sibedge

DOTNEXT

Полнотекстовый Поиск в PostgreSQL

Используем в .NET

Алексей Фадеев, старший разработчик .NET

О себе

Алексей Фадеев, старший разработчик .NET

Компания sibedge, г.Томск

Сфера деятельности

Backend-разработка
Работа с СУБД, оптимизация
Флагманская СУБД: PostgreSQL,

Контакты

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

Что будет интересного?

В БД есть полнотекстовый поиск (FTS)

Быстрый (индексный)

Два разных подхода

Примеры кода на C#, ORM

Бонус

Подход 1

Специальные типы данных

Операторы и функции FTS

Синтаксис запросов

База

Документ — это единица хранения в системе FTS

- Содержимое текстового поля таблицы
- Сочетание нескольких полей

tsvector – тип данных для документа

tsquery – тип данных для запроса

@@ – оператор соответствия документа запросу (TRUE/FALSE)

Документ

Хранится в виде лексем

Лексема – фрагмент слова (без окончания)

Исключаются стоп-слова (короткие, распространённые)

to_tsvector – преобразовать текст в документ

```
SELECT to_tsvector('russian', 'Конференция для разработчиков')
'конференц':1 'разработчик':3
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разработчики')
FROM texts
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разработчики')
FROM texts
```

TRUE

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разраб')
FROM texts
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разраб')
FROM texts
```

FALSE

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разраб:*')
FROM texts
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разраб:*')
FROM texts
```

TRUE

Поиск по подстроке (начинается с...)

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разраб:*')
FROM texts
```

✓ TRUE

Поиск по подстроке (начинается с...)

:*

Запрос

	text	search_vector
1	Привет, разрабы	'привет':1 'разраб':2
2	Разврат и ужас	'разврат':1 'ужас':3
3	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT text FROM s1
  WHERE search_vector @@ to_tsquery('russian', 'разраб:*')
  ORDER BY search_vector @@ to_tsquery('russian', 'разработчик') DESC
```

Запрос

	text	search_vector
1	Привет, разрабы	'привет':1 'разраб':2
2	Разврат и ужас	'разврат':1 'ужас':3
3	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT text FROM s1
  WHERE search_vector @@ to_tsquery('russian', 'разраб:*')
  ORDER BY search_vector @@ to_tsquery('russian', 'разработчик') DESC
```

```
Конференция для разработчиков

Привет, разрабы
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разработчики | девопсы')
FROM texts
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разработчики | девопсы')
FROM texts
```

TRUE

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разработчики & девопсы')
FROM texts
```

FALSE

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ to_tsquery('russian', 'разработчики & девопсы')
FROM texts
```

Эквивалентно:

```
SELECT
  search_vector @@ plainto_tsquery('russian', 'разработчики девопсы')
FROM texts
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ plainto_tsquery('russian', 'разработчики на конференциях')
FROM texts
```

Запрос

	text	search_vector
	Конференция для разработчиков	'конференц':1 'разработчик':3

```
SELECT
  search_vector @@ plainto_tsquery('russian', 'разработчики на конференциях')
FROM texts
```

TRUE

Таблица с текстом

Данные: из электронных книг (1 ГБ с небольшим)

```
CREATE TABLE paragraph
(
  id integer PRIMARY KEY NOT NULL,
  book_id integer NOT NULL,
  content text NOT NULL
)
```

Таблица с текстом

Данные: из электронных книг (1 ГБ с небольшим)

```
CREATE TABLE paragraph
(
  id integer PRIMARY KEY NOT NULL,
  book_id integer NOT NULL,
  content text NOT NULL
)
```

Где тут tsvector?

Таблица с tsvector

Вычисляемый столбец

```
CREATE TABLE paragraph
(
  id integer PRIMARY KEY NOT NULL,
  book_id integer NOT NULL,
  content text NOT NULL,
  search_vector tsvector GENERATED ALWAYS AS
    (to_tsvector('russian', content)) STORED
)
```

Поле автоматически создаётся при INSERT
и меняется при UPDATE

Индекс

```
CREATE INDEX ON paragraph USING gin (search_vector);
```

FTS поиск

В таблице ~ 1.1 млн строк

```
SELECT COUNT(*) FROM paragraph
  WHERE search_vector @@ to_tsquery('russian', 'собака');
```

