# Архитектурный анализ темпоральных данных и конкурентного доступа в среде Enterprise.NET: Глубокое погружение в День 4 Недели 8

## 1. Введение: Сдвиг парадигмы от Stateful к Stateless архитектуре

Переход квалифицированного разработчика Unity в сферу корпоративной backend-разработки на платформе.NET 8 знаменует собой не просто смену синтаксиса или инструментов, но фундаментальный сдвиг в ментальной модели управления состоянием приложения. В контексте плана обучения, представленного в документе 1, восьмая неделя является критической точкой бифуркации, где происходит отказ от управления данными в оперативной памяти (Memory Thinking) в пользу управления данными в персистентном хранилище (Database Thinking).

Четвертый день восьмой недели посвящен двум наиболее сложным аспектам распределенных систем: управлению временем (темпоральные данные) и управлению параллельным доступом (конкурентность). В игровой индустрии (GameDev) состояние мира часто является детерминированным в рамках одного кадра: физический движок проверяет коллизии объектов в памяти, и разработчик имеет полный контроль над последовательностью обновлений в цикле Update(). В Enterprise-системах, таких как проектируемый сервис «Календарь», тысячи транзакций могут конкурировать за одну и ту же запись одновременно. Здесь нет единого «главного цикла»; есть лишь база данных, выступающая в роли арбитра истины (Source of Truth).

Этот документ представляет собой исчерпывающее теоретическое и практическое руководство по темам четвертого дня, охватывающее математические основы интервальной алгебры, внутреннее устройство индексов PostgreSQL GiST, механику многоверсионного управления конкурентностью (MVCC) и интеграцию этих концепций в экосистему Entity Framework Core 8.

## 2. Теория темпоральных данных и Range Types в PostgreSQL

Традиционные подходы к хранению временных интервалов в реляционных базах данных исторически страдали от семантической неполноты. Стандартный паттерн проектирования подразумевает создание двух скалярных колонок: start\_at (начало) и end\_at (конец). Однако этот подход, называемый «скалярной моделью», перекладывает всю сложность валидации и вычислений на прикладной уровень.

### 2.1. Ограничения скалярной модели и Алгебра Аллена

В скалярной модели проверка пересечения двух событий требует написания громоздких предикатов. Рассмотрим задачу обнаружения коллизии (overlap) между запрашиваемым интервалом $(S\_1, E\_1)$ и существующим интервалом $(S\_2, E\_2)$. Логическое условие выглядит следующим образом:

$$(S\_1 \le E\_2) \land (E\_1 \ge S\_2)$$

Эта формула, хотя и корректна, обладает рядом недостатков в контексте Enterprise-разработки:

1. **Когнитивная нагрузка:** Разработчик должен помнить и правильно воспроизводить эту логику в каждом SQL-запросе или LINQ-выражении. Ошибка в знаке неравенства (например, < вместо <=) приводит к трудноуловимым багам на границах интервалов.2
2. **Отсутствие инкапсуляции:** Семантика «интервала» размыта между двумя независимыми колонками. База данных не «знает», что эти колонки связаны, что позволяет случайно вставить данные, где start\_at > end\_at, если не добавить отдельное ограничение CHECK.
3. **Сложность индексации:** B-Tree индексы, стандартные для реляционных СУБД, эффективно работают с точечными запросами или диапазонами по одной колонке. Индексация пересечений по двум колонкам одновременно требует сложных составных индексов или использования R-Tree структур, которые не поддерживаются для скалярных пар во многих СУБД.1

Для решения этих проблем PostgreSQL ввел нативные типы диапазонов (Range Types), основанные на интервальной алгебре Джеймса Аллена. Алгебра Аллена определяет 13 базовых отношений между временными интервалами (равенство, перекрытие, вхождение, смежность и т.д.), которые PostgreSQL реализует через специализированные операторы.3

### 2.2. Математика диапазонов и канонические формы

Диапазон в PostgreSQL — это единый тип данных, представляющий множество значений между верхней и нижней границами. Критически важным аспектом является управление включением границ (inclusivity).

#### 2.2.1. Виды границ

В математической нотации используются квадратные скобки `` для включения границы и круглые () для исключения.

