отличие от SQL Server, PostgreSQL поддерживает массивы как тип данных.

* **Маппинг:** string или int в C# могут быть напрямую смапплены на колонки базы данных.
* **Запросы:** Использование LINQ оператора .Contains() для массивов транслируется в SQL оператор @> (contains), что радикально эффективнее JOIN-таблиц для простых списков тегов.15

#### Тема 2.3: Shadow Properties и Backing Fields

* **Инкапсуляция:** Использование приватных полей в сущностях и настройка EF Core для записи в них (Backing Fields). Это позволяет поддерживать DDD-подход, не открывая сеттеры свойств.
* **Shadow Properties:** Свойства, которые существуют в таблице БД (например, CreatedAt, LastModifiedBy), но не объявлены в C# классе сущности.

### 4.2. Промт для Глубокого Исследования (Теория)

**Запрос к AI:** "Исследуй возможности работы с JSON в EF Core 8 с провайдером Npgsql.

1. Сравни производительность и удобство использования: 'JSON Column Mapping (ToJson)' против 'Table-per-Type inheritance'. Когда стоит денормализовать данные в JSONB?
2. Напиши пример конфигурации Fluent API для маппинга свойства List<Tag> в колонку jsonb и свойства string Categories в PostgreSQL Array.
3. Объясни, как EF Core 8 транслирует LINQ-запрос Where(x => x.Settings.Theme == "Dark") в SQL, если Settings хранится как JSONB. Будет ли использоваться индекс?"

### 4.3. Практическое Задание: "Гибридная Схема Данных"

**Цель:** Расширить модель данных, используя специфические возможности Postgres для хранения неструктурированных данных.

1. **Настройки Пользователя:** Добавить в сущность User свойство UserSettings Settings.
   * UserSettings содержит: Theme (string), NotificationPreferences (сложный объект).
   * Сконфигурировать через Fluent API: builder.Entity<User>().OwnsOne(u => u.Settings, b => b.ToJson()).4
2. **Теги Событий:** Добавить в Event свойство string Tags.
   * Сконфигурировать его как массив PostgreSQL (column type: text).
3. **Seed Data:** Создать скрипт наполнения базы 1000 тестовыми событиями с разными тегами и настройками JSON.
4. **Анализ SQL:** Выполнить запрос, фильтрующий пользователей по свойству внутри JSON. Изучить сгенерированный SQL в логах (ожидается использование операторов ->> или @>).

## 5. План Обучения: День 3 — Производительность и Проблема N+1

Это самый важный день для трансформации мышления. В Unity цикл Update выполняется каждый кадр. В бэкенде "цикл по базе данных" — это катастрофа производительности.

### 5.1. Теоретический Блок

#### Тема 3.1: Проблема N+1 и Eager Loading

Классическая ошибка: загрузка списка родителей, а затем в цикле загрузка детей.

* **Диагностика:** Как обнаружить N+1? Использование логгирования EF Core и перехватчиков (Interceptors) для алертинга в dev-режиме.17
* **Include/ThenInclude:** Механизм жадной загрузки (Eager Loading).
* **Cartesian Explosion:** Проблема "взрыва" количества строк при множественных Include коллекций. Решение через AsSplitQuery(), разделяющее один гигантский JOIN на несколько SELECT запросов.5

#### Тема 3.2: Проекции (Projections) vs Tracking

Unity-разработчики привыкли работать с полными объектами. В бэкенде это избыточно.

* **Select:** Использование .Select(x => new Dto {... }). EF Core запрашивает *только* указанные колонки. Это самый быстрый способ чтения данных.20
* **AsNoTracking:** Отключение Change Tracker для Read-Only запросов. Существенная экономия памяти и CPU (не создаются снапшоты объектов).21

#### Тема 3.3: Компилируемые Запросы (Compiled Queries)

Для сверхвысоконагруженных путей (Hot Paths) парсинг LINQ-выражения может быть узким местом. Компилируемые запросы позволяют кэшировать дерево разбора запроса.21

### 5.2. Промт для Глубокого Исследования (Теория)

**Запрос к AI:** "Подготовь технический анализ проблемы N+1 в EF Core 8.

1. Приведи пример кода, вызывающего N+1, и покажи сгенерированный SQL лог.
2. Сравни производительность: Lazy Loading (Proxies) vs Eager Loading (Include) vs Explicit Loading vs Projections (Select).
3. Объясни концепцию 'Cartesian Explosion' при джойне нескольких коллекций. В каких случаях AsSplitQuery работает быстрее, а когда медленнее единого запроса?
4. Как настроить EF Core, чтобы он выбрасывал исключение при попытке навигации по свойству, которое не было загружено (предотвращение неявного Lazy Loading)?"