```
count: 4385
```

Без индекса	С индексом
450-500 мс	5-6 мс

FTS поиск

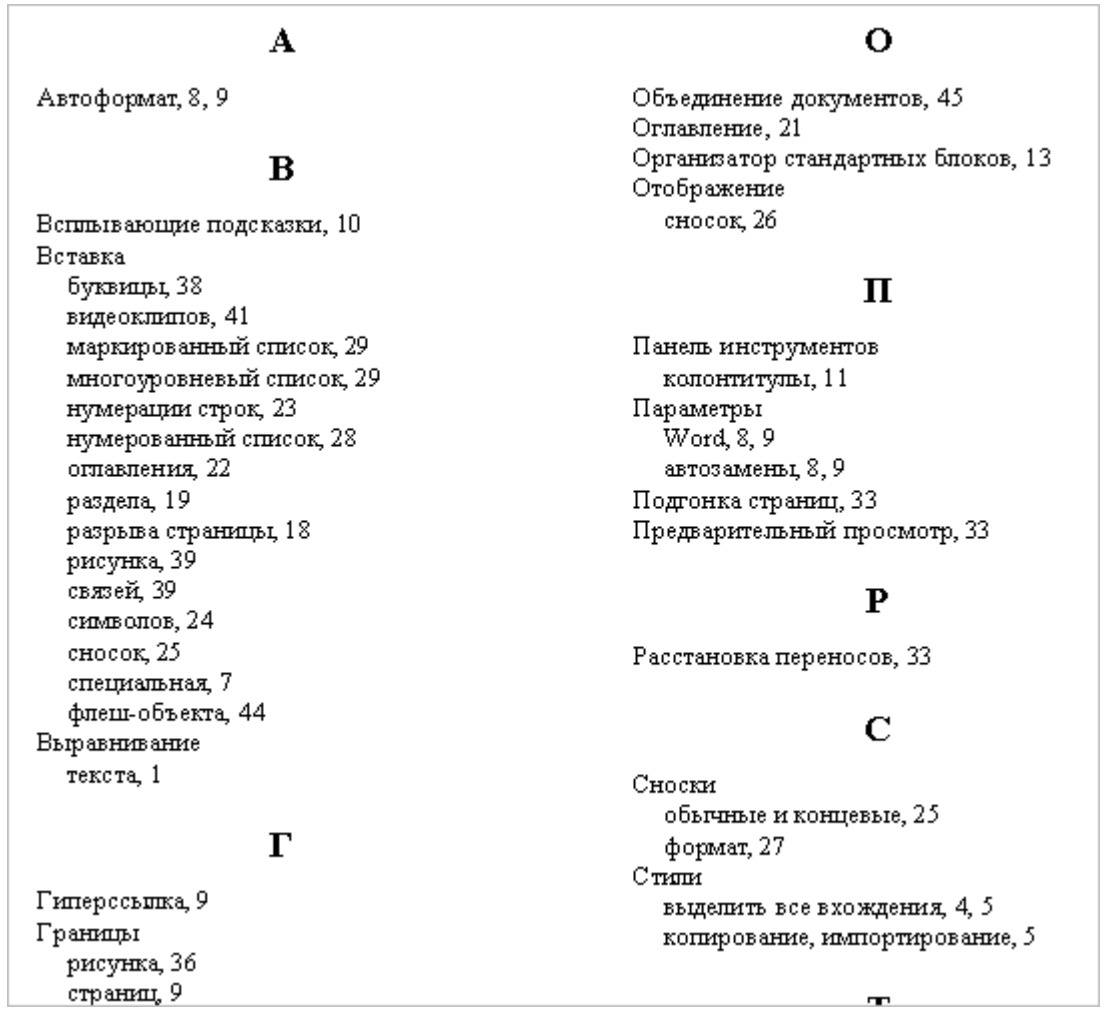
В таблице ~ 1.1 млн строк

```
SELECT COUNT(*) FROM paragraph
  WHERE search_vector @@ to_tsquery('russian', 'телевизор');
```