* [a, b]: Закрытый интервал (включает $a$ и $b$).
* [a, b): Полуоткрытый интервал (включает $a$, исключает $b$).
* (a, b]: Полуоткрытый интервал (исключает $a$, включает $b$).
* (a, b): Открытый интервал (исключает $a$ и $b$).

#### 2.2.2. Каноническая форма (Canonical Form)

Для дискретных типов данных (например, integer или date) PostgreSQL автоматически нормализует диапазоны к единой канонической форме: [ ) (включая нижнюю, исключая верхнюю).

Например, для целочисленного диапазона int4range:

* Ввод: `` (числа 1, 2, 3, 4, 5).
* Хранение: [1, 6) (числа $x$, где $1 \le x < 6$).  
  Это упрощает внутреннюю логику сравнения, так как базе данных не нужно обрабатывать 4 варианта скобок для каждого сравнения — все приводится к одному виду.4

Для непрерывных типов, таких как tsrange (timestamp without time zone) или numrange (numeric), дискретизация невозможна, поэтому PostgreSQL сохраняет указанные пользователем границы. Однако в календарных системах стандартом де-факто является форма [start, end), так как она позволяет идеально стыковать события: конец одной встречи ($14:00$) математически равен началу следующей ($14:00$), но при этом они не пересекаются, так как $14:00$ исключено из первого интервала.6

**Таблица 1: Сравнительный анализ типов диапазонов в PostgreSQL**

| **Тип PostgreSQL** | **Описание** | **.NET Тип (Npgsql)** | **Особенности использования** |
| --- | --- | --- | --- |
| int4range | Диапазон integer | NpgsqlRange<int> | Идентификаторы версий, номера страниц. Дискретный тип. |
| int8range | Диапазон bigint | NpgsqlRange<long> | Большие идентификаторы. Дискретный тип. |
| numrange | Диапазон numeric | NpgsqlRange<decimal> | Финансовые диапазоны, цены. Непрерывный тип. |
| tsrange | Timestamp (без зоны) | NpgsqlRange<DateTime> | Локальное планирование. Требует осторожности с часовыми поясами. |
| tstzrange | Timestamp (с зоной) | NpgsqlRange<DateTime> | Глобальное планирование. DateTime должен быть UTC (Kind=Utc). |
| daterange | Диапазон дат | NpgsqlRange<DateOnly> | Отпуска, бронирование отелей. Дискретный шаг в 1 день. |

### 2.3. Обработка бесконечности (Infinity) и NULL

В отличие от C#, где DateTime.MinValue и DateTime.MaxValue являются конкретными числовыми значениями (тиками), PostgreSQL поддерживает концепцию истинной математической бесконечности.

* -infinity: Значение меньше любого другого возможного значения.
* infinity: Значение больше любого другого возможного значения.

При проектировании системы бронирования это критически важно. Подписка, которая «действует с сегодняшнего дня и навсегда», должна храниться как `

Важный нюанс: в C# default(DateTime) — это 0001-01-01, что является валидной датой, а не бесконечностью. При маппинге необходимо явно указывать драйверу, что граница является бесконечной, используя свойство LowerBoundInfinite или UpperBoundInfinite.7

## 3. Инфраструктура индексации: GiST и btree\_gist

Использование типов диапазонов само по себе дает лишь семантическое преимущество. Производительность обеспечивается правильной стратегией индексации. В Unity разработчик привык к пространственному хешированию или Octree для поиска объектов. В PostgreSQL аналогом этих структур является GiST.

### 3.1. Внутреннее устройство GiST (Generalized Search Tree)

GiST — это не конкретная структура данных, а фреймворк для построения сбалансированных древовидных структур доступа. Для типов диапазонов GiST реализует логику R-Tree (R-дерева).

* **Принцип работы:** В отличие от B-Tree, которое упорядочивает данные в одном измерении (меньше/больше), R-Tree организует данные в иерархию вложенных «ограничивающих прямоугольников» (Bounding Boxes) или интервалов.
* **Дерево:** Каждый узел дерева содержит минимальный интервал, охватывающий все интервалы в его дочерних узлах.
* **Поиск:** При выполнении запроса «найти все встречи, пересекающиеся с 12:00-13:00», поисковый алгоритм спускается по дереву. Если ограничивающий интервал узла не пересекается с целевым диапазоном, все поддерево отсекается (pruning). Это позволяет выполнять поиск коллизий за логарифмическое время $O(\log N)$ вместо линейного сканирования $O(N)$.9

