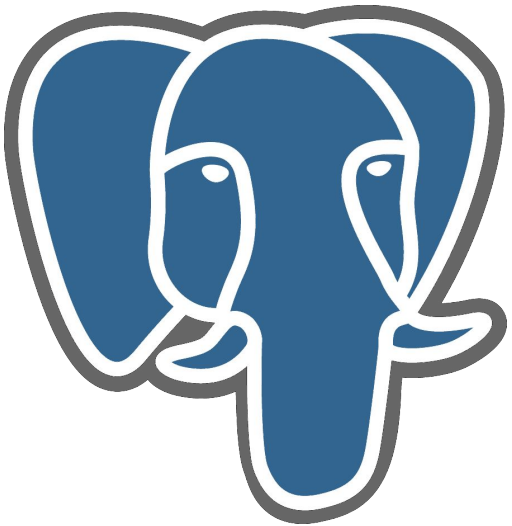
Средства работы с текстовой информацией PostgreSQL



Мамаев Алексей
Пичугин Владислав
ИУ9-52Б

СУБД PostgreSQL

Post In**gres**

Свободная объектно-реляционная система управления базами данных.

Первый выпуск – 8 июля 1996.

Реализована для большинства UNIX-подобных платформ и ОС семейства Microsoft Windows.

Ограничения

Максимальный размер базы данных	Нет ограничений
Максимальный размер таблицы	32 Тбайт
Максимальный размер записи	1,6 Тбайт
Максимальный размер поля	1 Гбайт
Максимум записей в таблице	Нет ограничений
Максимум полей в записи	250—1600, в зависимости от типов полей
Максимум индексов в таблице	Нет ограничений

Текстовые типы данных

Общего назначения:

- `character varying, varchar`
- `character, char`
- `text`
- `citext`

Системные:

- `“char”`
- `name`

Строки ограниченной переменной длины

`character varying(n), varchar(n)` $1 \le n < \ldots$

Хранят строки до n символов.

Верхняя граница n не определена, максимальный размер строки – 1 GB.

Длина сохраняемой строки меньше n – сохранение короткой строки; больше n – ошибка.

Явное приведение – удаление лишних символов.

`character varying` без аргументов – строка произвольной длины.

Строки фиксированной длины

`character(n), char(n)` $1 \le n < \ldots$

Хранят строки n символов.

Длина сохраняемой строки меньше n – сохранение короткой строки, дополненной пробелами.

`character` равносильно `character(1)`

```
CREATE TABLE test2 (b varchar(5));
INSERT INTO test2 VALUES ('ok');
INSERT INTO test2 VALUES ('good      ');
INSERT INTO test2 VALUES ('too long');
ОШИБКА:  значение не умещается в тип character varying(5)
INSERT INTO test2 VALUES ('too long'::varchar(5)); -- явное усечение
SELECT b, char_length(b) FROM test2;
```

```
   b    | char_length
-------+-------------
 ok     |           2
 good   |           5
 too l  |           5
```

Строки неограниченной длины

`text, citext`

Не описаны в стандарте SQL.

`citext` – дополнительно поставляемый модуль, регистронезависимое сравнение.

Специальные символьные типы

“char”, name

name – фиксированная строка из 64 байт, хранит идентификаторы во внутренних системных таблицах.

“char” – единичный символ, хранится в одном байте.

Не предназначены для обычного пользователя.

Полнотекстовый поиск

Полнотекстовый поиск — это возможность находить документы на естественном языке, соответствующие запросу, и, возможно, дополнительно сортировать их по релевантности для этого запроса.

В самом простом случае **запросом** считается набор слов, а **соответствие** определяется частотой слов в документе.

Операторы текстового поиска

- LIKE
- ILIKE
- SIMILAR TO
- Операторы регулярных выражений POSIX

LIKE

Синтаксис:

```
строка LIKE шаблон
```

Несколько примеров:

```
'abc' LIKE 'abc'    true

'abc' LIKE 'a%'     true

'abc' LIKE '_b_'    true

'abc' LIKE 'c'      false
```

ILIKE

Вместо LIKE можно использовать ключевое слово **ILIKE**, чтобы поиск был **регистр-независимым** с учётом текущей языковой среды.

Этот оператор **не описан** в стандарте SQL; это расширение PostgreSQL.

