



Возможности полнотекстового поиска в PostgreSQL

Ф.Г. Сигаев

А.А. Закиров

PgConf.Russia 2017, Москва

www.postgrespro.ru

Полнотекстовый поиск - это:

- поиск документов, удовлетворяющих некоторому запросу поиска
- сортировка результатов поиска по заданному критерию (дополнительно)

Типичный поиск — это:

- поиск документов, содержащих все слова запроса
- сортировка результатов поиска по релевантности

Почему не внешний поисковый движок

Внешние поисковые движки очень быстрые, но:

- не могут индексировать все документы, например, виртуальные
- не имеют доступа ко всем атрибутам, поэтому нельзя выполнить сложные запросы
- требуют поддержки администратором
- иногда должны быть сертифицированы
- не обеспечивают мгновенный поиск, т. к. нужно время для переиндексации новых данных
- не обеспечивают согласованность данных, например, данные могут быть удалены из базы данных

- Интеграция с движком базы данных:
 - транзакционность
 - конкурентный доступ
 - восстановление после сбоя
 - мгновенное индексирование
- Конфигурируемость (парсер, словарь)
- Масштабируемость

Полнотекстовый поиск в PostgreSQL

Операторы поиска по шаблону ~, ~*, LIKE, ILIKE плохо подходят:

- нет языковой поддержки
- нет сортировки по релевантности
- медленный поиск, т. к. необходимо последовательное сканирование документов

Но есть поддержка индексов с помощью модуля pg_trgm.

Полнотекстовый поиск в PostgreSQL

Типы для полнотекстового поиска:

- `tsvector` - сортированный массив лексем
- `tsquery` - выражение поиска документов

Операторы:

- `@@`: `tsvector @@ tsquery`
- `||`: `tsvector || tsvector` или `tsquery || tsquery`
- `&&`: `tsquery && tsquery`

Функции: `to_tsvector`, `to_tsquery`, `ts_lexize`, `ts_debug`, `ts_stat`, `ts_headline`, `ts_rank`, `ts_rank_cd`, `setweight`, ...

Индексы: GiST, GIN

Полнотекстовый поиск в PostgreSQL

Документ обрабатывается только один раз во время вставки, а не во время поиска:

- документ разбивается на токены с помощью подключаемого парсера
- токены преобразовываются в лексемы с помощью подключаемых словарей
- сохраняются координаты и важность слов и используются для сортировки по релевантности
- стоп-слова не учитываются в поиске

tsvector с позициями:

```
=# SELECT to_tsvector('The quick brown fox jumps over the  
lazy dog') AS tsvector;
```

```
-----  
'brown':3 'dog':9 'fox':4 'jump':5 'lazy':8 'quick':2
```

tsvector с позициями и метками:

```
=# SELECT setweight(to_tsvector('quick brown'), 'A') AS  
tsvector;
```

```
-----  
'brown':2A 'quick':1A
```

tsvector с позициями:

```
=# SELECT 'brown:3 dog:9 fox:4 jump:5 lazy:8  
quick:2'::tsvector;
```

tsvector

```
'brown':3 'dog':9 'fox':4 'jump':5 'lazy':8 'quick':2
```

tsvector с позициями и метками:

```
=# SELECT 'brown:2A quick:1A'::tsvector;  
tsvector
```

```
'brown':2A 'quick':1A
```

tsquery поддерживает операторы &, |, ! и <->:

```
=# SELECT to_tsquery('quick & (fox | dog)') AS tsquery;  
      tsquery
```

```
-----  
'quick' & ( 'fox' | 'dog' )
```

tsquery с метками:

```
=# SELECT to_tsquery('quick:AB & dog') AS tsquery;  
      tsquery
```

```
-----  
'quick':AB & 'dog'
```

tsquery для префиксного поиска:

```
=# SELECT to_tsquery('quick & eleph:*') AS tsquery;  
      tsquery
```

```
-----  
'quick' & 'eleph':*
```

tsquery с различными операторами:

```
=# SELECT 'quick & ( fox | dog )'::tsquery;  
      tsquery
```

```
-----  
'quick' & ( 'fox' | 'dog' )
```

tsquery с метками:

```
=# SELECT 'quick:AB & dog'::tsquery;  
      tsquery
```

```
-----  
'quick':AB & 'dog'
```

tsquery для префиксного поиска:

```
=# SELECT 'quick & eleph:*'::tsquery;  
      tsquery
```

```
-----  
'quick' & 'eleph':*
```