### 3.2. Функция Penalty и алгоритмы разделения (Split Algorithms)

Эффективность GiST индекса зависит от того, насколько плотно сгруппированы данные. При вставке новой записи GiST должен выбрать, в какой узел её поместить. Для этого используется функция Penalty. Она вычисляет, насколько придется «расширить» границы существующего узла, чтобы включить новый элемент. Алгоритм выбирает путь с минимальным «штрафом» (минимальным расширением), что сохраняет компактность дерева.9

В PostgreSQL реализация GiST для диапазонов использует квадратичный алгоритм разделения Гуттмана (Guttman's quadratic split algorithm), который оптимизирует покрытие пространства, минимизируя перекрытия между узлами одного уровня.12 Это критически важно понимать архитектору: вставка в GiST индекс дороже, чем в B-Tree, из-за сложных геометрических вычислений, но это плата за возможность мгновенного поиска пересечений.

### 3.3. Роль расширения btree\_gist

В реальных системах бронирования мы редко ищем пересечения по всей таблице. Обычно запрос звучит как: «Найти пересечения для Комнаты №101 в заданный интервал».

Здесь мы имеем смешанный запрос:

1. Равенство по скалярному типу (integer RoomId).
2. Пересечение по диапазонному типу (tsrange Duration).

Стандартный GiST индекс не умеет индексировать integer (для этого есть B-Tree). Чтобы создать единый составной индекс (RoomId, Duration), необходимо научить GiST работать со скалярными типами как с вырожденными диапазонами.

Именно эту задачу решает расширение btree\_gist. Оно добавляет классы операторов GiST для стандартных типов данных (int, text, uuid), позволяя комбинировать их с диапазонами в одном индексе.14 Без этого расширения создание Exclusion Constraint, включающего ID ресурса, невозможно.

## 4. Ограничения исключения (Exclusion Constraints): Гарантия целостности

Четвертый день обучения вводит концепцию, которая является аналогом физических коллайдеров в Unity, но на уровне данных. Exclusion Constraint — это механизм, который гарантирует, что в таблице не существует двух строк, для которых заданный предикат возвращает TRUE.

### 4.1. Проблема состояния гонки (Race Condition)

Рассмотрим классический сценарий ошибки, которую допускают разработчики, мыслящие в парадигме памяти (Memory Thinking):

C#

// Плохой паттерн: Check-Then-Act  
var existing = await \_context.Events  
 .Where(e => e.RoomId == roomId && e.Duration.Intersects(newDuration))  
 .ToListAsync();  
  
if (existing.Count == 0) {  
 \_context.Events.Add(newEvent); // RACE CONDITION HERE  
 await \_context.SaveChangesAsync();  
}

В многопоточной среде (или при запуске нескольких экземпляров сервиса) две параллельные транзакции могут выполнить проверку Where одновременно, обе получат 0 записей, и обе выполнят вставку. Результат — двойное бронирование (Overbooking).

### 4.2. Декларативная защита: EXCLUDE USING GIST

Решение заключается в переносе ответственности за валидацию с уровня приложения (C#) на уровень базы данных (PostgreSQL). Мы объявляем правило: «Запретить вставку, если ID комнаты совпадает (=) **И** время перекрывается (&&)».

SQL-синтаксис создания такого ограничения:

SQL

ALTER TABLE "Events"  
ADD CONSTRAINT "NoOverlappingEvents"  
EXCLUDE USING GIST (  
 "RoomId" WITH =,  
 "Duration" WITH &&  
);

.16

Этот DDL-запрос создает скрытый GiST индекс. Перед фиксацией любой транзакции PostgreSQL проверяет этот индекс. Если обнаружен конфликт, транзакция откатывается с ошибкой. Это гарантирует атомарность проверки и вставки.

### 4.3. Взаимодействие с Soft Deletes

В корпоративных системах удаление данных часто реализуется через флаг IsDeleted (Soft Delete), чтобы сохранить историю. Это создает проблему для Exclusion Constraint: если мы «удаляем» встречу, устанавливая IsDeleted = true, она все равно остается в таблице и продолжает блокировать временной слот, так как ограничение проверяет физическое наличие строки.

