sibedge

Не EF Core единым: альтернативная ORM LINQ to DB и её возможности

Алексей Фадеев, старший разработчик .NET

О себе

Алесей Фадеев, старший разработчик .NET

Компания sibedge, г.Томск

Сфера деятельности

Backend-разработка
Работа с СУБД, оптимизация
Используем: PostgreSQL,
Microsoft SQL Server

Контакты

Email: fadeevas@sibedge.com
https://vk.com/fadeev

LINQ to DB, альтернативная ORM

Заявка: преодолеть все ограничения ORM!

Другие преимущества LINQ to DB

Бонус: об интересных фичах PostgreSQL

LINQ to DB

Альтернативная ORM для .NET

Богатый функционал «из коробки»

Возможность легко писать расширения

Высокая производительность

Проблемы ORM

Встречаются плохие запросы для простых операций

Ограниченность (многое не поддерживается)

Проблемы ORM

Встречаются плохие запросы для простых операций

Но опытные разработчики научились писать хорошие

Ограниченность (многое не поддерживается)

Проблемы ORM

Встречаются плохие запросы для простых операций

Но опытные разработчики научились писать хорошие

Ограниченность (многое не поддерживается)

Особенно при работе с Postgres…

…с массой конструкций, выходящих за рамки стандарта SQL

ORM становятся лучше

Например, в EF Core появилась поддержка поиска по элементу массива

	id	title	key_words
	1	ORM для PostreSQL	[postgres, postgresql, dotnet, orm]

```
var items = await context.Articles
          .Where(x => x.KeyWords.Contains("postgres"))
          .ToListAsync();
```

```
SELECT *
   FROM "article" "x"
   WHERE 'postgres' = ANY ("x"."key_words");
```

И всё же, ORM ограничены

```
var items = await context.Articles
          .Where(x => x.KeyWords.Contains("postgres"))
          .ToListAsync();
```

EF Core (нет поддержки индексов)

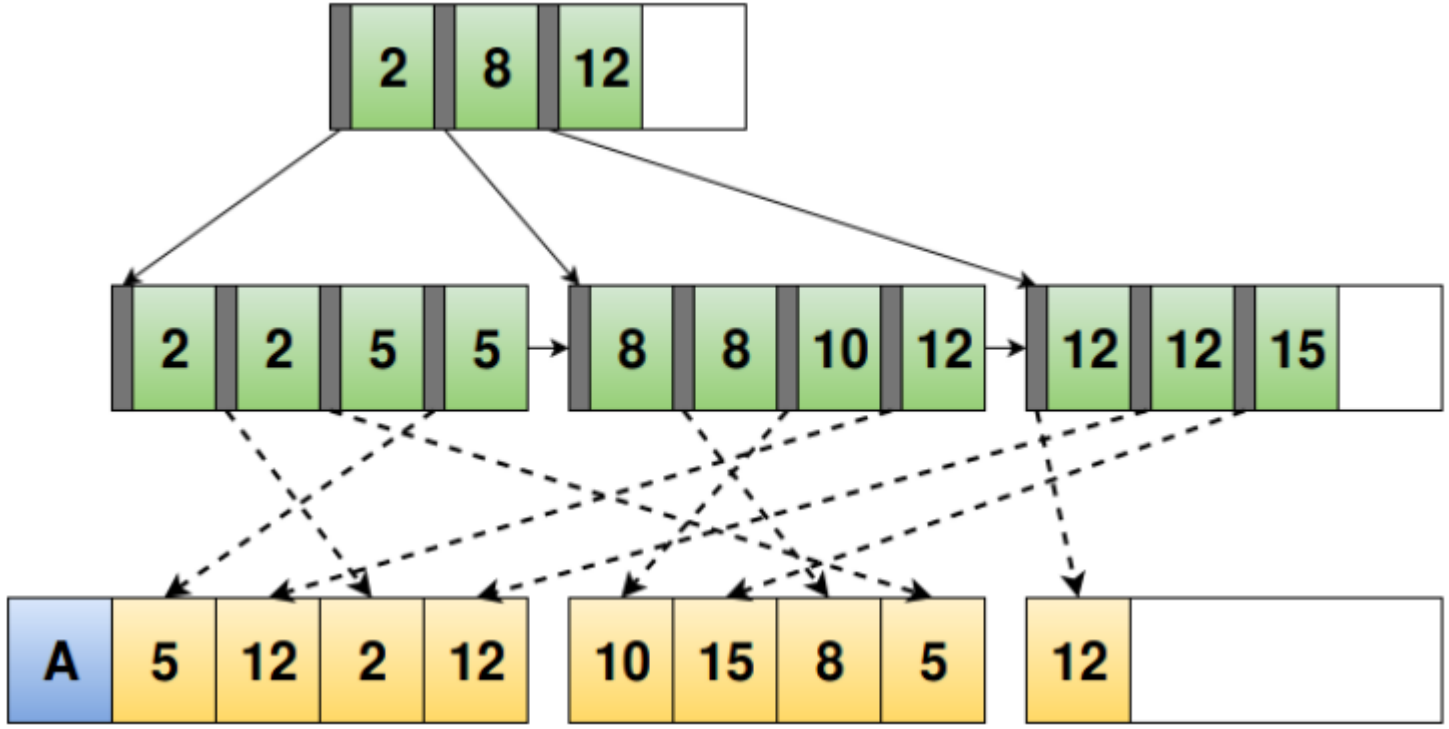
```
SELECT *
   FROM "article" "x"
   WHERE 'postgres' = ANY ("x"."key_words");
```

Хотелось бы (есть поддержка индексов)

```
SELECT *
   FROM "article" "x"
   WHERE "x"."key_words" @> ARRAY ['postgres'];
```

Btree индекс

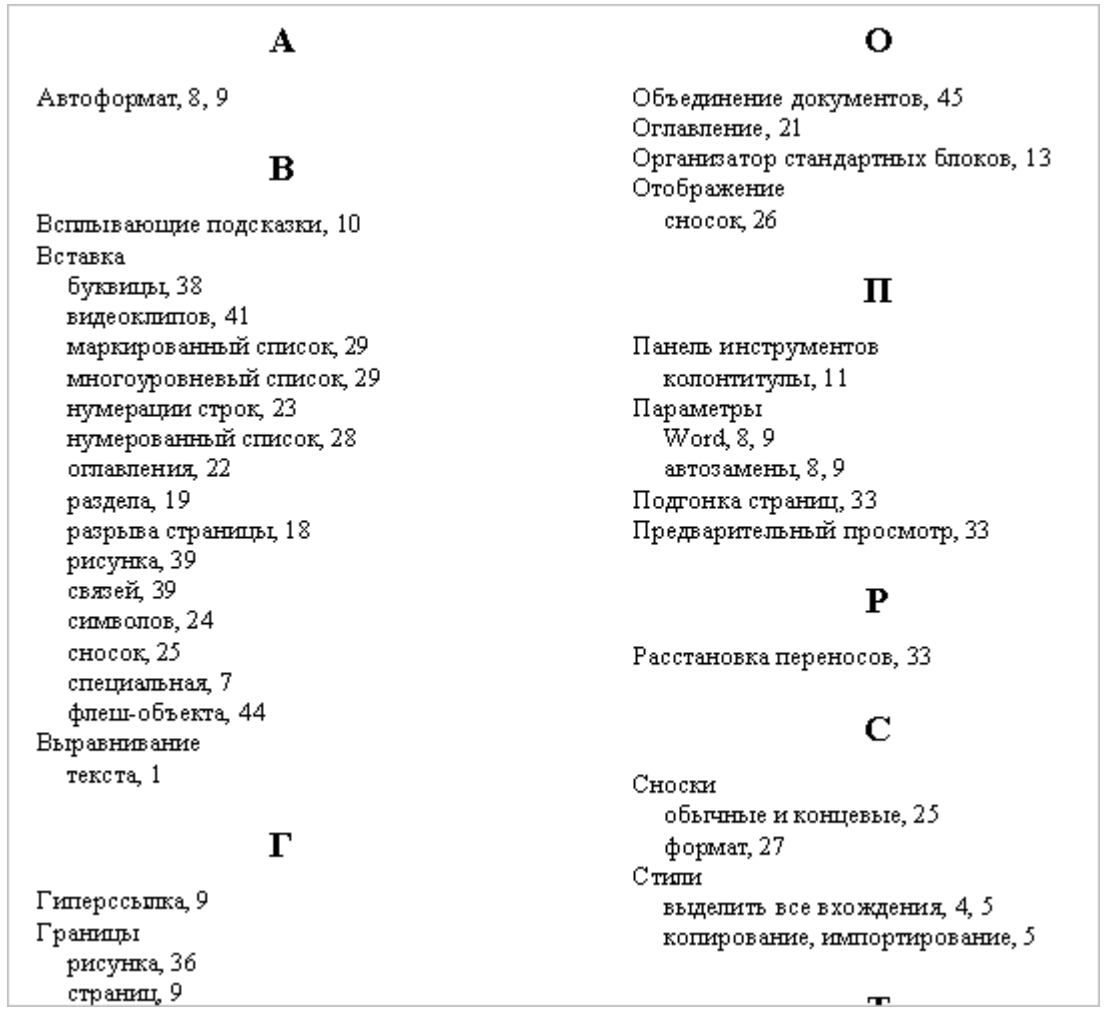
Соответствие между значением индекса и строками таблицы



Индексирование массивов?

```
[1, 2, 3, 4, 5]
[0, 100]
[3, 3, 3, 3, 3, 3, 3, 3]
```

GIN – обобщённый обратный индекс



А

Автоформат, 8, 9

В

Всплывающие подсказки, 10
Вставка
  буквицы, 38
  видеоклипов, 41
  маркированный список, 29
  многоуровневый список, 29
  нумерации строк, 23
  нумерованный список, 28
  оглавления, 22
  раздела, 19
  разрыва страницы, 18
  рисунка, 39
  связей, 39
  символов, 24
  сносок, 25
  специальная, 7
  флеш-объекта, 44
Выравнивание
  текста, 1

Г

Гиперссылка, 9
Границы
  рисунка, 36
  страниц, 9

О

Объединение документов, 45
Оглавление, 21
Организатор стандартных блоков, 13
Отображение
  сносок, 26

П

Панель инструментов
  колонтитулы, 11
Параметры
  Word, 8, 9
  автозамены, 8, 9
Подгонка страниц, 33
Предварительный просмотр, 33

Р

Расстановка переносов, 33

С

Сноски
  обычные и концевые, 25
  формат, 27
Стили
  выделить все вхождения, 4, 5
  копирование, импортирование, 5

Т

И всё же, ORM ограничены

```
var items = await context.Articles
          .Where(x => x.KeyWords.Contains("postgres"))
          .ToListAsync();
```

EF Core (нет поддержки индексов)

```
SELECT *
   FROM "article" "x"
   WHERE 'postgres' = ANY ("x"."key_words");
```

Хотелось бы (есть поддержка индексов)

```
SELECT *
   FROM "article" "x"
   WHERE "x"."key_words" @> ARRAY ['postgres'];
```

LINQ to DB

Альтернативная ORM для .NET

Богатый функционал «из коробки»

Возможность легко писать расширения

Высокая производительность

Работа с массивами

ArrayAggregate<T>(ISqlExtension?, T, AggregateModifier)
ArrayAggregate<T>(ISqlExtension?, T)
ArrayAggregate<TEntity, TV>(IEnumerable<TEntity>, Func<TEntity, TV>, …)
ArrayAggregate<TEntity, TV>(IQueryable<TEntity>, Expression<Func<TEntity, TV>>)
ArrayAppend<T>(IPostgreSQLExtensions?, T[], T)
ArrayCat<T>(IPostgreSQLExtensions?, T[], T[])
ArrayDims<T>(IPostgreSQLExtensions?, T[])
ArrayLength<T>(IPostgreSQLExtensions?, T[], int)
ArrayLower<T>(IPostgreSQLExtensions?, T[], int)
ArrayNDims<T>(IPostgreSQLExtensions?, T[])
ArrayPosition<T>(IPostgreSQLExtensions?, T[], T, int)
ArrayPosition<T>(IPostgreSQLExtensions?, T[], T)
ArrayPositions<T>(IPostgreSQLExtensions?, T[], T)
ArrayPrepend<T>(IPostgreSQLExtensions?, T, T[])
ArrayRemove<T>(IPostgreSQLExtensions?, T[], T)
ArrayReplace<T>(IPostgreSQLExtensions?, T[], T, T)
ArrayToString<T>(IPostgreSQLExtensions?, T[], string, string)
ArrayToString<T>(IPostgreSQLExtensions?, T[], string)
ArrayUpper<T>(IPostgreSQLExtensions?, T[], int)
Cardinality<T>(IPostgreSQLExtensions?, T[])
ConcatArrays<T>(IPostgreSQLExtensions?, T[], T[][])
ConcatArrays<T>(IPostgreSQLExtensions?, T[][], T[])
ConcatArrays<T>(IPostgreSQLExtensions?, params T[][])
ContainedBy<T>(IPostgreSQLExtensions?, T[], T[])
Contains<T>(IPostgreSQLExtensions?, T[], T[])
GreaterThan<T>(IPostgreSQLExtensions?, T[], T[])
GreaterThanOrEqual<T>(IPostgreSQLExtensions?, T[], T[])
LessThan<T>(IPostgreSQLExtensions?, T[], T[])
LessThanOrEqual<T>(IPostgreSQLExtensions?, T[], T[])
Overlaps<T>(IPostgreSQLExtensions?, T[], T[])
StringToArray(IPostgreSQLExtensions?, string, string, string)
StringToArray(IPostgreSQLExtensions?, string, string)
ValueIsEqualToAny<T>(IPostgreSQLExtensions?, T, T[])
ValueIsGreaterThanAny<T>(IPostgreSQLExtensions?, T, T[])
ValueIsGreaterThanOrEqualToAny<T>(IPostgreSQLExtensions?, T, T[])
ValueIsLessThanAny<T>(IPostgreSQLExtensions?, T, T[])
ValueIsLessThanOrEqualToAny<T>(IPostgreSQLExtensions?, T, T[])
ValueIsNotEqualToAny<T>(IPostgreSQLExtensions?, T, T[])