```
count: 6
```

Без индекса	С индексом
450-500 мс	0.15-0.2 мс
	x3000

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

Индексы GIN и GiST

GIN:	GiST:
Поиск работает быстрее	Быстрее операции обновления
Фразовый поиск, ранжирование	Компактнее

Индексы GIN и GiST

GIN:	GiST:
Поиск работает быстрее	Быстрее операции обновления
Фразовый поиск, ранжирование	Компактнее

По умолчанию используем **GIN**

EF Core

Конфигурирование

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    var dataSourceBuilder = new NpgsqlSlimDataSourceBuilder(_connectionString);
    dataSourceBuilder.EnableFullTextSearch();

    var dataSource = dataSourceBuilder.Build();
    optionsBuilder.UseNpgsql(dataSource);
}
```

EF Core

```
var matched = await db.Paragraphs
    .Where(x => x.SearchVector.Matches(
        EF.Functions.PlainToTsQuery("russian", "кошка с собакой")))
    .ToListAsync();
```

```
SELECT p.id, p.book_id, p.content, p.search_vector
    FROM paragraph AS p
    WHERE p.search_vector @@ plainto_tsquery('russian', 'кошка с собакой')
```

LINQ to DB

Нет функций и операторов FTS из коробки

Пишем расширения

LINQ to DB – v1

```
[Sql.Expression("{0} @@ {1}", ServerSideOnly = true, PreferServerSide = true)]
public static bool Matches(this NpgsqlTsVector vector, NpgsqlTsQuery query)
    => throw new InvalidOperationException("Только для трансляции в SQL.");

[Sql.Function("plainto_tsquery", ServerSideOnly = true)]
public static NpgsqlTsQuery PlainToTsQuery([SqlQueryDependent] string config,
        [SqlQueryDependent] string query)
    => throw new InvalidOperationException("Только для трансляции в SQL.");
```

```
var matched = await db.Paragraphs
    .Where(x => x.SearchVector.Matches(
        NpgsqlFullTextExtensions.PlainToTsQuery("russian", "кошка с собакой")))
    .ToListAsync();
```

LINQ to DB – v1

```
var matched = await db.Paragraphs
    .Where(x => x.SearchVector.Matches(
        NpgsqlFullTextExtensions.PlainToTsQuery("russian", "кошка с собакой")))
    .ToListAsync();
```

```
SELECT x.id, x.book_id, x.content, x.search_vector
    FROM paragraph x
    WHERE x.search_vector @@ plainto_tsquery('russian', 'кошка с собакой')
```

LINQ to DB – v2

```
[Sql.Expression("{0} @@ plainto_tsquery({1}, {2})", ServerSideOnly = true)]
public static bool MatchPlain(this NpgsqlTsVector vector, string lang, string query)
    => throw new InvalidOperationException("Только для трансляции в SQL.");


var matched = await db.Paragraphs
    .Where(x => x.SearchVector.MatchPlain("russian", "кошка с собакой"))
    .ToListAsync();
```

LINQ to DB – v3

```
[Sql.Expression("{0} @@ plainto_tsquery('russian', {1})", ServerSideOnly = true)]
public static bool MatchPlainRus(this NpgsqlTsVector vector, string query)
    => throw new InvalidOperationException("Только для трансляции в SQL.");


var matched = await db.Paragraphs
    .Where(x => x.SearchVector.MatchPlainRus("кошка с собакой"))
    .ToListAsync();
```

Поиск фраз

Важен порядок слов

phraseto_tsquery – функция, учитывающая порядок слов

<2> – оператор расстояния между словами

Запрос

	text	search_vector
	Конференция для крутых разработчиков	'конференц':1 'крут':3 'разработчик':4

```
SELECT
  search_vector @@ plainto_tsquery('russian', 'конференция для разработчика')
FROM texts
```

TRUE

Запрос

	text	search_vector
	Конференция для крутых разработчиков	'конференц':1 'крут':3 'разработчик':4

```
SELECT
  search_vector @@ phraseto_tsquery('russian', 'конференция для разработчика')
FROM texts
```

FALSE

Запрос

	text	search_vector
	Конференция для крутых разработчиков	'конференц':1 'крут':3 'разработчик':4

```
SELECT
  search_vector @@ phraseto_tsquery('russian',
      'конференция для крутого разработчика')

FROM texts
```

TRUE

Запрос

	text	search_vector
	Конференция для крутых разработчиков	'конференц':1 'крут':3 'разработчик':4

```
SELECT
  search_vector @@ phraseto_tsquery('russian',
      'конференция про крутого разработчика')

FROM texts
```