SIMILAR TO

Синтаксис:

```
строка SIMILAR TO шаблон
```

Несколько примеров:

```
'abc' SIMILAR TO 'abc'      true

'abc' SIMILAR TO 'a'        false

'abc' SIMILAR TO '%(b|d)%'  true

'abc' SIMILAR TO '(b|c)%'   false
```

Операторы регулярных выражений POSIX

Оператор	Описание
`~`	Проверяет *соответствие* регулярному выражению *с учётом* регистра
`~*`	Проверяет *соответствие* регулярному выражению *без учёта* регистра
`!~`	Проверяет *несоответствие* регулярному выражению *с учётом* регистра
`!~*`	Проверяет *несоответствие* регулярному выражению *без учёта* регистра

Недостатки

- Регулярные выражения не рассчитаны на работу со **словоформами**
- Они не позволяют упорядочивать результаты поиска (по релевантности)
- Они обычно выполняются медленно из-за отсутствия индексов

Полнотекстовая индексация

Состоит из двух этапов:

1. Предварительная обработка
    a. Разбор документов на фрагменты
    b. Преобразование фрагментов в лексемы
    c. Хранение документов в форме, подготовленной для поиска

2. Сохранение индекса

Документ

Документ — это единица обработки в системе полнотекстового поиска.

В контексте поиска в PostgreSQL документ — это обычно содержимое текстового поля в строке таблицы или объединение таких полей.

Документы также можно хранить в обычных текстовых файлах в файловой системе.

Разбор документов на фрагменты

Выделение различных классов **фрагментов**, например, чисел, слов, словосочетаний, почтовых адресов и т. д., которые будут обрабатываться по-разному.

Эту операцию в PostgreSQL выполняет **парсер**.

Преобразование фрагментов в лексемы

Лексема — нормализованный фрагмент, в котором разные словоформы приведены к одной.

Слова приводят к нижнему регистру, убирают окончания.

Словари

- Определение лексем
- Удаление стоп-слов
- Сопоставление синонимов
- Сопоставление словосочетаний

Существует возможность создания собственных словарей.

Хранение документов

Каждый документ может быть представлен в виде отсортированного массива нормализованных **лексем**.

Для хранения подготовленных документов в PostgreSQL предназначен тип данных **tsvector**.

Хранение поисковых запросов

Для представления запросов в PostgreSQL имеется тип данных **tsquery**.

Значение tsquery - искомые **лексемы**, объединяемые **логическими операторами** (для группировки можно использовать **скобки**):

- & (И)
- | (ИЛИ)
- ! (НЕ)
- <N> (*N* — целочисленная константа, задающая расстояние между двумя искомыми лексемами)
- <-> (ПРЕДШЕСТВУЕТ) ⇔ <1>

Пример

```
SELECT 'fat & (rat | cat)'::tsquery;

          tsquery

---------------------------

 'fat' & ( 'rat' | 'cat' )
```

Оператор соответствия @@

Возвращает булево значение.

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

 ?column?
----------
 t

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

 ?column?
----------
 f
```

Варианты использования

```
tsvector @@ tsquery
tsquery  @@ tsvector

text @@ tsquery   ⇔ to_tsvector(text) @@ tsquery
text @@ text      ⇔ to_tsvector(text) @@ plainto_tsquery(text)
```

```
SELECT 'fat cats ate fat rats' @@ 'fat & rat'

 ?column?
----------
 t
```

Поиск в таблице

```
SELECT title
FROM pgweb
WHERE to_tsvector(body) @@ to_tsquery('friend');
```

```
SELECT title
FROM pgweb
WHERE to_tsvector(title || ' ' || body) @@ to_tsquery('create &
table');
```

Разбор документов

```
to_tsvector([конфигурация regconfig,] документ text) returns tsvector
```

- Разбор текстового документа на фрагменты
- Удаление стоп-слов
- Преобразование фрагментов к лексемам

```
SELECT to_tsvector('english', 'a fat  cat sat on a mat - it ate a fat
rats');
                  to_tsvector
-----------------------------------------------------
 'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

Разбор запросов

```
to_tsquery([конфигурация regconfig,] текст_запроса text) returns tsquery
```

- Входные данные должны соответствовать синтаксису tsquery
- Удаление стоп-слов
- Преобразование фрагментов к лексемам

```
SELECT to_tsquery('english', 'The & Fat & Rats');

  to_tsquery
---------------
 'fat' & 'rat'
```

Разбор запросов

```
plainto_tsquery([конфигурация regconfig,] текст_запроса text) returns tsquery
```

- Разбор текста запроса на фрагменты
- Удаление стоп-слов
- Преобразование фрагментов к лексемам
- Вставка между лексемами оператора & (И)

```
SELECT plainto_tsquery('english', 'The Fat Rats');

 plainto_tsquery
-----------------
 'fat' & 'rat'
```

Разбор запросов

```
phraseto_tsquery([конфигурация regconfig,] текст_запроса text) returns tsquery
```

- Разбор текста запроса на фрагменты
- Удаление стоп-слов
- Преобразование фрагментов к лексемам
- Вставка между лексемами оператора <-> (ПРЕДШЕСТВУЕТ)

```
SELECT phraseto_tsquery('english', 'The Fat Rats');

 phraseto_tsquery
------------------
 'fat' <-> 'rat'
```

Оператор конкатенации векторов

Синтаксис:

```
tsvector || tsvector
```

Оператор конкатенации значений tsvector возвращает вектор, объединяющий лексемы и позиционную информацию двух векторов, переданных ему в аргументах.