Поиск в массиве с поддержкой индекса

	id	number	days_of_week
	1	5N-229	[1, 2, 3, 4, 5]
	2	5U-1531	[3, 6, 7]

```
var filter = new[] { 2, 5 };

var result = await context.Routes
    .Where(x =>
Sql.Ext.PostgreSQL().Contains(x.DaysOfWeek, filter))
    .ToListAsync();
```

Поиск в массиве с поддержкой индекса

	id	number	days_of_week
	1	5N-229	[1, 2, 3, 4, 5]
	2	5U-1531	[3, 6, 7]

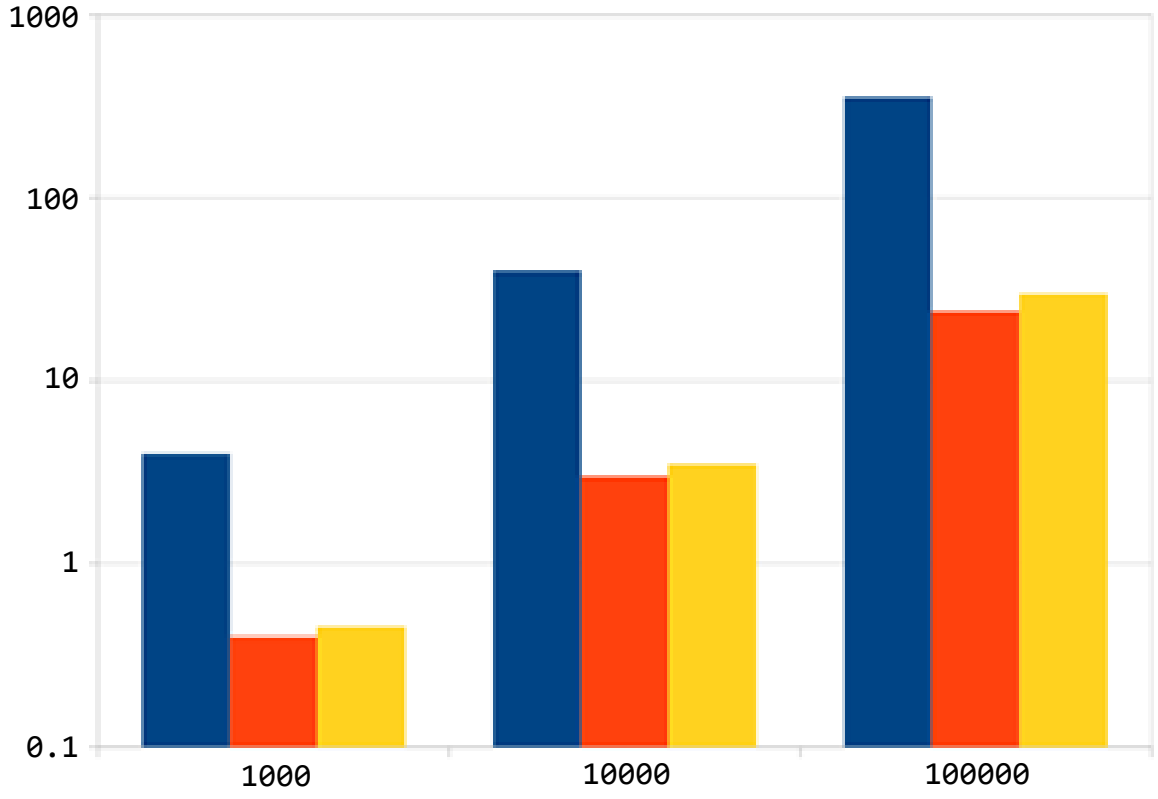
```
SELECT
    x.aircraft_code, x.arrival_airport, x.arrival_airport_name,
    x.arrival_city, x.days_of_week, x.departure_airport,
    x.departure_airport_name, x.departure_city, x.duration, x.flight_no
FROM
    "public".routes x
WHERE
    x.days_of_week @> :filter_1
```

Массивы в СУБД

Для ряда задач: простые и элегантные решения

Повышение производительности

Разные модели хранения: производительность



Count	flat	array	jsonb
1000	4 ms	0,40 ms	0,45 ms
10 000	40 ms	3 ms	3,5 ms
100 000	357 ms	24 ms	30 ms

Реляционный вид

Массивы

jsonb

Запросы WITH (CTE)

Позволяют разбивать сложные запросы на простые части

Рекурсия: запрос WITH может обращаться к собственному результату

Запросы WITH (CTE)

Пример задачи:

- Посчитать среднее значение стоимости
- По-разному обработать записи со стоимостью больше и меньше средней
- Сгруппировать по дням

Запросы WITH (CTE)

```
WITH a AS (SELECT avg(total_amount) av FROM bookings),
     d AS (SELECT book_date::date, total_amount FROM bookings)

SELECT NULL AS book_date, avg(total_amount) av1 FROM bookings b
  JOIN a ON TRUE
  WHERE b.total_amount > a.av

UNION

SELECT book_date, avg(total_amount) av1 FROM d
  JOIN a ON TRUE
  WHERE d.total_amount <= a.av
  GROUP BY book_date;
```

Запросы WITH (CTE)

```
var averageCte = context.Bookings.GroupBy(x => true)
    .Select(g => g.Average(b => b.TotalAmount))
    .AsCte();

var dateCte = context.Bookings
    .Select(x => new Book(x.BookDate.Date, x.TotalAmount))
    .AsCte();

var resultHi = from b in context.Bookings
    join a in averageCte on true equals true
    where b.TotalAmount > a
    group b by true into g
    select new Book(null, g.Average(x => x.TotalAmount));

var resultLow = from b in dateCte
    join a in averageCte on true equals true
    where b.TotalAmount <= a
    group b by b.BookDate into g
    select new Book(g.Key, g.Average(x => x.TotalAmount));

var result = await resultHi.Concat(resultLow).ToListAsync();
```

Запросы WITH (CTE)

```
WITH "CTE_1" (cte_field_1)
AS (
  SELECT Avg(t1.total_amount) FROM
    (SELECT True as "Key_1", "selectParam".total_amount FROM "public".bookings "selectParam") t1
     GROUP BY t1."Key_1"
),
"CTE_2" ("TotalAmount", "BookDate")
AS (
  SELECT x.total_amount, Cast(x.book_date as Date) FROM "public".bookings x
)
SELECT t3."BookDate", t3."TotalAmount"
FROM (
  SELECT Cast(NULL as TimeStamp) as "BookDate", Avg(t2.total_amount) as "TotalAmount"
  FROM (
    SELECT True as "Key_1", b.total_amount FROM "public".bookings b CROSS JOIN "CTE_1" a
      WHERE b.total_amount > a.cte_field_1
    ) t2
    GROUP BY t2."Key_1"
  ) t3

UNION ALL

SELECT t4."BookDate", t4."TotalAmount"
FROM (
  SELECT b_1."BookDate", Avg(b_1."TotalAmount") as "TotalAmount"
  FROM "CTE_2" b_1 CROSS JOIN "CTE_1" a_1 WHERE b_1."TotalAmount" <= a_1.cte_field_1
  GROUP BY b_1."BookDate"
  ) t4
```

Рекурсивные CTE

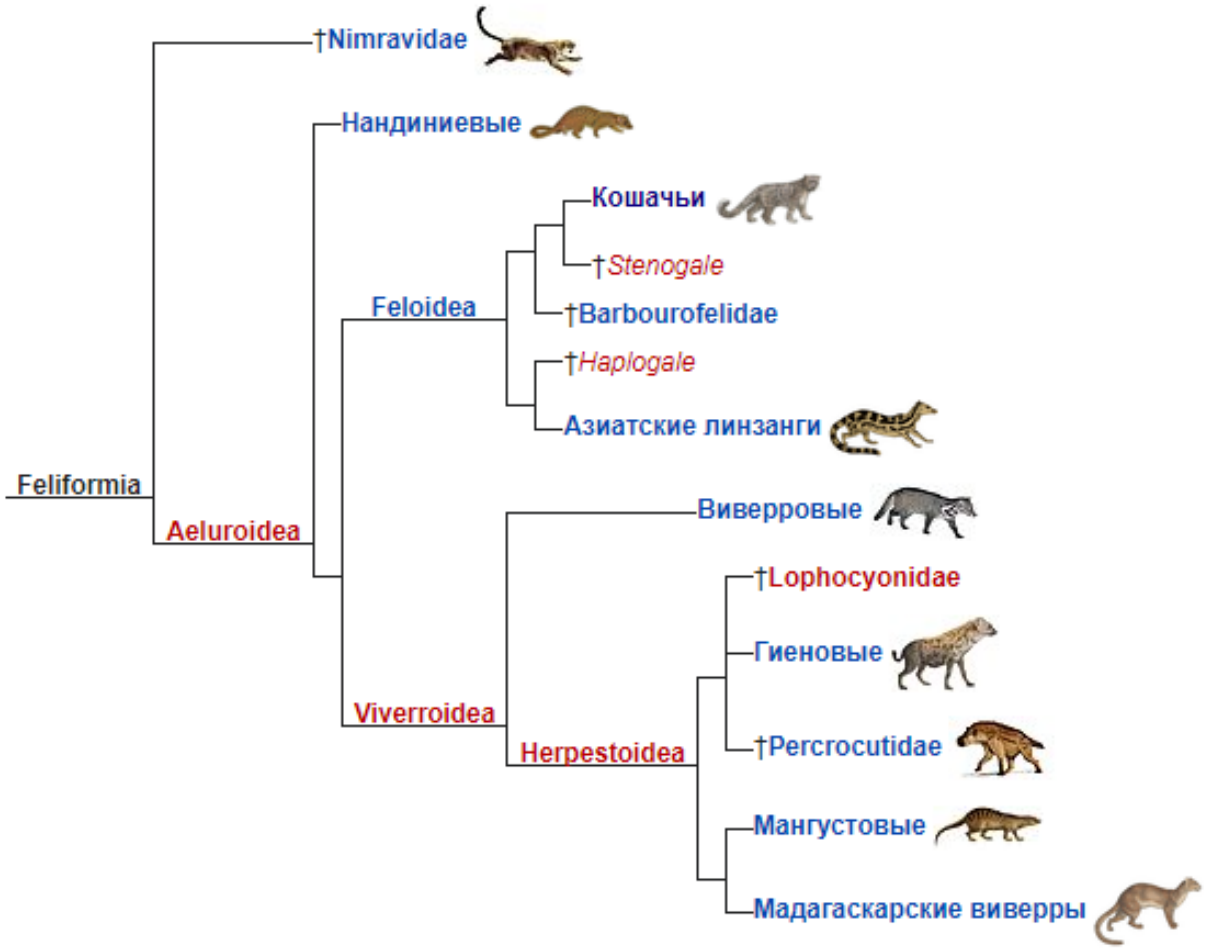
Пример запроса с рекурсией:

Просуммировать числа от 1 до 100

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Рекурсивные CTE

Для иерархических структур



Рекурсивные CTE

Пример задачи:

- Задано несколько Id
- Получить всех предков по иерархии

Рекурсивные CTE

Пишем класс для CTE (набор передаваемых параметров)

```
private class FolderHierarchy
{
    /// <summary> Заданный Id </summary>
    public int StartId { get; set; }

    /// <summary> Id текущей папки </summary>
    public int CurrentId { get; set; }

    /// <summary> Id родительской папки </summary>
    public int? ParentId { get; set; }

    /// <summary> Уровень иерархии </summary>
    public int HierarchyLevel { get; set; }
}
```