### 5.3. Практическое Задание: "Оптимизатор Запросов"

**Цель:** Научиться выявлять и устранять проблемы производительности LINQ.

1. **Плохой Код:** Реализовать метод API GET /users/report, который загружает всех пользователей (.ToList()), а затем в цикле foreach считает количество их событий, обращаясь к навигационному свойству.
   * Зафиксировать количество SQL-запросов (должно быть N+1).
2. **Оптимизация:** Переписать метод тремя способами и сравнить бенчмарки (время и память):
   * С использованием .Include(u => u.Events).
   * С использованием .AsSplitQuery().
   * С использованием проекции .Select(u => new { u.Name, EventCount = u.Events.Count }). *Ожидается, что этот вариант будет самым быстрым.*
3. **NoTracking:** Добавить AsNoTracking во все методы получения данных ("Get"), где не планируется изменение сущностей.

## 6. План Обучения: День 4 — Range Types и Конкурентность

В этот день мы реализуем функционал, который практически невозможно сделать надежно без базы данных: предотвращение овербукинга (двойного бронирования) и обработку гонки данных.

### 6.1. Теоретический Блок

#### Тема 4.1: Типы Диапазонов (Range Types) в PostgreSQL

Традиционный подход SQL проверяет пересечение интервалов через громоздкое условие (Start1 <= End2) AND (End1 >= Start2). Postgres предлагает элегантное решение.

* **tsrange/daterange:** Типы данных, хранящие диапазон. Npgsql мапит их на NpgsqlRange<DateTime>.2
* **Операторы:** && (пересечение), -|- (смежность), @> (вхождение). Это позволяет переложить логику проверки коллизий на ядро БД, что намного надежнее валидации в коде C#.

#### Тема 4.2: Ограничения Исключения (Exclusion Constraints)

Это "секретное оружие" Postgres для календарей.

* **Механизм:** Позволяет создать ограничение уровня таблицы: "Запретить вставку строки, если её временной диапазон пересекается (&&) с существующей строкой для того же ресурса (=), используя GiST индекс".24
* **Реализация в EF:** EF Core не имеет Fluent API для этого, поэтому используется Raw SQL в миграциях.

#### Тема 4.3: Оптимистичная Конкурентность (Optimistic Concurrency)

Сценарий: Два администратора одновременно открыли редактирование одного события. Первый сохранил. Второй сохранил, перезаписав изменения первого.

* **xmin:** Системная колонка в Postgres, которая меняется при каждом обновлении строки. EF Core может использовать её как токен конкурентности (`` аналог).6
* **Обработка Конфликтов:** Перехват DbUpdateConcurrencyException. Стратегии: "Client Wins" (перезаписать) или "Store Wins" (сообщить пользователю и обновить данные).

### 6.2. Промт для Глубокого Исследования (Теория)

**Запрос к AI:** "Разработай архитектуру защиты от Overbooking (двойного бронирования) на уровне PostgreSQL.

1. Объясни работу типа данных tsrange и индекса GiST.
2. Напиши SQL-команду для создания EXCLUDE ограничения, которое запрещает пересекающиеся интервалы времени для одного RoomId.
3. Как внедрить этот SQL в миграцию EF Core?
4. Объясни механизм оптимистичной блокировки через системную колонку xmin в PostgreSQL. Почему это эффективнее, чем ручное создание колонки Version? Как сконфигурировать это в EF Core Fluent API?"