Типы токенов:

```
=# SELECT alias, description FROM ts_token_type('default');
```

alias	description
asciword	Word, all ASCII
word	Word, all letters
numword	Word, letters and digits
email	Email address
url	URL
host	Host
sfloat	Scientific notation
version	Version number
hword_numpart	Hyphenated word part, letters and digits
hword_part	Hyphenated word part, all letters
hword_asciipart	Hyphenated word part, all ASCII
blank	Space symbols
tag	XML tag
protocol	Protocol head
numhword	Hyphenated word, letters and digits
asciuhword	Hyphenated word, all ASCII
hword	Hyphenated word, all letters
url_path	URL path
file	File or path name
float	Decimal notation
int	Signed integer
uint	Unsigned integer
entity	XML entity

Типы токенов:

=# **SELECT** alias, token, dictionary, lexemes

FROM ts_debug('The quick brown fox jumps over the lazy dog');

alias	token	dictionary	lexemes
asciword	The	english_stem	{}
blank		(null)	(null)
asciword	quick	english_stem	{ quick }
blank		(null)	(null)
asciword	brown	english_stem	{ brown }
blank		(null)	(null)
asciword	fox	english_stem	{ fox }
blank		(null)	(null)
asciword	jumps	english_stem	{ jump }
blank		(null)	(null)
asciword	over	english_stem	{}
blank		(null)	(null)
asciword	the	english_stem	{}
blank		(null)	(null)
asciword	lazy	english_stem	{ lazi }
blank		(null)	(null)
asciword	dog	english_stem	{ dog }

Типы словарей:

- simple

```
=# SELECT to_tsvector('simple', 'lazy');  
to_tsvector  
-----  
'lazy':1
```

- synonym

```
=# SELECT to_tsvector('synonym_sample', 'lazy');  
to_tsvector  
-----  
'indolent':1
```

- thesaurus

```
=# SELECT to_tsvector('thesaurus_sample', 'brown fox');  
to_tsvector  
-----  
'fox':1
```

Типы словарей:

- snowball

```
=# SELECT to_tsvector('english', 'lazy abilities');  
to_tsvector  
-----  
'abil':2 'lazi':1
```

- ispell

```
=# SELECT to_tsvector('english_hunspell', 'lazy');  
to_tsvector  
-----  
'lazy':1
```

```
=# SELECT to_tsvector('english_hunspell', 'abilities');  
to_tsvector  
-----  
'ability':1
```

```
=# SELECT to_tsvector('english_hunspell', 'inability');  
to_tsvector  
-----  
'ability':1
```

github.com/postgrespro/hunspell_dicts
github.com/postgrespro/shared_ispell

Конфигурации полнотекстового поиска

Создание конфигурации:

```

=# CREATE TEXT SEARCH CONFIGURATION ru_conf (parser='default');
=# ALTER TEXT SEARCH CONFIGURATION ...
=# \dF+ ru_conf
      Text search configuration "public.ru_conf"
Parser: "pg_catalog.default"

```

Token	Dictionaries
asciihword	english_hunspell,english_stem
asciiword	english_hunspell,english_stem
email	simple
file	simple
float	simple
host	simple
hword	russian_hunspell,russian_stem
hword_asciipart	english_hunspell,english_stem
hword_numpart	simple
hword_part	russian_hunspell,russian_stem
int	simple
numhword	simple
numword	simple
sfloat	simple
uint	simple
url	simple
url_path	simple
version	simple
word	russian_hunspell,russian_stem