Рекурсивные CTE

```
var hierarchyCte = context.GetCte<FolderHierarchy>(hierarchy =>
{
    return
        (
            from f in context.Folders
            where f.Id == 23 || f.Id == 11
            select new FolderHierarchy
            {
                CurrentId = f.Id,
                StartId = f.Id,
                ParentId = f.ParentId,
                HierarchyLevel = 1
            }
        )
        .Concat
        (
            from f in context.Folders
            join h in hierarchy on f.Id equals h.ParentId
            select new FolderHierarchy
            {
                CurrentId = f.Id,
                StartId = h.StartId,
                ParentId = f.ParentId,
                HierarchyLevel = h.HierarchyLevel + 1
            }
        );
});
```

Начало поиска

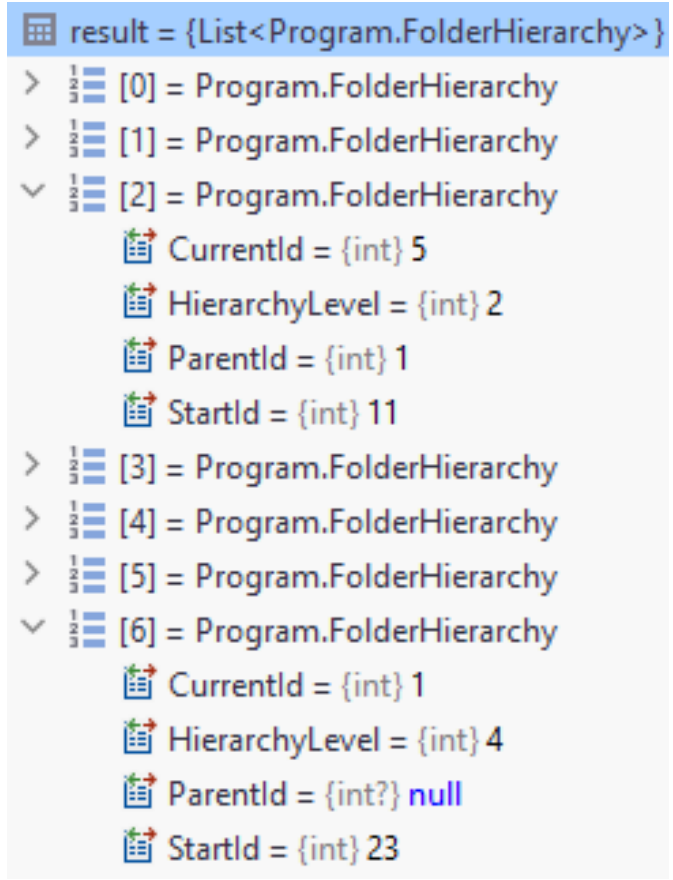
Задаём Id

Рекурсия

JOIN родителя

Рекурсивные CTE

```
WITH RECURSIVE hierarchy0
(
    "StartId", "CurrentId", "HierarchyLevel", "ParentId"
)
AS
(
    SELECT f.id, f.id, 1, f.parent_id
    FROM test.folder f
    WHERE (f.id = 23 OR f.id = 11)
  UNION ALL
    SELECT h."StartId", f_1.id, h."HierarchyLevel" + 1,
        f_1.parent_id
    FROM test.folder f_1
        INNER JOIN hierarchy0 h ON f_1.id = h."ParentId"
)
SELECT * FROM hierarchy0 t1
ORDER BY t1."StartId", t1."HierarchyLevel"
```



Табличные функции

Функция может возвращать набор строк (таблицу)

Пример: билетная система транспорта

Получить список тарифов для мест прохода

```
var result = await context.GetTariffList(place_id)
    .ToListAsync();
```

EF Core: можно как raw-SQL

Табличные функции

Пример задачи:

Получить информацию по проходам за период времени

Для каждого места прохода получить тарифы

Простое решение: вызвать функцию n раз

Табличные функции

Решение на LINQ to DB

```
var result = await (from t in context.PassTransactions
        from tl in context.GetTariffList(t.PlaceId)
        select tl)
    .ToListAsync();
```

```
SELECT tl
FROM pass_transactions t
   INNER JOIN LATERAL
      get_tariff_list(t.place_id) tl ON 1=1
```

LINQ to DB: чего нет в EF

Богатый инструментарий для работы с массивами

Поддержка CTE

Продвинутое использование табличных функций

Поддержка оператора MERGE

…

LINQ to DB: чего нет в EF

Богатый инструментарий для работы с массивами

Поддержка CTE

Продвинутое использование табличных функций

Поддержка оператора MERGE

…

Проблема ограниченности ORM

Особенно при работе с Postgres…

LINQ to DB

Альтернативная ORM для .NET

Богатый функционал «из коробки»

Возможность легко писать расширения

Высокая производительность

Встроенный полнотекстовый поиск

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector @@
  'cat & rat'::tsquery;
```

Удобно делать tsvector вычисляемой колонкой

```
ALTER TABLE folder
ADD COLUMN full_text_data tsvector GENERATED ALWAYS
  AS ((name || ' ' || comment)::tsvector) STORED;
```

Встроенный полнотекстовый поиск

id	name	full_text_data
1	Чилийская кошка	'кошк':2 'чилийск':1
2	Камышовый кот	'камышов':1 'кот':2

Нужно выполнить запрос:

```
SELECT full_text_data @@ to_tsquery('russian', :text_query)
   FROM folder;
```

ORM не поддерживают

Даже LINQ to DB

Встроенный полнотекстовый поиск

LinqToDb: пишем метод-расширение

```
[Sql.Expression("{0} @@ to_tsquery('russian', {1})", ServerSideOnly = true,
    Precedence = Precedence.Comparison, IsPredicate = true)]
public static bool Matches(this NpgsqlTsVector v, string p) =>
    throw new NotImplementedException();
```

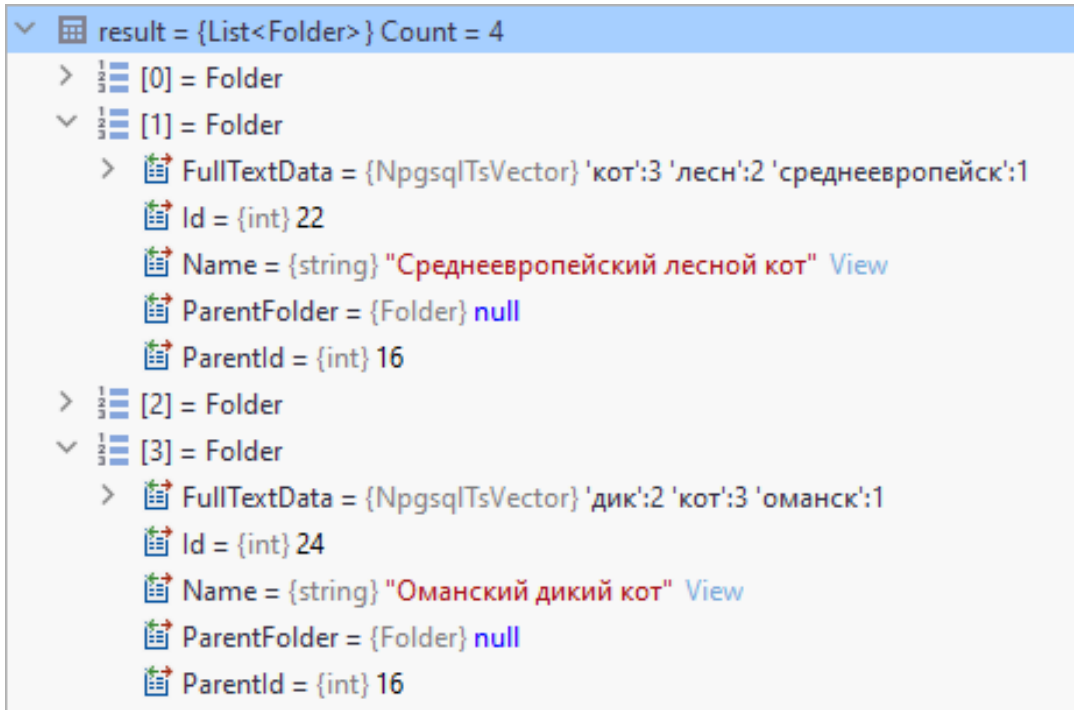
```
var result = await context.Folders
    .Where(x => x.FullTextData.Matches("лесные | дикие"))
    .ToListAsync();
```

Встроенный полнотекстовый поиск

id	name	full_text_data
1	Чилийская кошка	'кошк':2 'чилийск':1
2	Камышовый кот	'камышов':1 'кот':2

```
var result = await context.Folders
    .Where(x => x.FullTextData.Matches("лесные | дикие"))
    .ToListAsync();
```

```
SELECT *
FROM folder x
WHERE x.full_text_data @@
    to_tsquery('russian', 'лесные | дикие')
```



Поддержка словарей Hunspell

Встроенный полнотекстовый поиск

Не требует использования внешних сервисов

Нет временного лага

Поддержка индексов

Поддержка словарей Hunspell

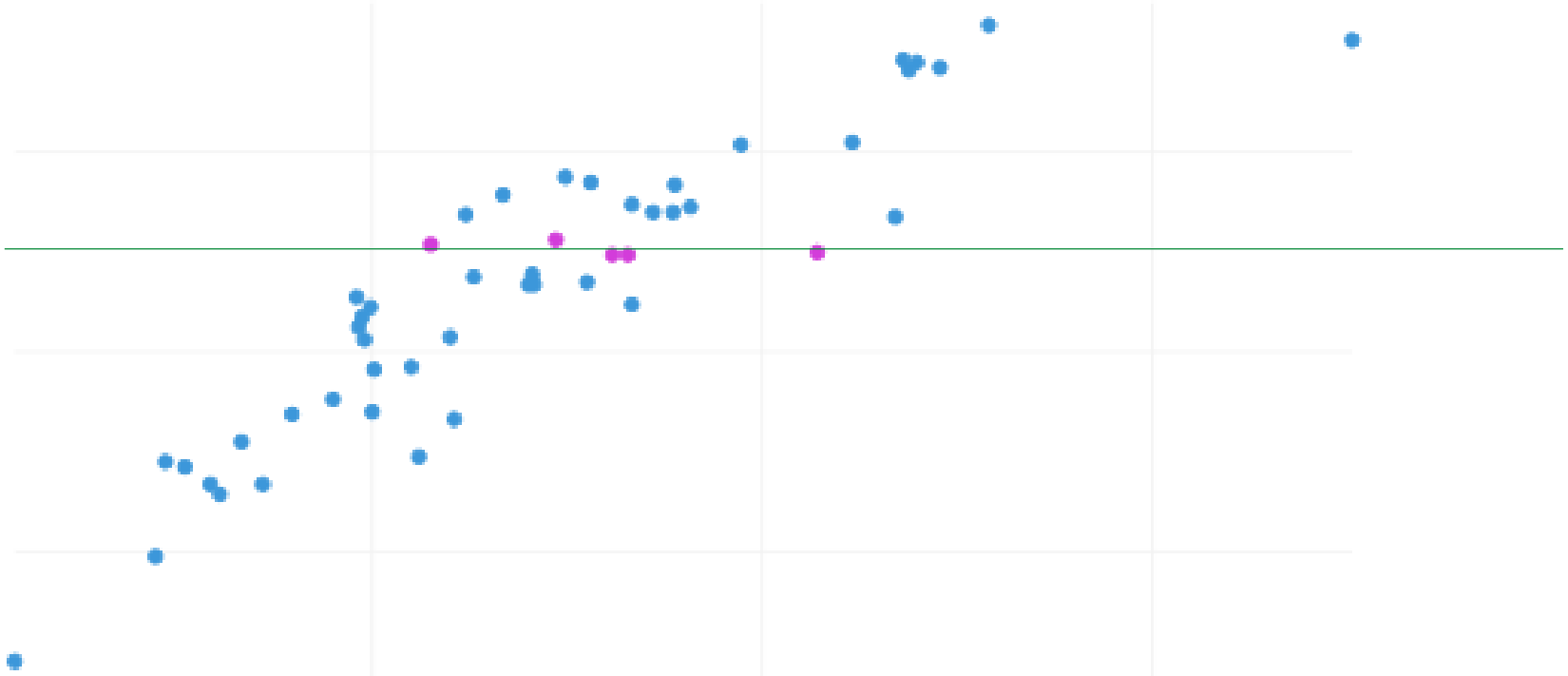
LINQ to DB: полнотекстовый поиск через ORM!

Поиск в пространстве

Одномерное пространство: просто

Пример: история показаний датчика за большой период времени

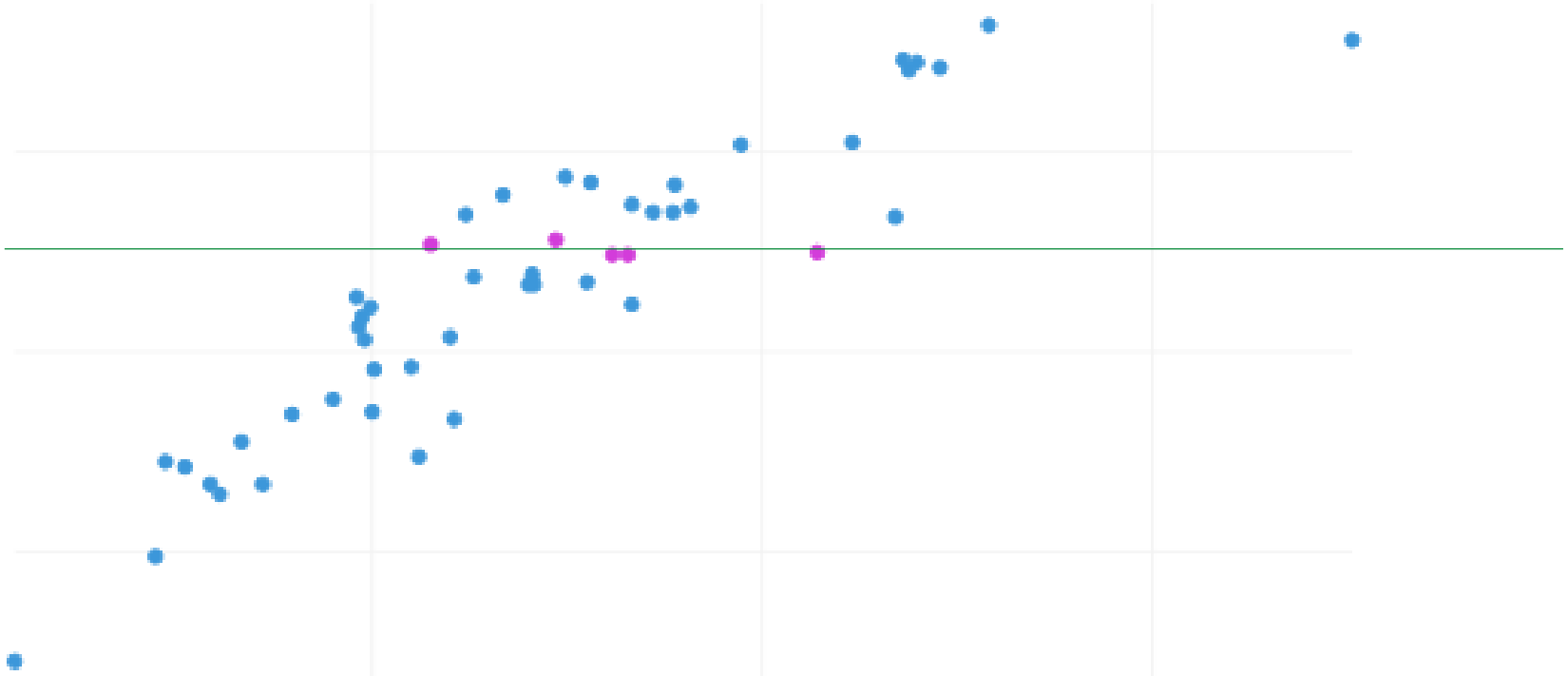
Найти **k** ближайших значений (в обе стороны)



Поиск в пространстве

	id	datetime	value
	1	2020-05-01 13:26:33	19.36
	2	2020-05-01 13:26:34	19.35

Найти **k** ближайших значений (в обе стороны)



Поиск в пространстве

	id	datetime	value
	1	2020-05-01 13:26:33	19.36
	2	2020-05-01 13:26:34	19.35

Найти **k** ближайших значений (в обе стороны)