### 6.3. Практическое Задание: "Атомарное Бронирование"

**Цель:** Реализовать надежную систему бронирования, устойчивую к гонкам.

1. **Refactoring:** Заменить поля StartAt и EndAt в Event на одно поле Duration типа NpgsqlRange<DateTime>.
2. **Exclusion Constraint:**
   * Создать пустую миграцию.
   * В методе Up прописать SQL:  
     SQL  
     ALTER TABLE "Events"  
     ADD CONSTRAINT "NoOverlappingEvents"  
     EXCLUDE USING GIST ("UserId" WITH =, "Duration" WITH &&);
   * Попытаться программно создать два пересекающихся события для одного юзера. Убедиться, что выбрасывается DbUpdateException (PostgresException: 23P01).
3. **Concurrency Token:**
   * Добавить в Event свойство uint Version и настроить маппинг на xmin через .IsRowVersion().27
   * Написать тест-кейс: загрузить событие в двух разных контекстах, изменить в обоих, сохранить первый, затем попытаться сохранить второй. Обработать исключение.

## 7. План Обучения: День 5 — Паттерны, Dapper и Тестирование

Завершающий день посвящен интеграции полученных знаний в архитектуру приложения и проверке качества.

### 7.1. Теоретический Блок

#### Тема 5.1: Репозиторий vs DbContext

Вечный холивар в мире.NET.

* **Аргументы против Repository:** DbContext и DbSet уже реализуют Unit of Work и Repository. Создание оберток лишает доступа к фичам EF (Include, SplitQuery) и усложняет код.8
* **Аргументы за:** Абстракция для Unit-тестов (Mocks).
* **Современный подход:** Использование "Vertical Slice Architecture" или CQRS, где запросы инкапсулируются в хендлерах, а не в репозиториях. Прямое использование DbContext предпочтительно для EF Core.

#### Тема 5.2: Гибридный подход (EF Core + Dapper)

Иногда ORM слишком медленный или генерирует неоптимальный SQL для сложных аналитических отчетов.

* **Dapper:** Микро-ORM. Работает с Raw SQL.
* **Симбиоз:** Использование context.Database.GetDbConnection() позволяет Dapper использовать ту же транзакцию, что и EF Core, обеспечивая согласованность.30

#### Тема 5.3: Интеграционное Тестирование с Testcontainers

Unit-тесты с моками DbSet бесполезны для проверки SQL-запросов. InMemory провайдер ведет себя иначе, чем Postgres (не поддерживает JSONB, транзакции).

* **Testcontainers:** Библиотека для запуска реального Postgres в Docker из кода тестов. Это золотой стандарт тестирования БД.32
* **Respawn:** Утилита для быстрой очистки таблиц между тестами без пересоздания схемы.

### 5.2. Промт для Глубокого Исследования (Теория)

**Запрос к AI:** "Сравни подходы 'Repository Pattern' и 'Direct DbContext' в 2024/2025 годах.

1. Почему сообщество.NET все чаще отказывается от репозиториев поверх EF Core? В чем риски абстракции IQueryable?
2. Как настроить совместную работу EF Core и Dapper в рамках одной транзакции?
3. Составь руководство по настройке Testcontainers для PostgreSQL в xUnit. Как оптимизировать время запуска контейнера (Singelton container lifetime)?"

### 5.3. Практическое Задание: "Качество и Надежность"

**Цель:** Покрыть критический функционал тестами на реальной БД.

1. **Dapper Report:** Реализовать метод получения статистики (например, топ-5 тегов за месяц) через Dapper, используя SQL-запрос с агрегацией.
2. **Integration Test Setup:** Настроить Testcontainers для xUnit.
3. **Тестирование Constraint:** Написать интеграционный тест, который проверяет, что база данных действительно отклоняет пересекающиеся интервалы (проверка работы Exclusion Constraint из Дня 4). Мок-объекты здесь бы не сработали.

## 8. Итоговые Задания Недели (Capstone Projects)

Эти задания требуют синтеза всех тем недели: моделирования, оптимизации запросов, использования специфики Postgres и управления конкурентностью.

### Проект А: "Система Бронирования Переговорных (Room Booker)"

Бизнес-задача: Реализовать backend для бронирования переговорных комнат в офисе.