Для решения этой проблемы PostgreSQL позволяет добавлять условие WHERE к ограничению исключения (аналогично частичным индексам):

SQL

ALTER TABLE "Events"  
ADD CONSTRAINT "NoOverlappingEvents"  
EXCLUDE USING GIST (  
 "RoomId" WITH =,  
 "Duration" WITH &&  
)  
WHERE ("IsDeleted" = FALSE);

.18

Теперь «удаленные» записи игнорируются индексом и не вызывают конфликтов при бронировании того же времени. Это изящное решение недоступно во многих других СУБД (например, SQL Server требует сложных обходных путей с вычисляемыми колонками для реализации подобной логики).20

## 5. Многоверсионное управление конкурентностью (MVCC) и xmin

Если Exclusion Constraints защищают от *логических* коллизий (бизнес-правила), то механизмы конкурентного доступа защищают от *технических* коллизий (перезапись данных).

### 5.1. Аномалия «Потерянное обновление» (Lost Update)

В Unity данные обычно изменяются одним потоком. В веб-сервисе возможен сценарий:

1. **Пользователь А** загружает форму редактирования события (Версия 1).
2. **Пользователь Б** загружает ту же форму (Версия 1).
3. **Пользователь А** меняет Заголовок и сохраняет. В базе теперь Версия 2.
4. **Пользователь Б** меняет Описание и сохраняет. Он отправляет данные, основанные на Версии 1.

Если система просто выполнит UPDATE, изменения Пользователя А будут перезаписаны старыми данными из формы Пользователя Б. Изменение заголовка будет потеряно.

### 5.2. MVCC в PostgreSQL

PostgreSQL использует архитектуру MVCC. При обновлении строки старая версия не удаляется и не перезаписывается на месте. Вместо этого создается новая версия строки (tuple).

Каждая строка имеет скрытые системные поля:

* xmin: ID транзакции, которая создала эту версию строки.
* xmax: ID транзакции, которая удалила эту версию (или 0, если жива).

xmin является идеальным кандидатом для токена оптимистичной блокировки. Он автоматически изменяется при каждом обновлении строки, так как каждое обновление происходит в новой транзакции с новым ID.21

### 5.3. Проблема Wraparound (Переполнение счетчика транзакций)

Transaction ID в PostgreSQL — это 32-битное целое число (около 4 миллиардов значений). В высоконагруженных системах этот счетчик может переполниться. PostgreSQL использует механизм "Vacuum Freeze" для обработки старых транзакций, чтобы предотвратить потерю данных (Transaction ID Wraparound).

Однако для целей оптимистичной блокировки (проверка WHERE xmin = @original\_xmin) риск коллизии из-за переполнения ничтожно мал. Вероятность того, что строка будет прочитана, затем пройдет 4 миллиарда транзакций, и только потом пользователь попытается её сохранить, и при этом новый xmin совпадет со старым — статистически невозможна в рамках одной сессии редактирования.23

## 6. Интеграция с.NET 8 и EF Core 8

Теперь рассмотрим, как эти теоретические концепции реализуются на практике с использованием современного стека.NET.

### 6.1. Маппинг Range Types

Библиотека Npgsql предоставляет обобщенный тип NpgsqlRange<T>, который полностью поддерживает все нюансы диапазонов PostgreSQL.

#### 6.1.1. Конфигурация сущности

C#

public class Booking  
{  
 public int Id { get; set; }  
 // Использование DateOnly (новая фича.NET 8/6)  
 public NpgsqlRange<DateOnly> Period { get; set; }   
 // Для точного времени используем DateTime  
 public NpgsqlRange<DateTime> Duration { get; set; }  
}

#### 6.1.2. Нюансы маппинга

* **DateOnly:** В Npgsql 8.0 DateOnly мапится на тип date напрямую. NpgsqlRange<DateOnly> автоматически мапится на daterange.25
* **DateTime и TimeZone:** По умолчанию DateTime мапится на timestamp without time zone. Если требуется tstzrange (с часовым поясом), необходимо убедиться, что DateTime.Kind установлен в Utc, и настроить Npgsql на использование timestamp with time zone.
* **NodaTime:** Для наиболее корректной работы с временем рекомендуется использовать плагин Npgsql.EntityFrameworkCore.PostgreSQL.NodaTime. Тип Interval из NodaTime семантически ближе к tsrange, чем NpgsqlRange<DateTime>, и позволяет избежать многих ошибок с часовыми поясами.27

### 6.2. Реализация Exclusion Constraints через Миграции

EF Core 8 не имеет Fluent API для создания Exclusion Constraints. Это требует использования Raw SQL внутри миграций.