```
SELECT * FROM data
  ORDER BY ABS(:myvalue - value)
  LIMIT 50
```

Не будет использован индекс

Поиск в пространстве

	id	datetime	value
	1	2020-05-01 13:26:33	19.36
	2	2020-05-01 13:26:34	19.35

Найти **k** ближайших значений (в обе стороны)

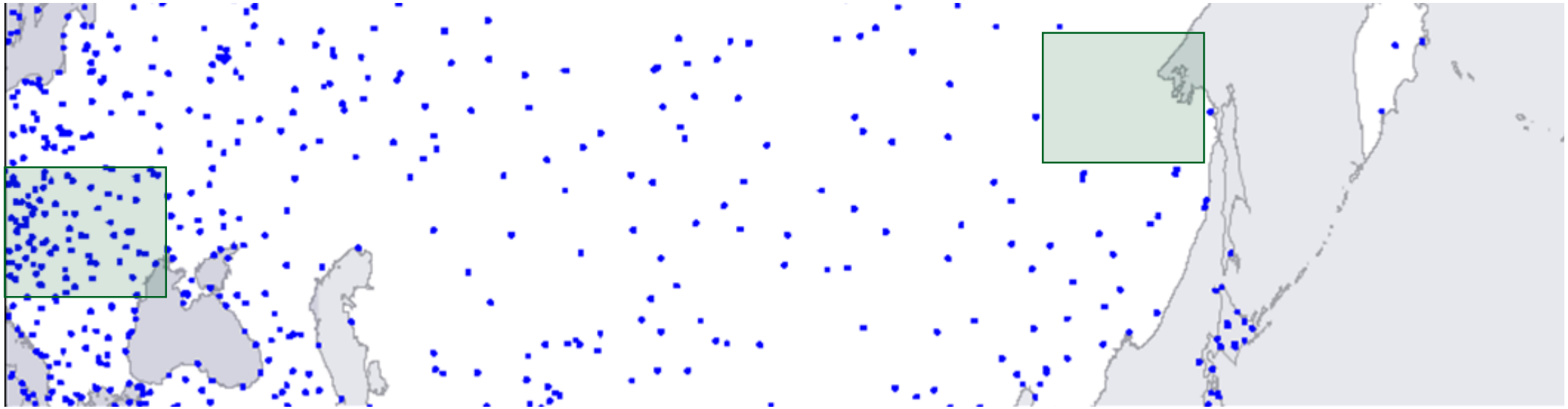
```
SELECT * FROM
(SELECT * FROM data WHERE value >= :myvalue
  ORDER BY value LIMIT 50) s1

UNION ALL

SELECT * FROM
(SELECT * FROM data WHERE value < :myvalue
  ORDER BY value DESC LIMIT 50) s2
```

Поиск в 2D-пространстве

Двухмерное пространство

Задача: найти ближайший бар

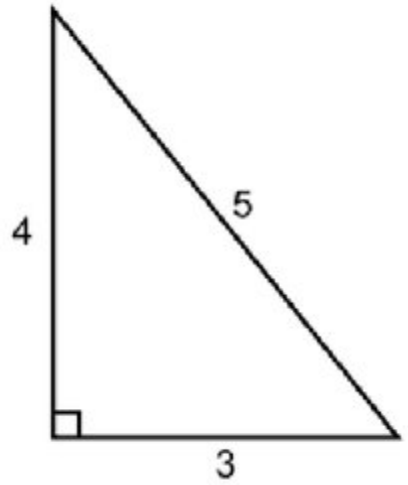
Ограничить прямоугольную область?



Поиск в пространстве (KNN)