Конфигурации полнотекстового поиска

Конфигурация для поиска только по словам:

```
=# \dF+ ru_conf_word
```

```
Text search configuration "public.ru_conf_word"
```

```
Parser: "pg_catalog.default"
```

Token	Dictionaries
asciihword	english_hunspell,english_stem
asciword	english_hunspell,english_stem
hword	russian_hunspell,russian_stem
hword_asciipart	english_hunspell,english_stem
hword_part	russian_hunspell,russian_stem
word	russian_hunspell,russian_stem

```
=# SELECT to_tsvector('ru_conf_word', '100500 elephants on  
postgresql.org');
```

```
to_tsvector
```

```
-----  
'elephant':1
```

Функции ранжирования:

- `ts_rank(weights, vector, query, normalization)`
- `ts_rank_cd(weights, vector, query, normalization)`

формат `weights`:

`{D-weight, C-weight, B-weight, A-weight}`

`weights` по-умолчанию:

`{0.1, 0.2, 0.4, 1.0}`

Далекие лексемы:

```
=# SELECT ts_rank_cd(to_tsvector('The quick brown fox jumps  
over the lazy dog'), to_tsquery('brown & dog'));
```

```
ts_rank_cd
```

```
-----
```

```
0.0166667
```

Ближкие лексемы:

```
=# SELECT ts_rank_cd(to_tsvector('The quick brown fox jumps  
over the lazy dog'), to_tsquery('brown & fox'));
```

```
ts_rank_cd
```

```
-----
```

```
0.1
```

```
=# SELECT ts_rank_cd(setweight(to_tsvector('The quick brown  
fox jumps'), 'A') || to_tsvector('over the lazy dog'),  
to_tsquery('brown & fox'));
```

```
ts_rank_cd
```

```
-----
```

```
1
```

Ранжирование с помощью меток:

```
=# SELECT ts_rank_cd(setweight(to_tsvector('The quick brown  
fox jumps'), 'A') || to_tsvector('over the lazy dog'),  
to_tsquery('brown & fox'));  
ts_rank_cd  
-----  
1
```