Задание веса вектору

Синтаксис:

```
setweight(вектор tsvector, вес "char") returns tsvector
```

Данная функция возвращает копию входного вектора, помечая в ней каждую позицию заданным **весом**, меткой A, B, C или D.

Эти метки сохраняются при конкатенации векторов, что позволяет придавать разные веса словам из разных частей документа и, как следствие, ранжировать их по-разному.

Ранжирование результатов поиска

В PostgreSQL встроены две **функции ранжирования**:

```
ts_rank([веса float4[],] вектор tsvector, запрос tsquery) returns float4
```

Ранжирует векторы по частоте найденных лексем.

```
ts_rank_cd([веса float4[],] вектор tsvector, запрос tsquery) returns float4
```

Ранг вычисляется подобно ts_rank, но в расчёт берётся ещё и близость соответствующих лексем друг к другу.

В передаваемом массиве весов определяется, насколько весома каждая категория слов, в следующем порядке:

```
{вес D, вес C, вес B, вес A}
```

Если этот аргумент опускается, подразумеваются следующие значения:

```
{0.1, 0.2, 0.4, 1.0}
```

Пример

```
SELECT title, ts_rank_cd(textsearch, query) AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
WHERE query @@ textsearch
ORDER BY rank DESC
LIMIT 10;
                     title                     | rank
-----------------------------------------------+----------
Neutrinos in the Sun                           | 3.1
The Sudbury Neutrino Detector                  | 2.4
A MACHO View of Galactic Dark Matter           | 2.01317
Hot Gas and Dark Matter                        | 1.91171
The Virgo Cluster: Hot Plasma and Dark Matter  | 1.90953
Rafting for Solar Neutrinos                    | 1.9
NGC 4650A: Strange Galaxy and Dark Matter      | 1.85774
Hot Gas and Dark Matter                        | 1.6123
Ice Fishing for Cosmic Neutrinos               | 1.6
Weak Lensing Distorts the Universe             | 0.818218
```

Конфигурации

Позволяют гибко задавать функционал текстового поиска, строятся из четырёх объектов:

1. Анализаторы текстового поиска
2. Словари текстового поиска
3. Шаблоны текстового поиска
4. Конфигурации текстового поиска

Анализаторы и шаблоны текстового поиска строятся из низкоуровневых функций на **языке C**.

Индексы

Полнотекстовый поиск можно выполнить, не применяя индекс.

Однако при регулярном поиске для большинства приложений скорость будет неприемлемой. 😡

Поэтому для практического применения полнотекстового поиска обычно создаются **индексы**.

Типы индексов

Для ускорения полнотекстового поиска можно использовать индексы двух видов: **GIN** и **GiST**.

Более предпочтительными для текстового поиска являются индексы **GIN**. Они содержат записи для всех отдельных слов (лексем) с компактным списком мест их вхождений.

Индекс GiST допускает *неточности*, то есть он допускает ложные попадания и поэтому их нужно исключать дополнительно, сверяя результат с фактическими данными таблицы, что приводит к снижению производительности (PostgreSQL делает это *автоматически*)

Создание индексов

```
CREATE INDEX idx_name ON pgweb USING GIN (to_tsvector(regconfig, text));
```

Явное задание конфигурации необходимо.

Используется только для запросов с данной конфигурацией.

Можно определять конфигурацию в отдельном столбце таблицы:

```
CREATE INDEX pgweb_idx ON pgweb USING GIN (to_tsvector(config_name, body));
```

Можно создать индекс по объединению столбцов:

```
CREATE INDEX pgweb_idx ON pgweb USING GIN
                          (to_tsvector('english', title || ' ' || body));
```

Оптимизация индексов

Можно создать автообновляющийся столбец с результатами `to_tsvector`, затем построить по нему индекс:

```
ALTER TABLE pgweb
    ADD COLUMN ts_index_col tsvector GENERATED ALWAYS AS
        (to_tsvector('english',title || ' ' || body)) STORED;

CREATE INDEX ts_idx ON pgweb USING GIN (ts_index_col);
```

Ограничения

Текущая реализация текстового поиска в PostgreSQL имеет следующие ограничения:

- Длина лексемы не может превышать **2 килобайт**
- Длина значения tsvector (лексемы и их позиции) не может превышать **1 мегабайта**
- Число лексем должно быть **меньше 264**
- Значения позиций в tsvector должны быть **от 0 до 16383**
- Расстояние в операторе <*N*> не может быть больше **16384**
- Не больше **256** позиций для одной лексемы
- Число узлов (лексемы + операторы) в значении tsquery должно быть **меньше 32768**