```
SELECT point(3, 0) <-> point (0, 4);
```

```
5
```



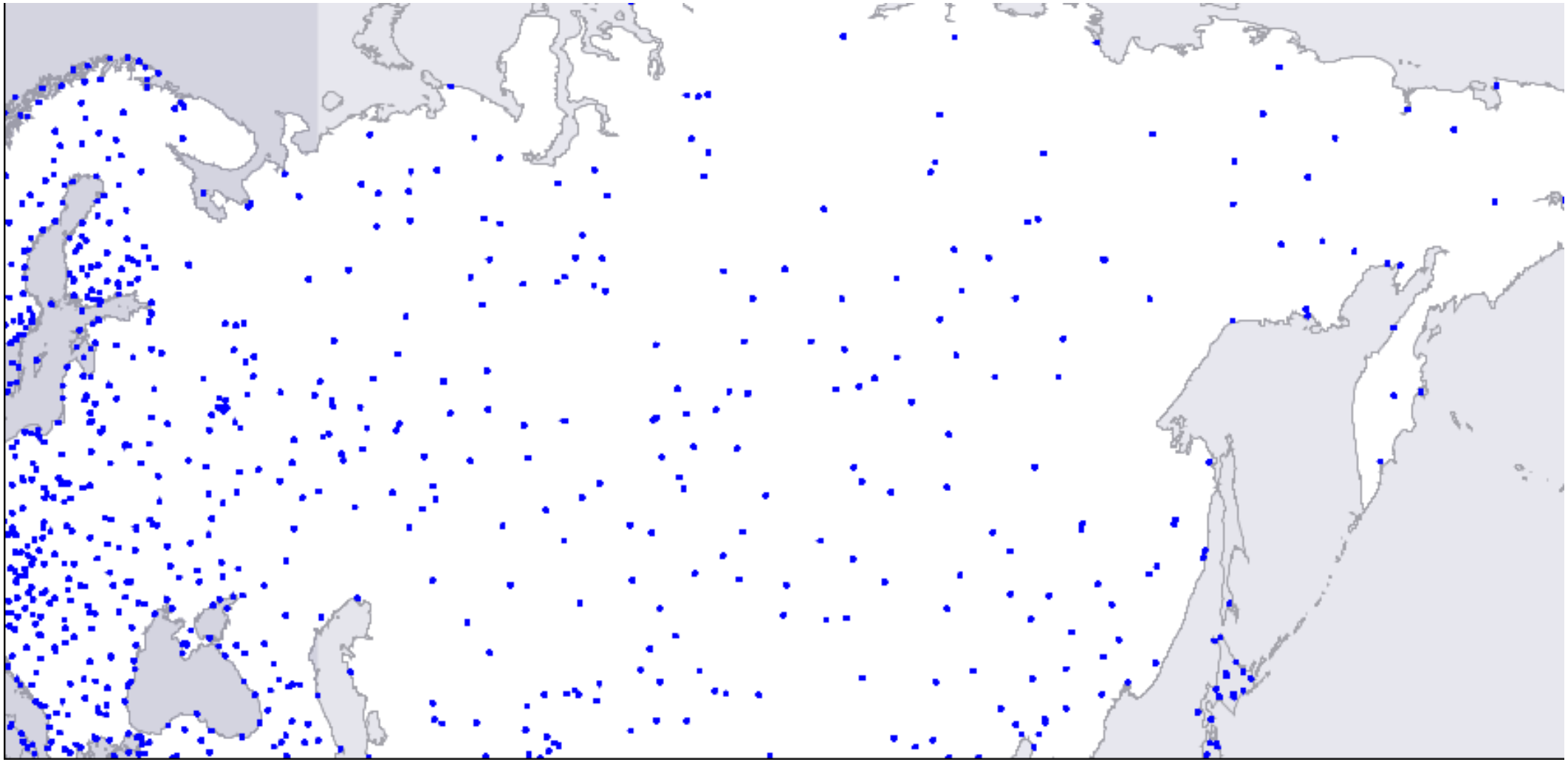
Поиск в пространстве (KNN)

id	amenity	coordinates
1	shop	(64.2, 88.3)
2	bar	(50.5, 87.4)

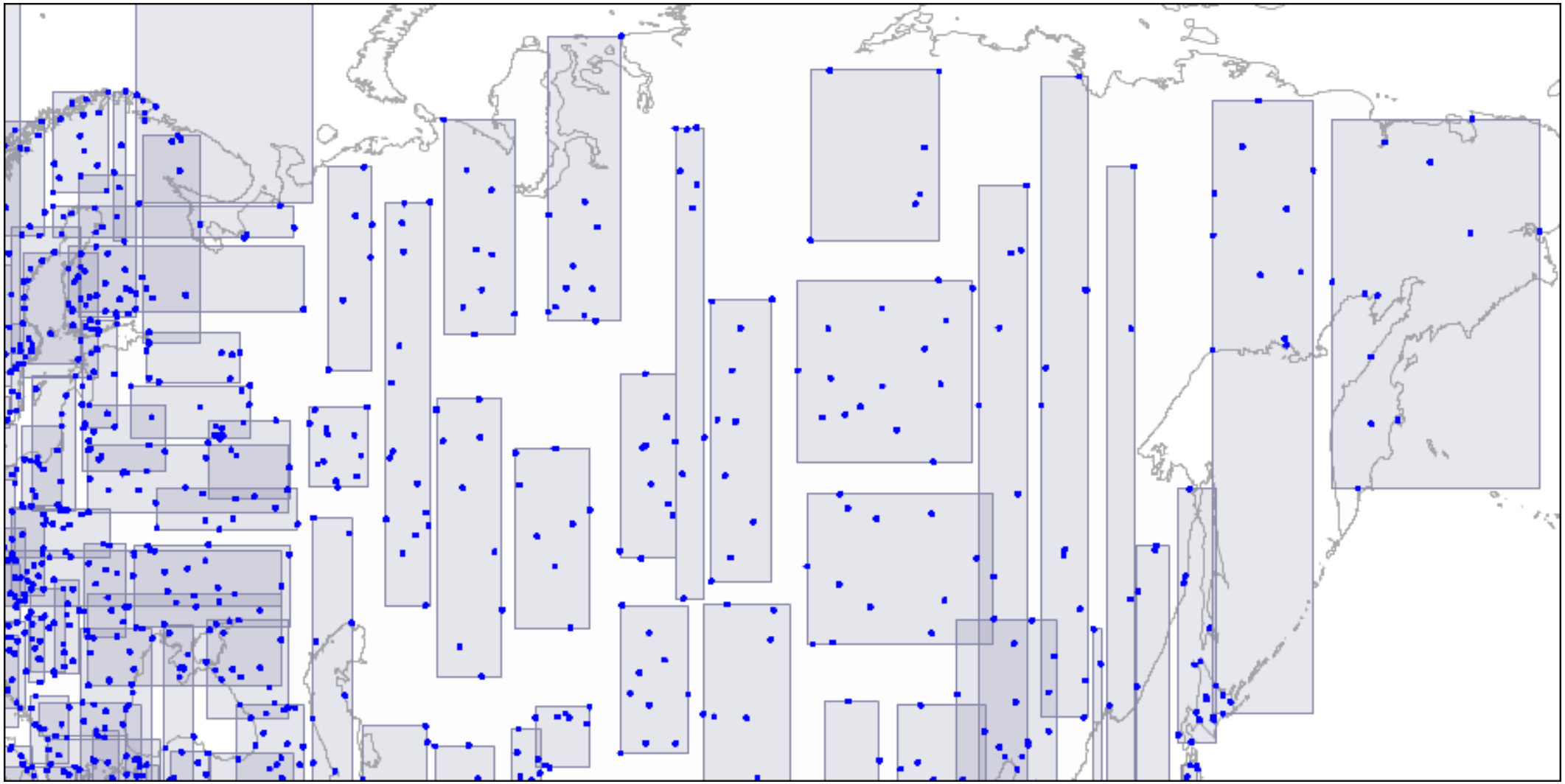
```
SELECT * FROM building
  ORDER BY coordinates <-> point (53.7, 87.7)
  LIMIT 10;
```

Индекс GiST: индексный поиск от произвольной точки

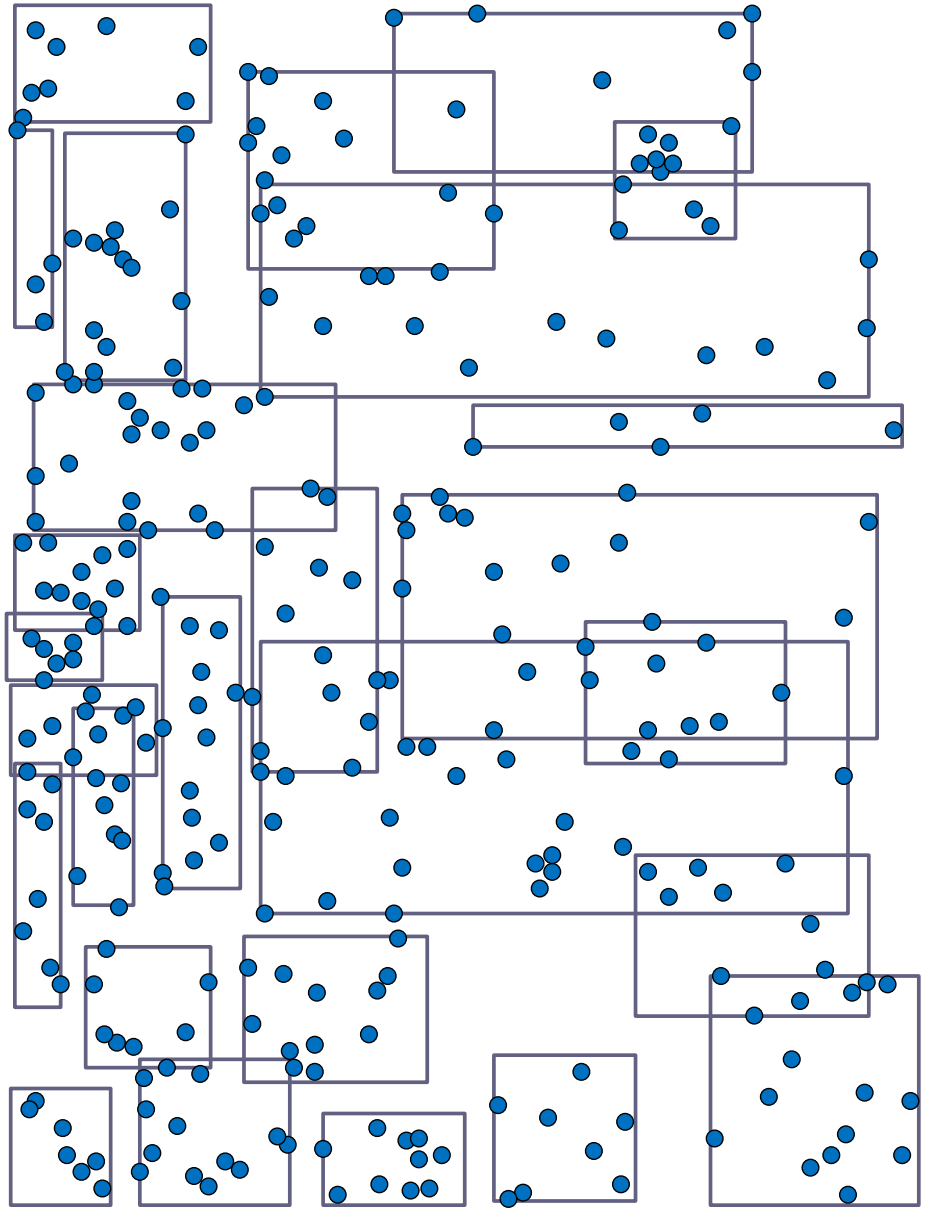
Индекс GiST



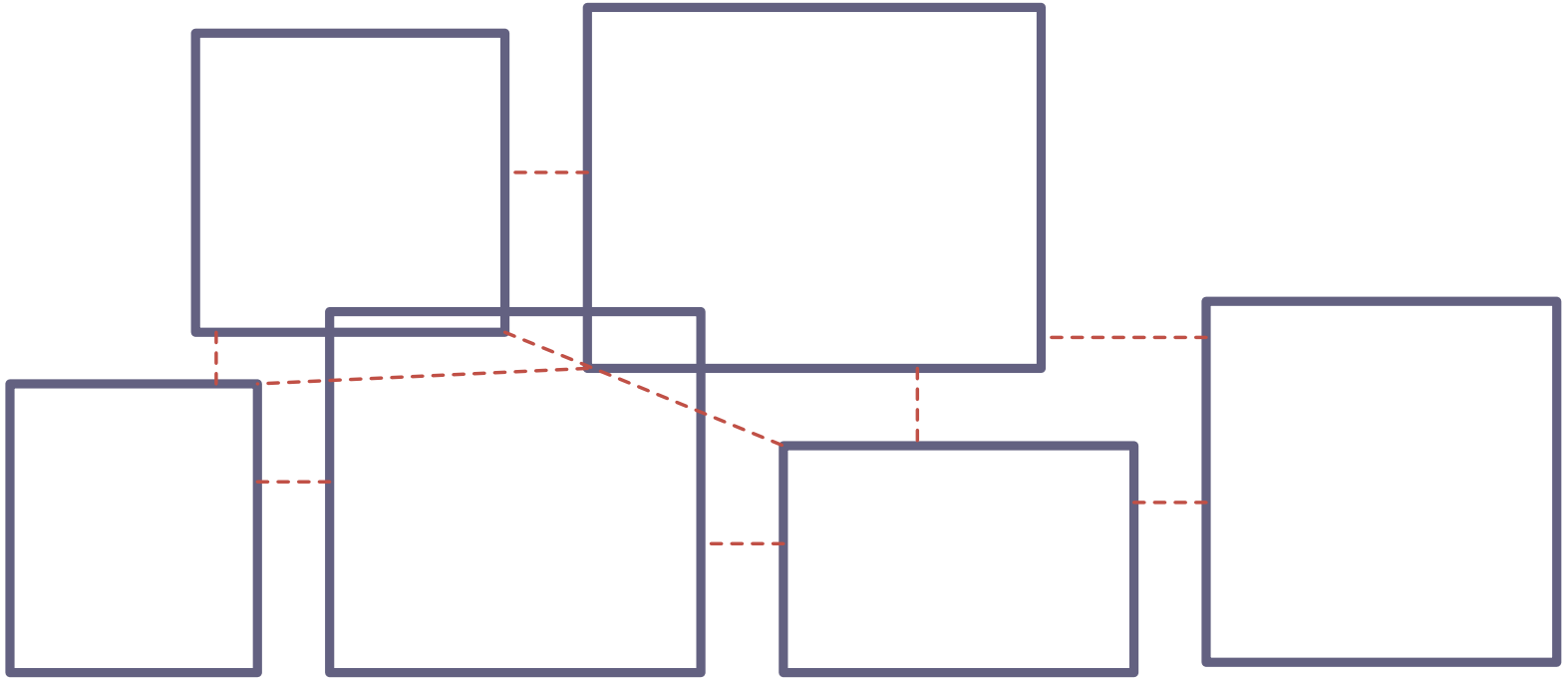
Индекс GiST