Шаг 1: Создание пустой миграции

dotnet ef migrations add AddBookingConstraint

**Шаг 2: Реализация Up/Down методов**

C#

protected override void Up(MigrationBuilder migrationBuilder)  
{  
 // Обязательно: активация расширения для поддержки скалярных типов в GiST  
 migrationBuilder.Sql("CREATE EXTENSION IF NOT EXISTS btree\_gist;");  
  
 migrationBuilder.Sql(@"  
 ALTER TABLE ""Bookings""  
 ADD CONSTRAINT ""NoOverlappingBookings""  
 EXCLUDE USING GIST (  
 ""RoomId"" WITH =,  
 ""Duration"" WITH &&  
 )  
 WHERE (""IsDeleted"" = FALSE);  
 ");  
}  
  
protected override void Down(MigrationBuilder migrationBuilder)  
{  
 migrationBuilder.Sql(@"ALTER TABLE ""Bookings"" DROP CONSTRAINT ""NoOverlappingBookings"";");  
}

.19

Важно отметить, что попытка создать ограничение без btree\_gist приведет к ошибке data type integer has no default operator class for access method "gist", так как RoomId является целым числом.30

### 6.3. Настройка Оптимистичной Конкурентности (xmin)

В EF Core 8 настройка xmin как токена конкурентности стала стандартизированной. Ранее использовался специфичный метод UseXminAsConcurrencyToken, но теперь он помечен как Obsolete.

**Fluent API конфигурация:**

C#

protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
 modelBuilder.Entity<Booking>()  
 .Property(b => b.Version)  
 .IsRowVersion(); // Автоматически мапится на xmin в Npgsql  
}

**Сущность:**

C#

public class Booking  
{  
 public int Id { get; set; }  
 // Тип должен быть uint, так как xmin - 32-битный беззнаковый  
   
 public uint Version { get; set; }   
}

.21

Критический момент при миграциях:

EF Core может попытаться создать физическую колонку Version или xmin в таблице при генерации миграции. Поскольку xmin — это системная колонка, которая уже существует, добавление её в CreateTable или AddColumn приведет к ошибке.

Необходимо вручную удалить команды создания колонки Version/xmin из сгенерированного файла миграции, оставив только конфигурацию модели.32

## 7. Обработка исключений и паттерны проектирования

Внедрение жестких ограничений в БД требует изменения подхода к обработке ошибок в коде.

### 7.1. Обработка коллизий (Exclusion Violation)

Когда база данных отвергает вставку из-за пересечения интервалов, Npgsql выбрасывает исключение PostgresException с кодом ошибки 23P01.

**Паттерн перехвата:**

C#

try  
{  
 await \_context.SaveChangesAsync();  
}  
catch (DbUpdateException ex)  
 when (ex.InnerException is PostgresException pgEx && pgEx.SqlState == "23P01")  
{  
 // Ошибка 23P01: exclusion\_violation  
 throw new DomainException("Выбранный временной слот уже занят.");  
}

.33

Это позволяет трансформировать техническую ошибку БД в понятную пользователю бизнес-ошибку без предварительных проверок (SELECT).

### 7.2. Обработка конкуренции (Concurrency Conflict)

При несовпадении xmin EF Core выбрасывает DbUpdateConcurrencyException.

Стратегия "Store Wins" (Магазин выигрывает):

Мы уведомляем пользователя, что данные изменились, и просим обновить страницу.

C#

catch (DbUpdateConcurrencyException)  
{  
 // Отмена транзакции и уведомление  
 return Conflict(new { message = "Данные были изменены другим пользователем. Пожалуйста, обновите страницу." });  
}

Стратегия "Client Wins" (Клиент выигрывает):

Мы принудительно перезаписываем данные, обновляя токен. В EF Core это требует явной перезагрузки сущности и наложения значений клиента.35 Для календарных систем эта стратегия опасна и не рекомендуется.

## 8. Заключение

Четвертый день восьмой недели обучения предоставляет разработчику инструменты для построения систем с гарантированной целостностью данных.