TRUE

EF Core

```
var matched = await db.Paragraphs
    .Where(x => x.SearchVector.Matches(EF.Functions.PhraseToTsQuery("russian",
        "конференция для крутого разработчика")))
    .ToListAsync();
```

```
SELECT * FROM paragraph AS p
    WHERE p.search_vector @@ phraseto_tsquery('russian',
        'конференция для крутого разработчика')
```

LINQ to DB

```
[Sql.Expression("{0} @@ phraseto_tsquery('russian', {1})", ServerSideOnly = true)]
public static bool MatchPhraseRus(this NpgsqlTsVector vector, string query)
    => throw new InvalidOperationException("Только для трансляции в SQL.");
```

```
var matched = await db.Paragraphs
    .Where(x => x.SearchVector.MatchPhraseRus("конференция для крутого разработчика"))
    .ToListAsync();
```

Запрос

	text	search_vector
	Конференция для крутых разработчиков	'конференц':1 'крут':3 'разработчик':4

Ищем:

конференция для [...] разработчика

Запрос

	text	search_vector
	Конференция для крутых разработчиков	'конференц':1 'крут':3 'разработчик':4

```
SELECT
  search_vector @@ to_tsquery('russian',
      'конференция <3> разработчика')

FROM texts
```

TRUE

EF Core

```
var matched = await db.Paragraphs
    .Where(x => x.SearchVector.Matches(EF.Functions.ToTsQuery("russian",
        "конференция <3> разработчика")))
    .ToListAsync();
```

```
SELECT *
    FROM paragraph AS p
    WHERE p.search_vector @@ to_tsquery('russian',
        'конференция <3> разработчика')
```

LINQ to DB

```
[Sql.Expression("{0} @@ to_tsquery('russian', {1} || ' <{2}> ' || {3})",
    ServerSideOnly = true)]
public static bool MatchPhraseDistanceRus(this NpgsqlTsVector vector, string word1,
    int distance, string word2)
    => throw new InvalidOperationException("Только для трансляции в SQL.");


var matched = await db.Paragraphs
    .Where(x => x.SearchVector
        .MatchPhraseDistanceRus("конференция", 3, "разработчика"))
    .ToListAsync();


SELECT * FROM paragraph x
    WHERE x.search_vector @@ to_tsquery('russian',
          'конференция' || ' <3> ' || 'разработчика')
```

Альтернативные словари

Поиск с чередованием гласных (камень/камни)

Поддержка Ispell (например hunspell-ru)

Установка словаря hunspell-ru

```
CREATE TEXT SEARCH DICTIONARY russian_hunspell (
    TEMPLATE = ispell,
    DictFile = ru_ru_aot,
    AffFile = ru_ru_aot,
    Stopwords = russian);

CREATE TEXT SEARCH CONFIGURATION public.russian_hunspell (
    COPY = pg_catalog.russian);

ALTER TEXT SEARCH CONFIGURATION russian_hunspell;

ALTER MAPPING REPLACE russian_stem WITH russian_hunspell;
```

Запрос

	text	search_vector
	Время собирать камни	'врем':1 'камн':3 'собира':2

```
SELECT
  search_vector @@ to_tsquery('russian_hunspell', 'камень')
FROM texts
```

TRUE

Альтернативные словари

Поддержка синонимов

Текстовый формат словарей

ИИ поможет!

Ранжирование

Возможность задавать разные веса словам (4 веса: A, B, C, D)

Функция ts_rank_cd – считает рейтинг соответствия запросу

Можно явно задавать веса для поиска

Пример: данные товара

Храним в одном поле tsvector:

- A – название
- B – описание
- C – доп. информация

Пример: данные товара

```
CREATE TABLE good
(
  id integer PRIMARY KEY NOT NULL,
  name text NOT NULL,
  description text NOT NULL,
  additional text,
  search_vector tsvector GENERATED ALWAYS AS (
    setweight(to_tsvector('russian', name), 'A') ||
    setweight(to_tsvector('russian', description), 'B')  ||
    setweight(to_tsvector('russian', coalesce(additional, '')), 'C')
  ) STORED
)
```

Поиск с ранжированием

```
SELECT *, ts_rank_cd(search_vector, query) AS rank
FROM good,
     plainto_tsquery('russian', 'вата минеральная огнеупорная') AS query
WHERE query @@ search_vector
ORDER BY rank DESC
```