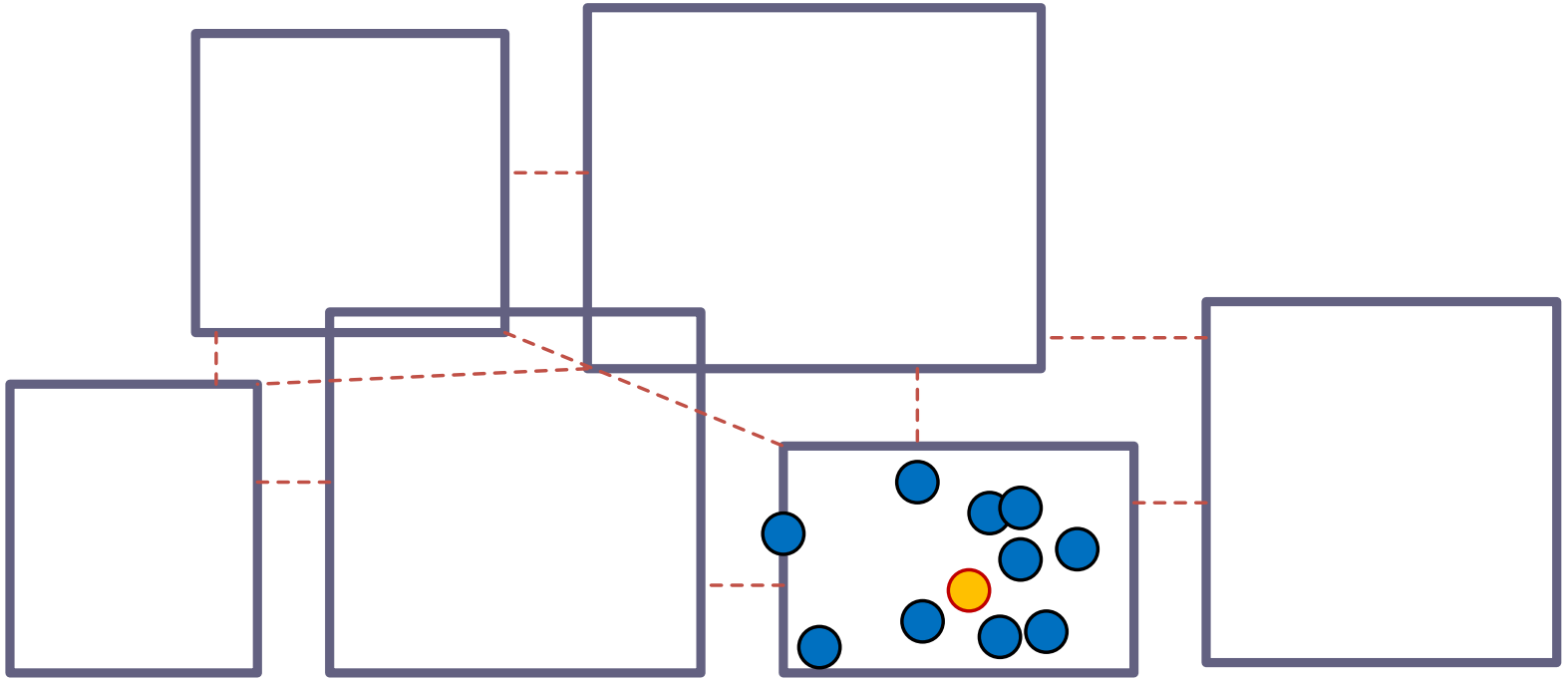
Индекс GiST



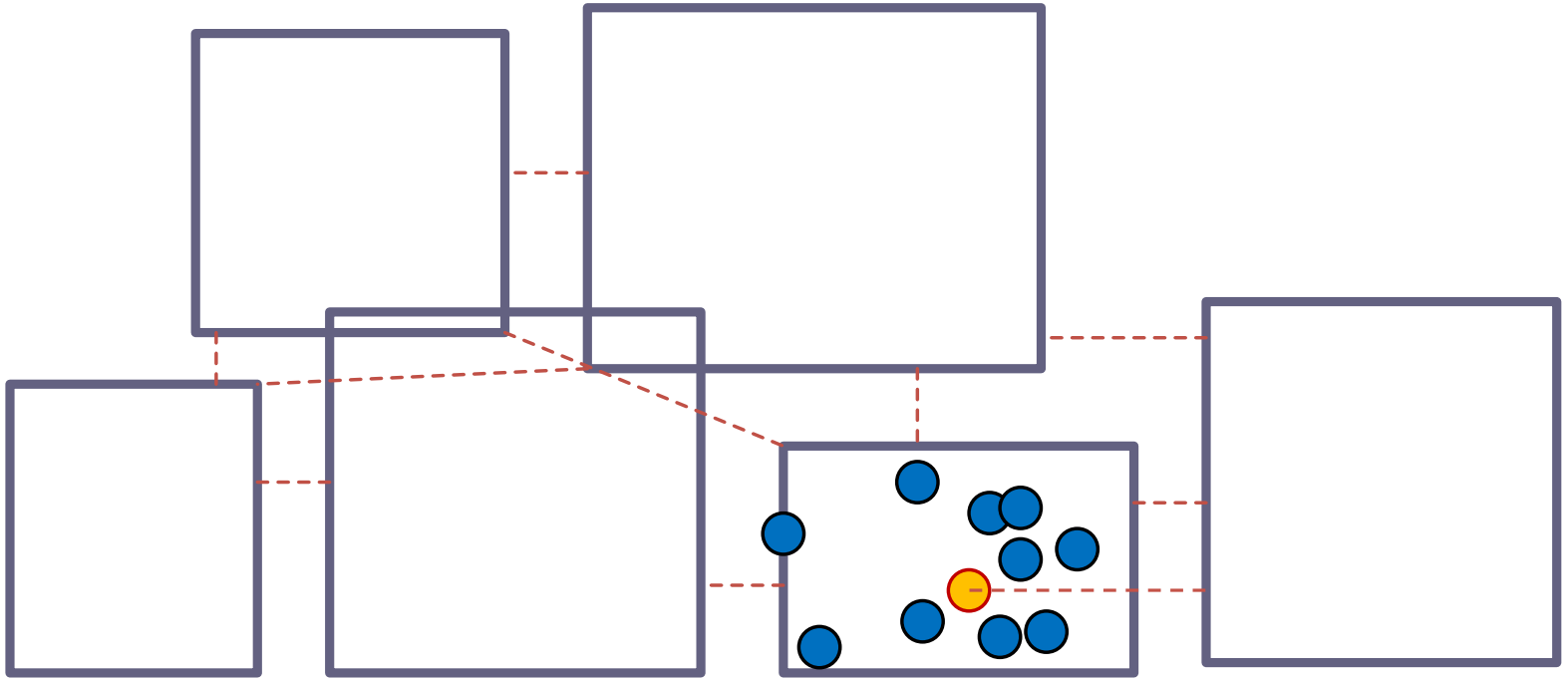
Индекс GiST



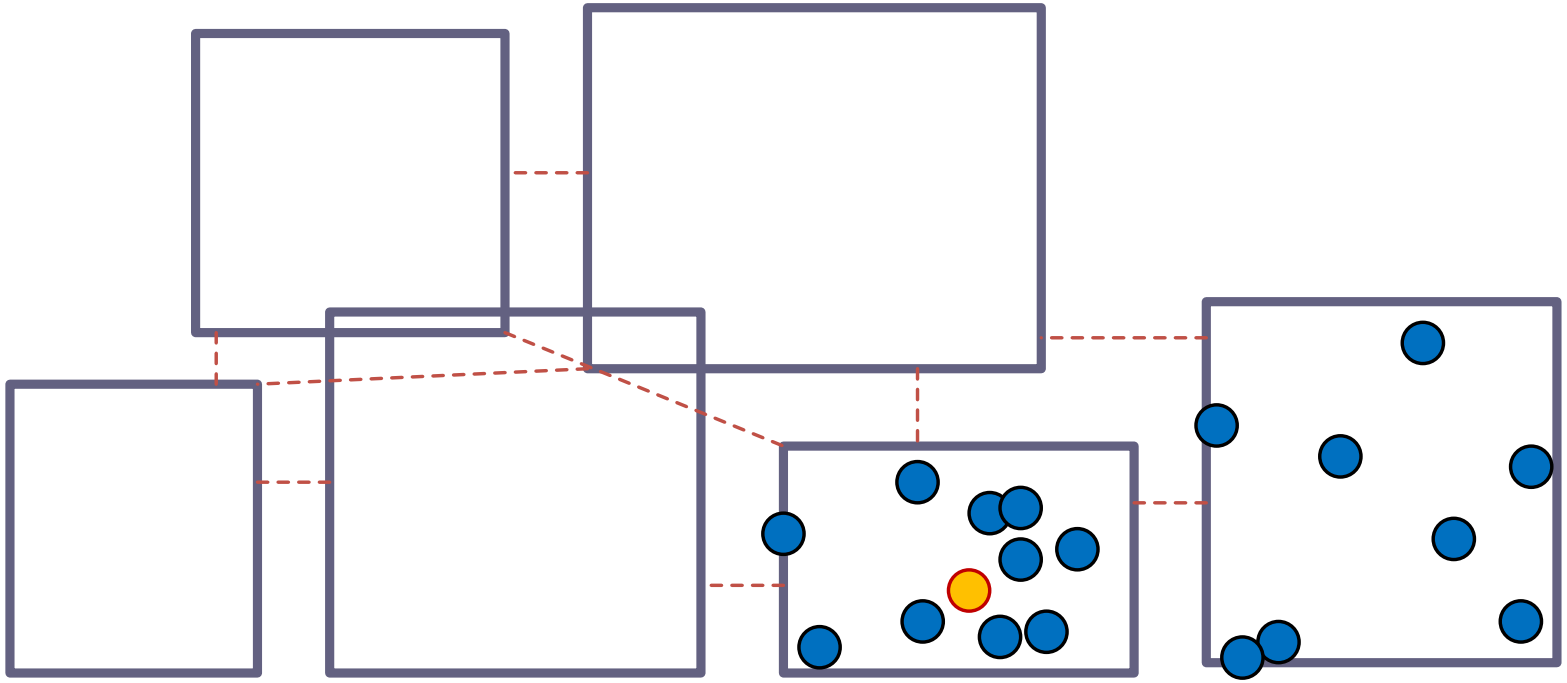
Индекс GiST



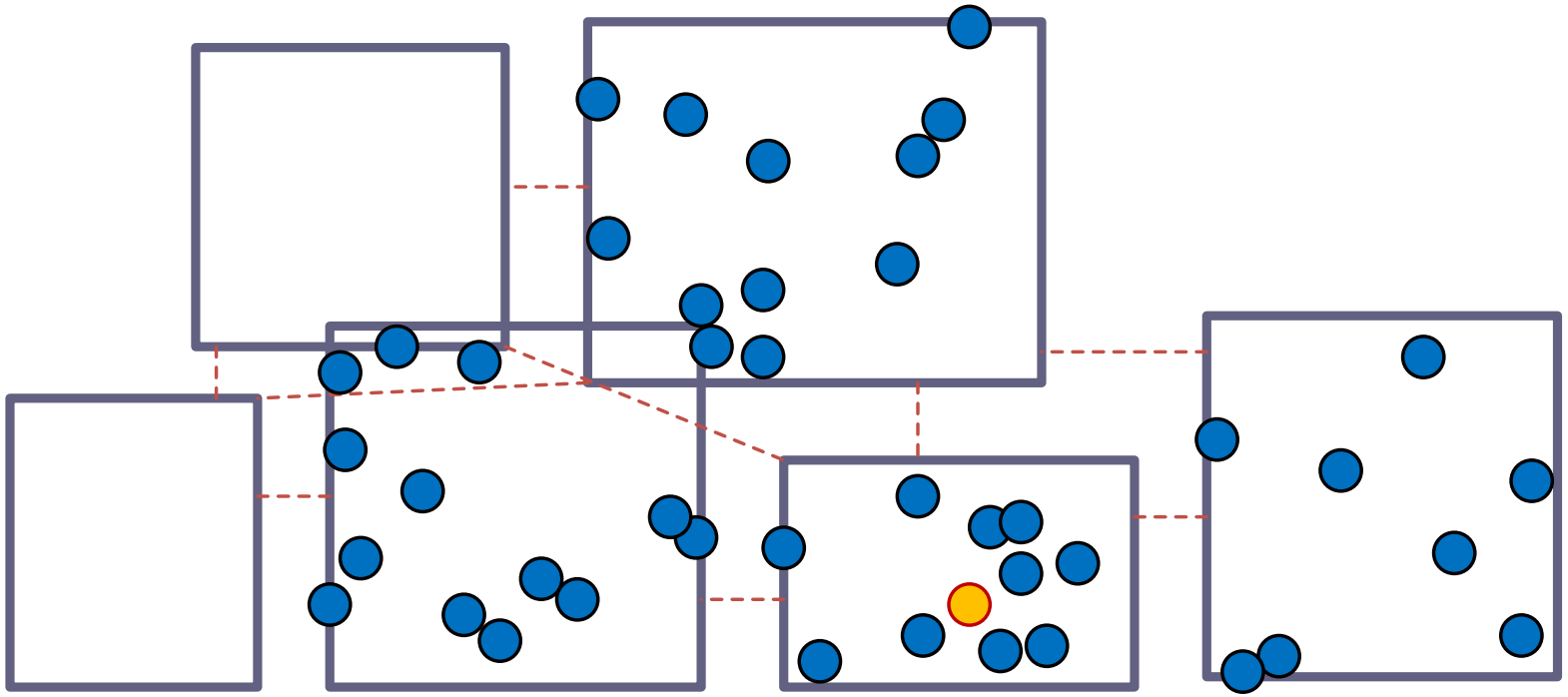
Индекс GiST



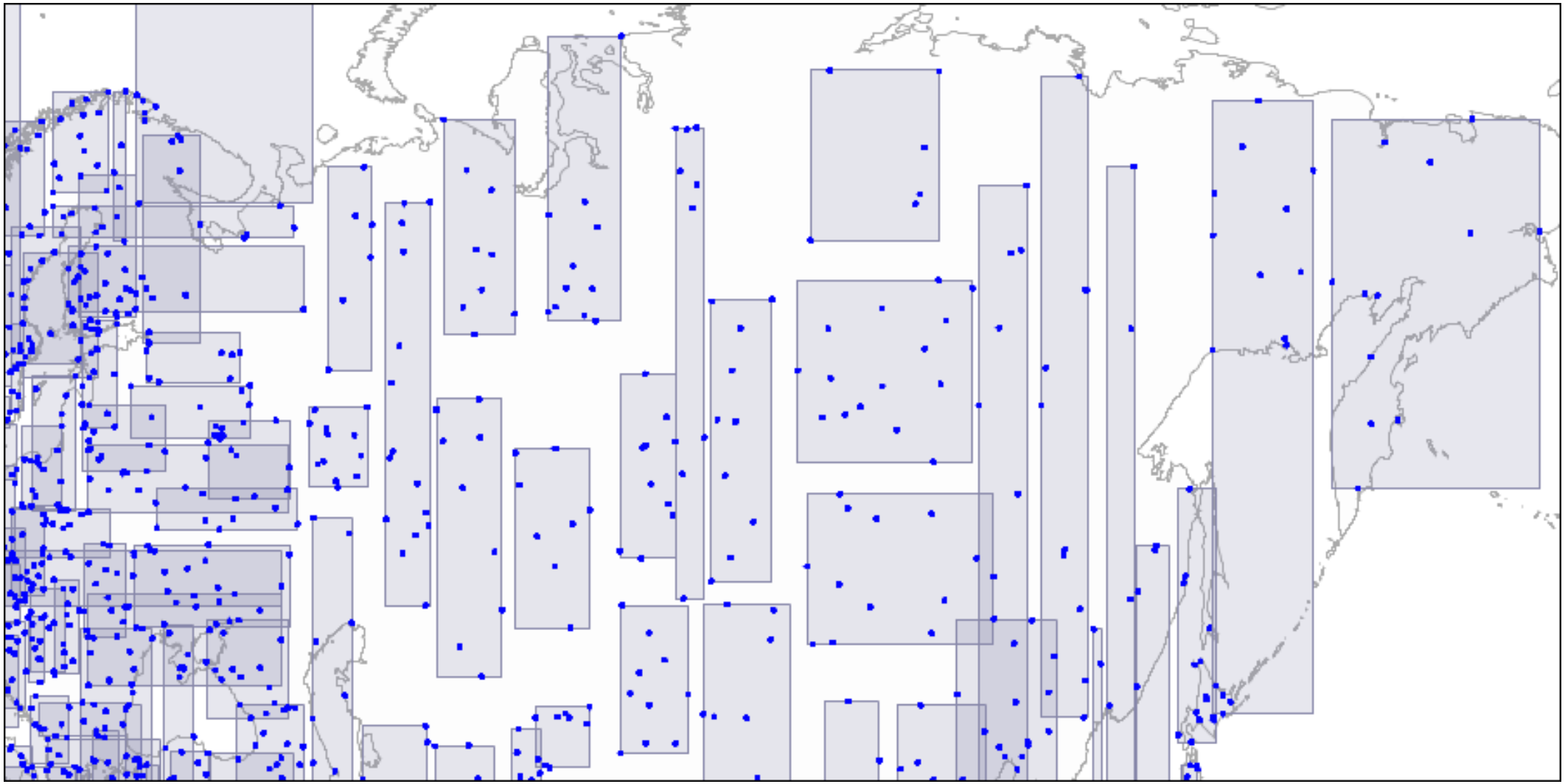
Индекс GiST



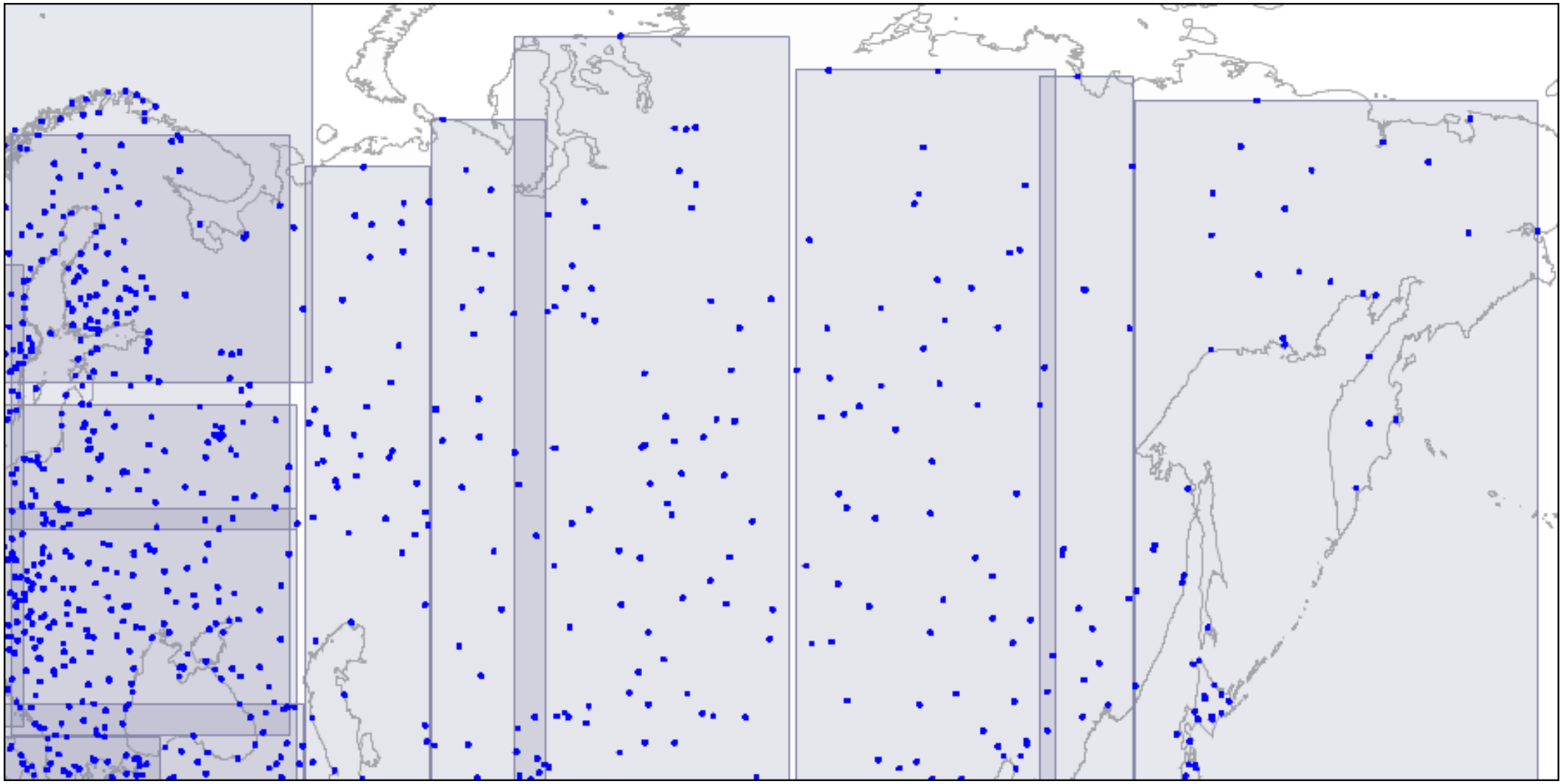
Индекс GiST



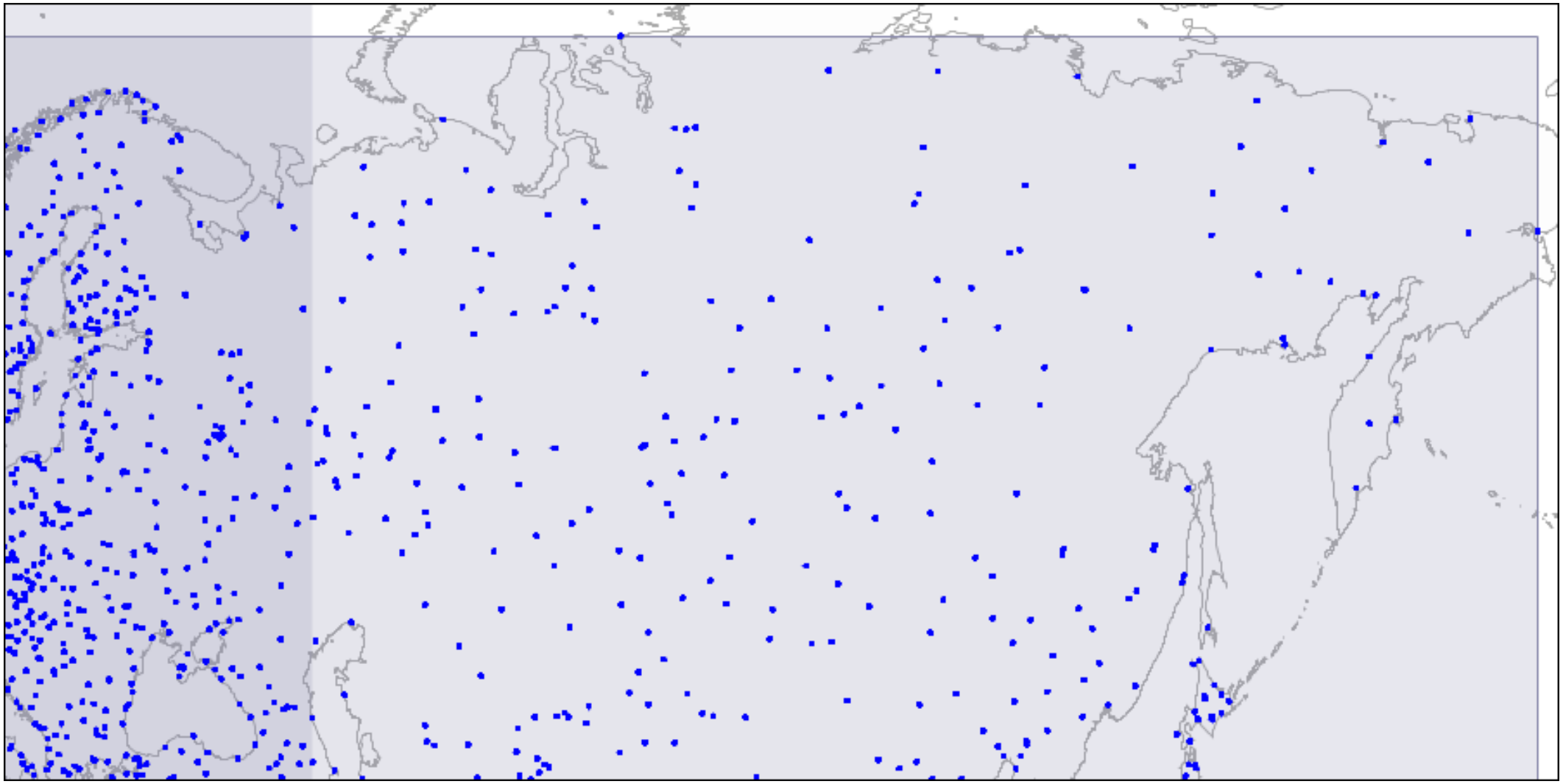
Индекс GiST



Индекс GiST



Индекс GiST



Поиск в пространстве (KNN)

id	amenity	coordinates
1	shop	(64.2, 88.3)
2	bar	(50.5, 87.4)