1. **PostgreSQL Range Types** заменяют хрупкую логику сравнения границ Start/End на мощную алгебру интервалов.
2. **GiST индексы** обеспечивают производительность поиска пересечений, недостижимую для B-Tree.
3. **Exclusion Constraints** переносят гарантии уникальности времени с уровня приложения (где возможны гонки) на уровень ACID-транзакций базы данных.
4. **Optimistic Concurrency (xmin)** защищает от потери данных при совместной работе без накладных расходов на блокировки.

Освоение этих паттернов трансформирует мышление разработчика: вместо императивной проверки состояния («проверь, потом запиши») используется декларативное определение ограничений («запиши, если это не нарушает инварианты»), что является признаком зрелой архитектуры высоконагруженных систем.

**Таблица 2: Сводная таблица ключевых технологий Дня 4**

| **Концепция** | **Реализация в PostgreSQL** | **Реализация в.NET 8** | **Основная цель** |
| --- | --- | --- | --- |
| **Интервал** | tsrange, daterange | NpgsqlRange<T> | Атомарное хранение периода времени. |
| **Поиск** | Индекс GiST | LINQ: .Where(x => x.Range.Overlaps(y)) | Быстрый поиск пересечений ($O(\log N)$). |
| **Валидация** | EXCLUDE USING GIST | Raw SQL в миграции | Предотвращение овербукинга (Race Condition). |
| **Конкуренция** | Колонка xmin | uint Version | Предотвращение Lost Update. |
| **Расширение** | btree\_gist | CREATE EXTENSION | Поддержка скалярных ключей в GiST. |

Источники:

1 План обучения общий неделя 8.

3 Документация PostgreSQL: Range Types.

25 Документация Npgsql: Маппинг типов.

16 Exclusion Constraints и Soft Deletes.

21 Concurrency Tokens и xmin.

14 btree\_gist extension.

33 Обработка ошибок PostgresException.