EF Core

```
var searchTerm = "вата минеральная огнеупорная";

var matched = await db.Goods
    .Where(x => x.SearchVector.Matches(EF.Functions
        .PlainToTsQuery("russian", searchTerm)))
    .OrderByDescending(d =>
        d.SearchVector.Rank(EF.Functions.PlainToTsQuery("russian", searchTerm)))
    .ToListAsync();
```

Явно задаём веса для поиска

```
SELECT * FROM good
    WHERE search_vector @@ to_tsquery('russian', 'огнеупорная:A')
```

'огнеупорная:A' – искать только в названии

'огнеупорная:AB' – искать в названии и описании

LINQ to DB

```
[Sql.Expression("{0} @@ to_tsquery('russian', {1} || ':' || {2})", ServerSideOnly = true)]
public static bool MatchWeighedRus(this NpgsqlTsVector vector, string word1, string weight)
    => throw new InvalidOperationException("Только для трансляции в SQL.");
```

```
var matched = await db.Goods
    .Where(x => x.SearchVector.MatchWeighedRus("огнеупорная", "AB"))
    .ToListAsync();
```

Пример: транзакции маркетплейса

good

	id	name	description	additional	price
	28374831	Книга 'Война и мир'	Роман Л. Толстого 'Война и мир', издательство АСТ, 2018г. Твердый переплет		500
	98457853	Ноутбук Pro	16 ГБ ОЗУ, 512 ГБ SSD, серебристый	Надежный продавец	45000
	29845754	Наушники Wireless	Функция шумоподавления, белые		3000

good_transaction

	tran_id	good_id	quantity	cost	datetime	search_vector
	1000001	28374831	2	1000	2026-09-15 10:30:00+03	…
	1000002	28908682	1	15000	2026-09-16 14:20:00+03	…
	1000003	98457853	1	45000	2026-09-17 09:15:00+03	…
	1000004	29845754	3	9000	2026-09-18 16:45:00+03	…

Подход 1 (tsvector) - итоги

Гибкий полнотекстовый поиск

Требуется порог входа

Требуется поддержка на стороне ORM

Подход 2

Привычная конструкция:

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%собак%';
```

Подход 2

Привычная конструкция:

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%собак%';
```

Может ли быть индексным?

Триграммы

```
CREATE EXTENSION pg_trgm;
```

Триграммы

Построить индекс GIN:

```
CREATE INDEX ON paragraph
  USING gin (content gin_trgm_ops);
```

Поиск LIKE

В таблице ~ 1.1 млн строк

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%собак%';
```

Без индекса	С индексом
2.5-3 с	30-80 мс
	x80

Поиск LIKE

В таблице ~ 1.1 млн строк

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%собак%';
```

Поиск по подстроке

Поиск LIKE

В таблице ~ 1.1 млн строк

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%собак%';
```

В документе:	Ищем	
собаки	собака	

Поиск LIKE

В таблице ~ 1.1 млн строк

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%...%';
```

В документе:	Ищем
экспрезидент	президент

Поиск LIKE

В таблице ~ 1.1 млн строк

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%...%';
```

В документе:	Ищем	
экспрезидент	президент	
нехороший	хороший	

Поиск LIKE

В таблице ~ 1.1 млн строк

```
SELECT COUNT(1) FROM paragraph
  WHERE content ILIKE '%собак%';
```

Поиск по подстроке

Можно ускорить не меняя код

EF Core

```
var matched = await db.Paragraphs
    .Where(x => EF.Functions.ILike(x.Content, "%собак%"))
    .ToListAsync();
```

LINQ to DB

```
[Sql.Expression("({0} ILIKE '%' || {1} || '%')", ServerSideOnly = true)]
public static bool ILike(this string column, string pattern)
    => throw new InvalidOperationException("Только для трансляции в SQL.");


var matched = await db.Paragraphs
    .Where(x => x.Content.ILike("собак"))
    .ToListAsync();
```

Объём

В таблице ~ 1.1 млн строк

Размер таблицы	Размер индекса tsvector	Размер индекса триграмм	Размер PK
565 MB	86 MB	279 MB	24 MB

Поиск фраз – сравнение подходов