```
SELECT * FROM building
  ORDER BY coordinates <-> point (53.7, 87.7)
  LIMIT 10;
```

Индекс GiST: индексный поиск от произвольной точки

Оператор **<->** не поддерживается ORM

Поиск в пространстве (KNN)

```
[Sql.Expression("{0} <-> {1}", ServerSideOnly = true)]
public static double Distance(NpgsqlPoint p1, NpgsqlPoint p2) =>
    throw new NotImplementedException();
```

```
var myPoint = new NpgsqlPoint { X = 48, Y = 46 };

var result = await context.Buildings
    .Select(x => new
    {
        Data = x,
        Distance = PgExt.Distance(myPoint, x.Coordinates)
    })
    .OrderBy(x => x.Distance)
    .Take(10)
    .ToListAsync();
```

Поиск в пространстве (KNN)

	id	amenity	coordinates
	1	shop	(64.2, 88.3)
	2	bar	(50.5, 87.4)

```
SELECT x.id, x.amenity, x.coordinates
  FROM building x
  ORDER BY :myPoint <-> x.coordinates
LIMIT 10;
```

PostGIS: правильные расстояния на земном шаре

Поиск в пространстве (KNN)

Поиск от произвольной точки с поддержкой индексов

PostGIS: правильные расстояния на земном шаре

Идеально для работы с геоданными!

LINQ to DB: поиск в пространстве через ORM!

Поиск в JSONB

id	name	city

DME	Домодедово	{
		"en": "Moscow",
		"ru": "Москва",
		"fr": "Moscou"
		}

TOF	Богашёво	{
		"en": "Tomsk",
		"ru": "Томск",
		"fr": "Tomsk"
		}

Поиск в JSONB

Поиск по вхождению объекта

```
{                                    {
  "en": "Moscow",                      
  "ru": "Москва",          @>          "ru": "Москва"
  "fr": "Moscou"                     }
}
```

```
{                                    {
  "en": "Moscow",                      
  "ru": "Москва",          @>          "ru": "Томск"
  "fr": "Moscou"                     }
}
```

Поиск в JSONB

Вхождение (для поиска, поддерживаются индексы)

```
[Sql.Expression("{0} @> {1}::jsonb", ServerSideOnly = true,
    Precedence = Precedence.Comparison, IsPredicate = true)]
public static bool JsonContains(this string j1, string j2) =>
    throw new NotImplementedException();
```

Значение по ключу объекта

```
[Sql.Expression("{0}->>{1}", ServerSideOnly = true)]
public static string JsonChildText(this string json, string key) =>
    throw new NotImplementedException();
```

Поиск в JSONB

	id	name	city
	DME	Домодедово	{ "en": "Moscow", "ru": "Москва", "fr": "Moscou" }
	TOF	Богашёво	{ "en": "Tomsk", "ru": "Томск", "fr": "Tomsk" }

```
var filter = new { en = "Tomsk" };

var filterJson = JsonSerializer.Serialize(filter);

var result = await context.AirportsDatas
    .Where(x => x.City.JsonContains(filterJson))
    .Select(x => new { x.AirportName, City = x.City.JsonChildText("ru") })
    .FirstOrDefaultAsync();
```

Поиск в JSONB

	id	name	city
	DME	Домодедово	{ "en": "Moscow", "ru": "Москва", "fr": "Moscou" }
	TOF	Богашёво	{ "en": "Tomsk", "ru": "Томск", "fr": "Tomsk" }

```
SELECT x.airport_name, x.city->>'ru'
FROM airports_data x
WHERE x.city @> :filterJson::jsonb
LIMIT :take
```

Продвинутый поиск в JSONB

Расширение jsquery

```
SELECT * FROM js_test WHERE value @@
    '%(NOT #(NOT ($ >= 0 AND $ <= 1)) AND $ IS ARRAY)'::jsquery;
```

Продвинутый поиск в JSONB

```
[Sql.Expression("{0} @@ {1}::jsquery", ServerSideOnly = true,
    Precedence = Precedence.Comparison, IsPredicate = true)]
public static bool MatchesJsquery(this string data, string query) =>
    throw new NotImplementedException();
```

```
var result = await context.Items
    .Where(x => x.Data.MatchesJsquery("#:.%:($ >= 0 AND $ <= 1)"))
    .ToListAsync();
```

```
SELECT * FROM items x
WHERE x.data @@ '#:.%:($ >= 0 AND $ <= 1)'::jsquery
```

Найти что-нибудь красное

https://2015.codefest.ru/lecture/1000