Технические Требования:

1. **Сущность Room:** Имеет Schedule (расписание доступности), хранящееся как jsonb (например, {"Mon": ["09:00-18:00"],...}).
2. **Поиск Слотов:** Реализовать алгоритм поиска свободных слотов. Принимает желаемую длительность и диапазон дат. Должен использовать SQL-функции Postgres для работы с tsrange (пересечение желаемого времени с существующими бронями).
   * *Challenge:* Реализовать это одним эффективным LINQ или SQL запросом, исключая выборку всех броней в память.
3. **Strict Consistency:** Использовать Exclusion Constraint для гарантии того, что комната не будет забронирована дважды.
4. **Soft Delete:** Реализовать "мягкое удаление" броней (флаг IsDeleted), используя Global Query Filters в EF Core. Убедиться, что Exclusion Constraint корректно игнорирует удаленные записи (добавить условие WHERE).

### Проект Б: "Аналитический Дашборд Событий"

Бизнес-задача: API для отображения статистики использования календаря.

Технические Требования:

1. **Hybrid Mapping:** События имеют поле Metadata (jsonb), куда сторонние сервисы пишут произвольные данные (например, { "JiraId": 123, "Priority": "High" }).
2. **Smart Filter:** Реализовать метод поиска, который принимает словарь фильтров. Он должен искать как по колонкам таблицы, так и внутри JSONB (например, найти все события, где Metadata->>'Priority' == 'High').
   * Использовать EF.Functions.JsonContains или аналоги.
3. **Performance:** Создать GIN индекс на поле Metadata. Сравнить план выполнения запроса (EXPLAIN ANALYZE) с индексом и без него.
4. **Read-Only Optimization:** Все запросы дашборда должны выполняться с AsNoTracking и, где возможно, использовать Dapper для агрегации данных.

### Заключение

К концу этой недели вы не просто научитесь "сохранять данные". Вы овладеете инструментарием архитектора баз данных, способного проектировать системы, устойчивые к высоким нагрузкам и конкурентному доступу. Вы поймете, почему перенос логики целостности данных из кода C# (где возможны гонки) на уровень ограничений PostgreSQL (где гарантируется ACID) является признаком зрелого Enterprise-решения.