```
EXPLAIN ANALYZE
SELECT * FROM paragraph
  WHERE search_vector @@ to_tsquery('russian', 'кошка <2> собакой');
-- "Execution Time: 1.689 ms"


EXPLAIN ANALYZE
  SELECT * FROM paragraph
  WHERE content ILIKE '%кошка с собакой%'
--"Execution Time: 32.654 ms"
```

Поиск фразы по подстроке

```
EXPLAIN ANALYZE
SELECT * FROM
(SELECT * FROM paragraph
  WHERE search_vector @@ phraseto_tsquery('russian', 'кошка с собакой')) s
  WHERE content ILIKE '%кошка с собакой%';
--"Execution Time: 4.678 ms"
```

Подход 2 (триграммы) - итоги

Поиск по подстроке (недополнотекстовый)

Медленнее, чем tsvector/tsquery

Больше размер индекса

Проще для разработки

Применимо к старому коду

Не зависит от языка (локали)

Нечеткий поиск (k-NN)

Расширение pg_trgm

Для строковых типов

```
SELECT 'посадить' <-> 'подсадить'
```

```
0.416
```

```
SELECT 'тролль' <-> 'королева'
```

```
0.933
```

Год назад здесь

k-NN для геоданных

Или как найти ближайший бар

Поиск с опечатками (k-NN)

```
SELECT author, author <-> 'толстый' as dist FROM book
  ORDER BY dist
  LIMIT 5;
```

Поиск с опечатками (k-NN)

```
SELECT author, author <-> 'толстый' as dist FROM book
  ORDER BY dist
  LIMIT 5;
```

	author	dist
1	Лев Николаевич Толстой	0.807692
2	Алексей Николаевич Толстой	0.833333
3	Томас Манн	0.882353
4	Томас Гарди	0.888889
5	Антология	0.941177

Поиск с опечатками (k-NN)

```
SELECT author, author <-> 'николаевич' as dist FROM book
  ORDER BY dist
  LIMIT 9;
```

	author	dist
1	Лев Николаевич Толстой	0.521739
2	Алексей Николаевич Толстой	0.592593
3	Николай Васильевич Гоголь	0.678571
4	Николай Алексеевич Некрасов	0.689655
5	Николай Семенович Лесков	0.714286
6	Николай Гаврилович Чернышевский	0.771429
7	Александр Сергеевич Грибоедов; Александр Васильевич Сухово-Кобылин; Александр Николаевич Островский	0.84507
8	Дмитрий Андреевич Фурманов; Александр Серафимович Серафимович; Николай Алексеевич Островский	0.863636
9	Адам Мицкевич	0.863636

LINQ to DB

```
[Sql.Expression("{0} <-> {1}", ServerSideOnly = true)]
public static double WordDistance(string s1, string s2) =>
    throw new InvalidOperationException("Только для трансляции в SQL.");


var matched = await db.Books
    .Select(x => new
    {
        Data = x,
        Distance = FullTextExtensions.WordDistance(x.Author, "Толстый")
    })
    .OrderBy(x => x.Distance)
    .Take(10)
    .ToListAsync();
```

Индекс GiST

В таблице ~ 1 млн строк

```
CREATE INDEX ON items
  USING gist (name gist_trgm_ops);
```

Без индекса	С индексом
2.5–4 с	3–60 мс

Postgres FTS vs Open/Elastic Search

Postgres – базовый функционал, Elastic Search – мощные возможности:

- Поиск фраз
- Нечёткий поиск
- Ранжирование
- Языковая поддержка

Open/Elastic Search – встроенный шардинг

Postgres FTS

Основной базовый функционал

100% актуальность данных - ACID

Не требует дополнительной инфраструктуры

Postgres FTS

Когда использовать:

- Данные большие, но не огромные
- Базовых возможностей достаточно

Полезное

FTS в Postgres:

- Не заменит Elastic Search, но тоже классный

tsvector, tsquery:

- Гибкий полнотекстовый поиск
- Поиск фраз, ранжирование, языки

Триграммы, ILike:

- Максимально просто
- Легко ускорить, не трогая код

Триграммы, k-NN:

- Нечеткий поиск (опечатки)


КОНЕЦ

Ссылки

Документация PostgreSQL на русском

https://postgrespro.ru/docs

Мои контакты

fadeevas@sibedge.com

https://vk.com/fadeev