#### Источники

1. План обучения общий неделя 8
2. Beyond Start and End: PostgreSQL Range Types - boringSQL, дата последнего обращения: декабря 6, 2025, <https://boringsql.com/posts/beyond-start-end-columns/>
3. Documentation: 18: 8.17. Range Types - PostgreSQL, дата последнего обращения: декабря 6, 2025, <https://www.postgresql.org/docs/current/rangetypes.html>
4. Canonical Form of PostgreSQL DateRange Columns - Hashrocket, дата последнего обращения: декабря 6, 2025, <https://hashrocket.com/blog/posts/canonical-form-of-postgresql-daterange-columns>
5. This is the effect of canonicalization in PostgreSQL - HEY World, дата последнего обращения: декабря 6, 2025, <https://world.hey.com/dinom/this-is-the-effect-of-canonicalization-in-postgresql-3ed5507c>
6. PostgreSQL DateRange and Efficient Time Management - Hashrocket, дата последнего обращения: декабря 6, 2025, <https://hashrocket.com/blog/posts/postgresql-daterange-and-efficient-time-management>
7. Struct NpgsqlRange
8. Documentation: 18: 65.2. GiST Indexes - PostgreSQL, дата последнего обращения: декабря 6, 2025, <https://www.postgresql.org/docs/current/gist.html>
9. R-Tree: algorithm for efficient indexing of spatial data - Bartosz Sypytkowski, дата последнего обращения: декабря 6, 2025, <https://www.bartoszsypytkowski.com/r-tree/>
10. Documentation: 9.1: Implementation - PostgreSQL, дата последнего обращения: декабря 6, 2025, <https://www.postgresql.org/docs/9.1/gist-implementation.html>
11. Documentation: 7.1: Index Types - PostgreSQL, дата последнего обращения: декабря 6, 2025, <https://www.postgresql.org/docs/7.1/indices-types.html>
12. Documentation: 7.3: Index Types - PostgreSQL, дата последнего обращения: декабря 6, 2025, <https://www.postgresql.org/docs/7.3/indexes-types.html>
13. The btree\_gist extension - Neon Docs, дата последнего обращения: декабря 6, 2025, <https://neon.com/docs/extensions/btree_gist>
14. btree\_gist improvements in PostgreSQL 18, дата последнего обращения: декабря 6, 2025, <https://www.cybertec-postgresql.com/en/btree_gist-improvements-in-postgresql-18/>
15. Overlapping Ranges in Subsets in PostgreSQL - Simple Talk - Redgate Software, дата последнего обращения: декабря 6, 2025, <https://www.red-gate.com/simple-talk/databases/postgresql/overlapping-ranges-in-subsets-in-postgresql/>
16. Prevent Overlapping Ranges in Versioned Records with Exclusion Constraints - Atomic Spin, дата последнего обращения: декабря 6, 2025, <https://spin.atomicobject.com/versioned-records-prevent-overlaps/>
17. Postgresql Exclusion constraint with soft-deleted rows - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/54305683/postgresql-exclusion-constraint-with-soft-deleted-rows>
18. Protect Your Data with PostgreSQL Constraints - NathanMLong.com, дата последнего обращения: декабря 6, 2025, <https://nathanmlong.com/2016/01/protect-your-data-with-postgresql-constraints/>
19. postgresql - Unique constraint with soft deleted rows excluded - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/13260456/unique-constraint-with-soft-deleted-rows-excluded>
20. Concurrency Tokens | Npgsql Documentation, дата последнего обращения: декабря 6, 2025, <https://www.npgsql.org/efcore/modeling/concurrency.html>
21. PostgreSQL MVCC: The Secret Behind Its Powerful Concurrency | by Jeyaram Ayyalusamy | Medium, дата последнего обращения: декабря 6, 2025, <https://medium.com/@jramcloud1/postgresql-mvcc-the-secret-behind-its-powerful-concurrency-6d6dfe2452d2>
22. Postgres Transaction ID (XID) Wraparound - Bytebase, дата последнего обращения: декабря 6, 2025, <https://www.bytebase.com/blog/postgres-transaction-id-wraparound/>
23. Entity Framework, PostgreSQL, Optimistic Concurrency with hidden xmin column, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/11257826/entity-framework-postgresql-optimistic-concurrency-with-hidden-xmin-column>
24. Supported Types and their Mappings | Npgsql Documentation, дата последнего обращения: декабря 6, 2025, <https://www.npgsql.org/doc/types/basic.html>
25. Ranges and Multiranges | Npgsql Documentation, дата последнего обращения: декабря 6, 2025, <https://www.npgsql.org/efcore/mapping/range.html>
26. 6.0 Release Notes | Npgsql Documentation, дата последнего обращения: декабря 6, 2025, <https://www.npgsql.org/efcore/release-notes/6.0.html>
27. Date and Time Handling | Npgsql Documentation, дата последнего обращения: декабря 6, 2025, <https://www.npgsql.org/doc/types/datetime.html>
28. Entity Framework Core Migrations Guide - Custom Software Development, дата последнего обращения: декабря 6, 2025, <https://blackflow.co.uk/entreprise-software-development/entity-framework-core-migrations-guide/>
29. Postgres Exclude constraints · sqlalchemy sqlalchemy · Discussion #9924 - GitHub, дата последнего обращения: декабря 6, 2025, <https://github.com/sqlalchemy/sqlalchemy/discussions/9924>
30. Combine a PostgreSQL EXCLUDE range constraint with a UNIQUE constraint, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/33464032/combine-a-postgresql-exclude-range-constraint-with-a-unique-constraint>
31. Npgsql EF Core concurrency token property gets included in migrations - Stack Overflow, дата последнего обращения: декабря 6, 2025, <https://stackoverflow.com/questions/78974688/npgsql-ef-core-concurrency-token-property-gets-included-in-migrations>
32. Exceptions, errors and notices | Npgsql Documentation, дата последнего обращения: декабря 6, 2025, <https://www.npgsql.org/doc/diagnostics/exceptions_notices.html>
33. PostgresDB 23P01: Exclusion Violation - Doctor Droid, дата последнего обращения: декабря 6, 2025, <https://drdroid.io/stack-diagnosis/postgresdb-23p01-exclusion-violation>
34. Handling Concurrency Conflicts - EF Core - Microsoft Learn, дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/ef/core/saving/concurrency>
35. Handling Concurrency with the Entity Framework in an ASP.NET MVC Application (7 of 10), дата последнего обращения: декабря 6, 2025, <https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/handling-concurrency-with-the-entity-framework-in-an-asp-net-mvc-application>