```
      Table "public.js_test"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer | not null
 value  | jsonb   |

select * from js_test;
 id |                                        value
----+-----------------------------------------------------------------------
  1 | [1, "a", true, {"b": "c", "f": false}]
  2 | {"a": "blue", "t": [{"color": "red", "width": 100}]}
  3 | [{"color": "red", "width": 100}]
  4 | {"color": "red", "width": 100}
  5 | {"a": "blue", "t": [{"color": "red", "width": 100}], "color": "red"}
  6 | {"a": "blue", "t": [{"color": "blue", "width": 100}], "color": "red"}
  7 | {"a": "blue", "t": [{"color": "blue", "width": 100}], "colr": "red"}
  8 | {"a": "blue", "t": [{"color": "green", "width": 100}]}
  9 | {"color": "green", "value": "red", "width": 100}
(9 rows)
```

Найти что-нибудь красное

```
      Table "public.js_test"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer | not null
 value  | jsonb   |
```

```
SELECT * FROM js_test
  WHERE value @@ '*.color="red"'::jsquery;
```

```
select * from js_test;
```

```
 id |                                        value
----+-----------------------------------------------------------------------
  1 | [1, "a", true, {"b": "c", "f": false}]
  2 | {"a": "blue", "t": [{"color": "red", "width": 100}]}
  3 | [{"color": "red", "width": 100}]
  4 | {"color": "red", "width": 100}
  5 | {"a": "blue", "t": [{"color": "red", "width": 100}], "color": "red"}
  6 | {"a": "blue", "t": [{"color": "blue", "width": 100}], "color": "red"}
  7 | {"a": "blue", "t": [{"color": "blue", "width": 100}], "colr": "red"}
  8 | {"a": "blue", "t": [{"color": "green", "width": 100}]}
  9 | {"color": "green", "value": "red", "width": 100}
(9 rows)
```

Продвинутый поиск в JSONB

```
[Sql.Expression("{0} @@ '*.{1}=\"{2}\"'::jsquery",
    ServerSideOnly = true, Precedence = Precedence.Comparison, IsPredicate = true)]
public static bool JsonContainsAnywhere(this string j, string k, string v) =>
    throw new NotImplementedException();
```

Продвинутый поиск в JSONB

```
[Sql.Expression("{0} @@ '*.{1}=\"{2}\"'::jsquery",
    ServerSideOnly = true, Precedence = Precedence.Comparison, IsPredicate = true)]
public static bool JsonContainsAnywhere(this string j, string k, string v) =>
    throw new NotImplementedException();
```

```
[Sql.Expression("{0} @@ ('*.' || {1} || '=\"' || {2} || '\"')::jsquery",
    ServerSideOnly = true, Precedence = Precedence.Comparison, IsPredicate = true)]
public static bool JsonContainsAnywhere(this string j, string k, string v) =>
    throw new NotImplementedException();
```

Продвинутый поиск в JSONB

Найти что-нибудь красное

```
var result = await context.Items
    .Where(x =>
x.Data.JsonContainsAnywhere("color", "red"))
    .ToListAsync();
```

```
SELECT * FROM items x
WHERE x.data @@
    ('*.' || 'color' || '="' || 'red' || '"')::jsquery
```

Продвинутый поиск в JSONB

Найти что-нибудь красное

```
var result = await context.Items
    .Where(x =>
x.Data.JsonContainsAnywhere("color", "red"))
    .ToListAsync();
```

Найти что-нибудь русское

```
var result = await context.Items
    .Where(x =>
x.Data.JsonContainsAnywhere("lang", "ru"))
    .ToListAsync();
```

JSONPath: поиск в JSONB

```
[Sql.Expression("{0} @? {1}::jsonpath", ServerSideOnly = true,
    Precedence = Precedence.Comparison, IsPredicate = true)]
public static bool MatchesJsonPath(this string data, string query) =>
    throw new NotImplementedException();
```

```
var result = await context.Items
    .Where(x => x.Data.MatchesJsonPath("$.a[*] ? (@ > 2)"))
    .ToListAsync();
```

```
SELECT * FROM items x
WHERE x.data @? '$.a[*] ? (@ > 2)'::jsonpath
```

JSONB

Хранение данных со сложной структурой

Гибкий и функциональный поиск

Поддержка индексов

LINQ to DB: поиск в JSONB через ORM!

Агрегация в массив

`array_agg(…)`

Собирает все входные значения в массив.

	id	name	coordinates	timezone
	KRO	Курган	(65.416,55.475)	Asia/Yekaterinburg
	AER	Сочи	(37.906,55.409)	Europe/Moscow

Пример задачи:

- Таблица аэропортов
- Сгруппировать по часовым поясам
- Для каждого получить массив координат

Агрегация в массив

```
[Sql.Expression("array_agg({1})", ServerSideOnly = true, IsAggregate = true)]
public static TV[] ArrayAgg<TEntity, TK, TV>(this IGrouping<TK, TEntity> g,
        Expression<Func<TEntity, TV>> expr) =>
    throw new NotImplementedException();
```

Агрегация в массив

	id	name	coordinates	timezone
	KRO	Курган	(65.416,55.475)	Asia/Yekaterinburg
	AER	Сочи	(37.906,55.409)	Europe/Moscow

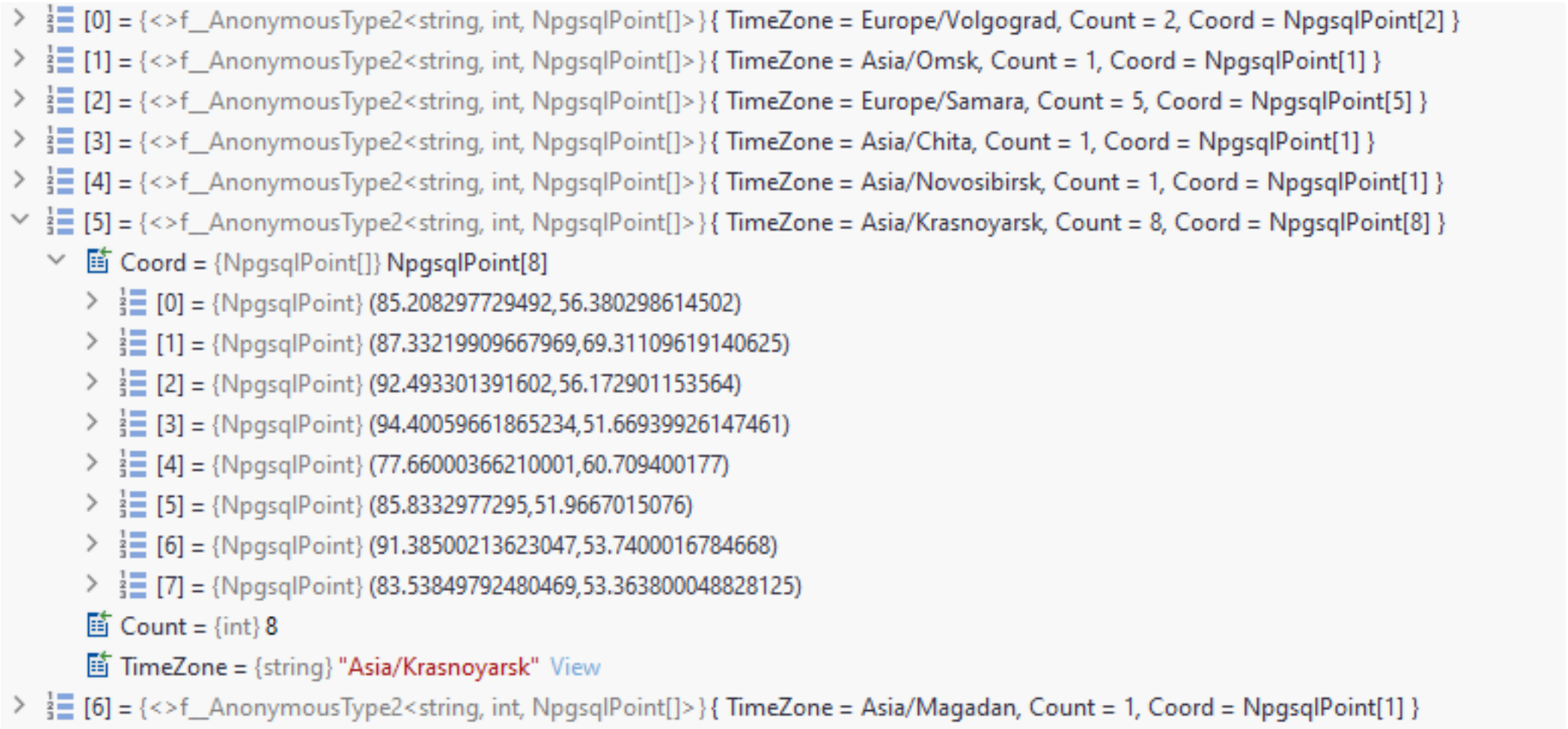
```
var result = await context.Airports
    .GroupBy(x => x.TimeZone)
    .Select(g => new
    {
        TimeZone = g.Key,
        Count = g.Count(),
        Coord = g.ArrayAgg(x => x.Coordinates)
    })
    .ToListAsync();
```

Агрегация в массив

	id	name	coordinates	timezone
	KRO	Курган	(65.416,55.475)	Asia/Yekaterinburg
	AER	Сочи	(37.906,55.409)	Europe/Moscow

```
SELECT  t1.timezone, Count(*),
        array_agg(t1.coordinates)
FROM airports t1
GROUP BY t1.timezone
```

Агрегация в массив



Агрегация в JSON

	id	name	coordinates	timezone
	KRO	Курган	(65.416,55.475)	Asia/Yekaterinburg
	AER	Сочи	(37.906,55.409)	Europe/Moscow

Пример задачи:

Таблица аэропортов

Сгруппировать по часовым поясам

Для каждого получить объект вида:

```
{
  "AER": "Сочи",
  "DME": "Домодедово",
  "GDZ": "Геленджик",
  "KZN": "Казань",
  "LED": "Пулково"
}
```

Агрегация в JSON

```
[Sql.Expression("jsonb_object_agg({1}, {2})", ServerSideOnly = true,
    IsAggregate = true)]
public static string JsonObjAgg<TEntity, TK, TV>(this IGrouping<TK, TEntity> g,
    Expression<Func<TEntity, string>> exprKey, Expression<Func<TEntity, TV>> exprVal)
=> throw new NotImplementedException();
```

Агрегация в JSON

```
var result = await context.AirportsDatas
    .GroupBy(x => x.TimeZone)
    .Select(g => new
    {
        TimeZone = g.Key,
        Count = g.Count(),
        Ids = g.JsonObjAgg(x => x.Id, x => x.AirportName.JsonChildText("ru"))
    })
    .ToListAsync();
```

```
SELECT  t1.timezone, Count(*),
        jsonb_object_agg(t1.id, t1.airport_name->>'ru')
FROM airports_data t1
GROUP BY t1.timezone
```

Агрегация в JSON

Агрегация - прочее

Postgres позволяет писать собственные агрегаты

Расширяемость LINQ to DB позволит применить их в ORM

Принцип простой расширяемости

Можно использовать весь арсенал СУБД

Даже специфические конструкции Postgres

И даже из сторонних расширений

Неограниченная гибкость

LINQ to DB

Альтернативная ORM для .NET

Богатый функционал «из коробки»

Возможность легко писать расширения

Высокая производительность

Получаем миллион

3 колонки

Сначала "прогреваем"

Отключаем логи

Получаем миллион

EF Core (AsNoTracking)

```
965.3 мс
922.6 мс
```

Dapper

```
1068.2 мс
1097.9 мс
```

LINQ to DB

```
650.7 мс
676.5 мс
```

x1.63 vs Dapper

x1.42 vs EF Core

LINQ to DB

Альтернативная ORM для .NET

Богатый функционал «из коробки»

Возможность легко писать расширения

Высокая производительность

Не готовы отказаться от EF Core?

Страшно / рисковано

Существующий проект (дорого)

Зависимости

...

linq2db.EntityFrameworkCore

Позволяет выполнять запросы LINQ to DB

Классы DbContext и Entities – для EF Core

LINQ to DB

Альтернативная ORM для .NET

Богатый функционал «из коробки»

Возможность легко писать расширения

Высокая производительность

Интеграция с EF Core

Проблемы ORM решаемы!

Встречаются плохие запросы для простых операций

- Но опытные разработчики научились писать хорошие

Ограниченность (многое не поддерживается)

- LINQ to DB решает проблему


КОНЕЦ

Ссылки

LINQ to DB

https://github.com/linq2db/linq2db

https://github.com/linq2db/linq2db.EntityFrameworkCore

https://linq2db.github.io/api/LinqToDB.DataProvider.PostgreSQL.PostgreSQLExtensions.html

https://www.nuget.org/packages/linq2db

Документация PostgreSQL на русском

https://postgrespro.ru/docs

Демонстрационная база данных

https://postgrespro.ru/education/demodb

Мои контакты

fadeevas@sibedge.com

https://vk.com/fadeev