#### Источники

1. Разработка бэкенда: план обучения C#
2. Ranges and Multiranges | Npgsql Documentation, дата последнего обращения: декабря 5, 2025, <https://www.npgsql.org/efcore/mapping/range.html>
3. Documentation: 18: 8.17. Range Types - PostgreSQL, дата последнего обращения: декабря 5, 2025, <https://www.postgresql.org/docs/current/rangetypes.html>
4. What's New in EF Core 8 | Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-8.0/whatsnew>
5. Efficient Querying - EF Core - Microsoft Learn, дата последнего обращения: декабря 5, 2025, <https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying>
6. Concurrency Tokens | Npgsql Documentation, дата последнего обращения: декабря 5, 2025, <https://www.npgsql.org/efcore/modeling/concurrency.html>
7. JSON Mapping | Npgsql Documentation, дата последнего обращения: декабря 5, 2025, <https://www.npgsql.org/efcore/mapping/json.html>
8. Repository Pattern vs Direct EF Core: The Great Debate | by Adrian Bailador - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@adrianbailador/repository-pattern-vs-direct-ef-core-the-great-debate-3c914348b7e0>
9. 10 Essential Best Practices for Using Entity Framework Core in .NET 8 - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@solomongetachew112/10-essential-best-practices-for-using-entity-framework-core-in-net-8-3274d6143992>
10. Use DateOnly and TimeOnly with EF Core 6, 7 & 8 and Azure SQL / SQL Server, дата последнего обращения: декабря 5, 2025, <https://erikej.github.io/efcore/sqlserver/2023/09/03/efcore-dateonly-timeonly.html>
11. Persisting Date Only in PostgreSQL Database - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/78187437/persisting-date-only-in-postgresql-database>
12. Using JSON data types with .NET EF Core - DEV Community, дата последнего обращения: декабря 5, 2025, <https://dev.to/gaatif/net-ef-core-postgresql-and-json-data-types-362o>
13. JSONB in PostgreSQL with EF Core — Part 2 | by Serhii Kokhan - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@serhiikokhan/jsonb-in-postgresql-with-ef-core-part-2-7cc29b83b43b>
14. PostgreSQL JSONB in .NET - Marek Sirkovský - Medium, дата последнего обращения: декабря 5, 2025, <https://mareks-082.medium.com/postgresql-jsonb-in-net-25fbcc7b64b2>
15. A beginner's guide to mapping arrays in EF Core 8 - .NET Blog, дата последнего обращения: декабря 5, 2025, <https://devblogs.microsoft.com/dotnet/array-mapping-in-ef-core-8/>
16. Supported Types and their Mappings | Npgsql Documentation, дата последнего обращения: декабря 5, 2025, <https://www.npgsql.org/doc/types/basic.html>
17. Entity Framework Core 7: N+1 Queries Problem - Thinktecture AG, дата последнего обращения: декабря 5, 2025, <https://www.thinktecture.com/en/entity-framework-core/entity-framework-core7-n1-queries-problem/>
18. N+1 Query Problem with a custom EF Core detector! - YouTube, дата последнего обращения: декабря 5, 2025, <https://www.youtube.com/watch?v=8cH0mUdaV_s>
19. Best Way to Handle Complex Queries with Multiple Includes in EF Core 8? - Reddit, дата последнего обращения: декабря 5, 2025, <https://www.reddit.com/r/dotnet/comments/1if3ei8/best_way_to_handle_complex_queries_with_multiple/>
20. Essential Entity Framework Core Tips: How to Optimize Performance, Streamline Queries, and More | HackerNoon, дата последнего обращения: декабря 5, 2025, <https://hackernoon.com/essential-entity-framework-core-tips-how-to-optimize-performance-streamline-queries-and-more>
21. 10 Performance Optimization Tips for .NET 8 and EF Core | by Nahid Al Fuad - Medium, дата последнего обращения: декабря 5, 2025, <https://medium.com/@fuad.cse27/10-performance-optimization-tips-for-net-8-and-ef-core-993befc975c4>
22. What are the Best Practices for EF Core Performance Optimization? - C# Corner, дата последнего обращения: декабря 5, 2025, <https://www.c-sharpcorner.com/article/what-are-the-best-practices-for-ef-core-performance-optimization/>
23. Entity Framework Core - Deep Performance Optimization Guide - C# Corner, дата последнего обращения: декабря 5, 2025, <https://www.c-sharpcorner.com/article/entity-framework-core-deep-performance-optimization-guide/>
24. PostgreSQL range types and Entity Framework Core | Giorgi Dalakishvili | Personal Website, дата последнего обращения: декабря 5, 2025, <https://www.giorgi.dev/database/postgresql-range-types-entity-framework/>
25. Preventing adjacent/overlapping entries with EXCLUDE in PostgreSQL - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/19504727/preventing-adjacent-overlapping-entries-with-exclude-in-postgresql>
26. Preventing Overlapping Data in PostgreSQL - What Goes Into an Exclusion Constraint, дата последнего обращения: декабря 5, 2025, <https://blog.danielclayton.co.uk/posts/overlapping-data-postgres-exclusion-constraints/>
27. Npgsql EF Core concurrency token property gets included in migrations - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/78974688/npgsql-ef-core-concurrency-token-property-gets-included-in-migrations>
28. Repository implemented in EF Core: use whole DbContext or only DbSet? - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/70645188/repository-implemented-in-ef-core-use-whole-dbcontext-or-only-dbset>
29. What value do you gain from using the Repository Pattern when using EF Core? - Reddit, дата последнего обращения: декабря 5, 2025, <https://www.reddit.com/r/dotnet/comments/1loxmzr/what_value_do_you_gain_from_using_the_repository/>
30. Dapper vs. EF Core: Performance Showdown in 2025 - DEV Community, дата последнего обращения: декабря 5, 2025, <https://dev.to/dev_saeid_ghaderi/dapper-vs-ef-core-performance-showdown-in-2025-4j32>
31. EF Core vs. Dapper : r/dotnet - Reddit, дата последнего обращения: декабря 5, 2025, <https://www.reddit.com/r/dotnet/comments/1fmnhgr/ef_core_vs_dapper/>
32. c# - Entity Framework Core jsonb column type - Stack Overflow, дата последнего обращения: декабря 5, 2025, <https://stackoverflow.com/questions/43089058/entity-framework-core-jsonb-column-type>