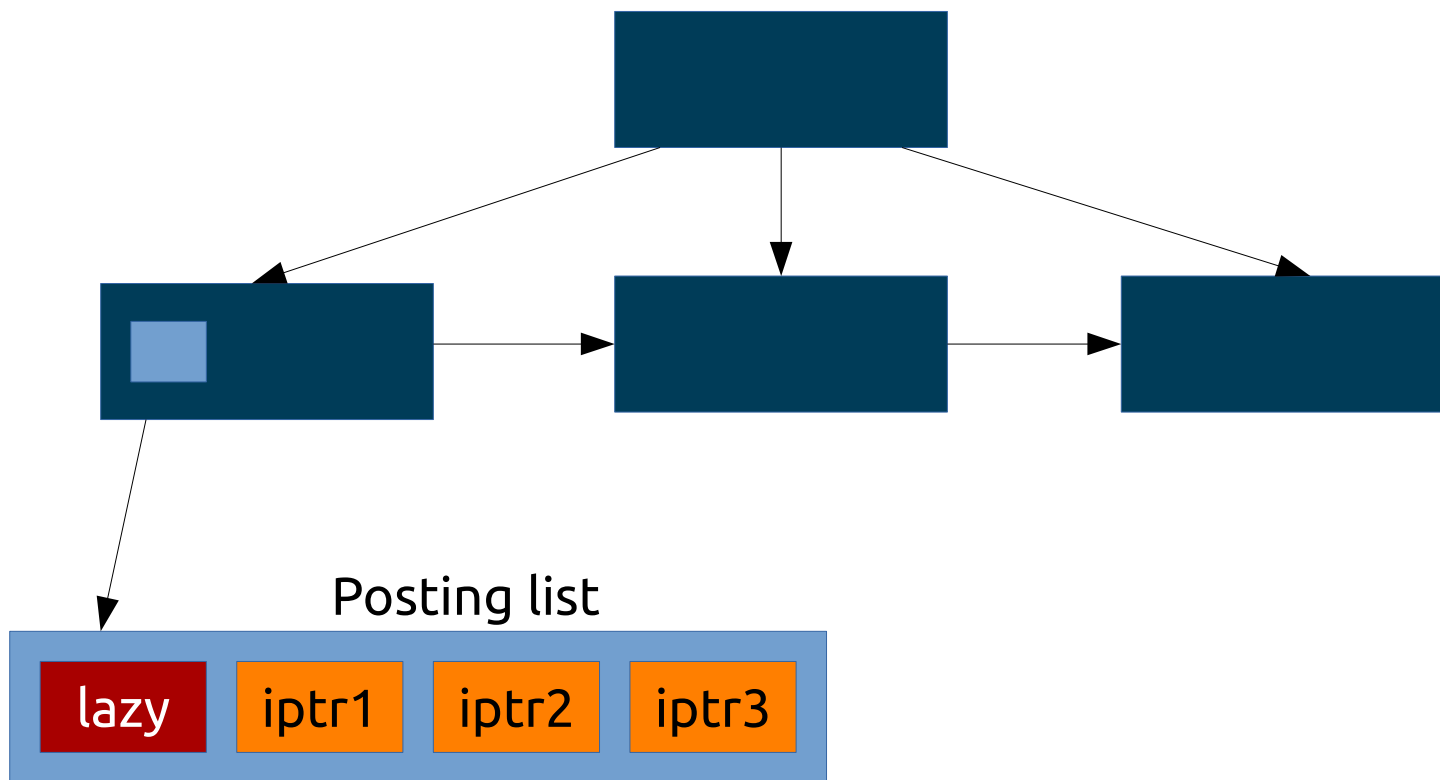
Веса для меток:

```
=# SELECT ts_rank_cd(ARRAY[0.1, 0.1, 0.1, 0.1],  
setweight(to_tsvector('The quick brown fox jumps'), 'A') ||  
to_tsvector('over the lazy dog'), to_tsquery('brown &  
fox'));  
ts_rank_cd  
-----  
0.1
```

Индексы:

- GiST
 - быстрая вставка данных
 - более медленный поиск, чем в GIN
- GIN
 - более медленная вставка данных
 - но полнотекстовый поиск быстрее
- RUM
 - вставка данных медленнее, чем в GIN
 - размер индекса больше
 - но полнотекстовый поиск и ранжирование быстрее

Индексы. GIN



1 - Медленное ранжирование

```
=# SELECT id, ts_rank(fts, to_tsquery('english', 'Text &
search')) AS rank
FROM pglist
WHERE fts @@ to_tsquery('english', 'Text & search')
ORDER BY rank DESC
LIMIT 10;
```

```
Limit (actual time=157.811..157.814 rows=10 loops=1)
-> Sort (actual time=157.808..157.809 rows=10 loops=1)
    Sort Key: (ts_rank(fts, '''text' & 'search' '::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 25kB
-> Bitmap Heap Scan on pglist (actual time=26.211..152.331
rows=9924 loops=1)
    Recheck Cond: (fts @@ '''text' & 'search' '::tsquery)
    Heap Blocks: exact=8125
-> Bitmap Index Scan on pglist_idx (actual
time=24.119..24.119 rows=9924 loops=1)
    Index Cond: (fts @@ '''text' & 'search' '::tsquery)
Planning time: 0.703 ms
Execution time: 157.948 ms
```

2 - Медленный фразовый поиск

```
=# SELECT id, ts_rank(fts, to_tsquery('english', 'Text <->
search')) AS rank
FROM pglist
WHERE fts @@ to_tsquery('english', 'Text <-> search')
ORDER BY rank DESC
LIMIT 10;
```

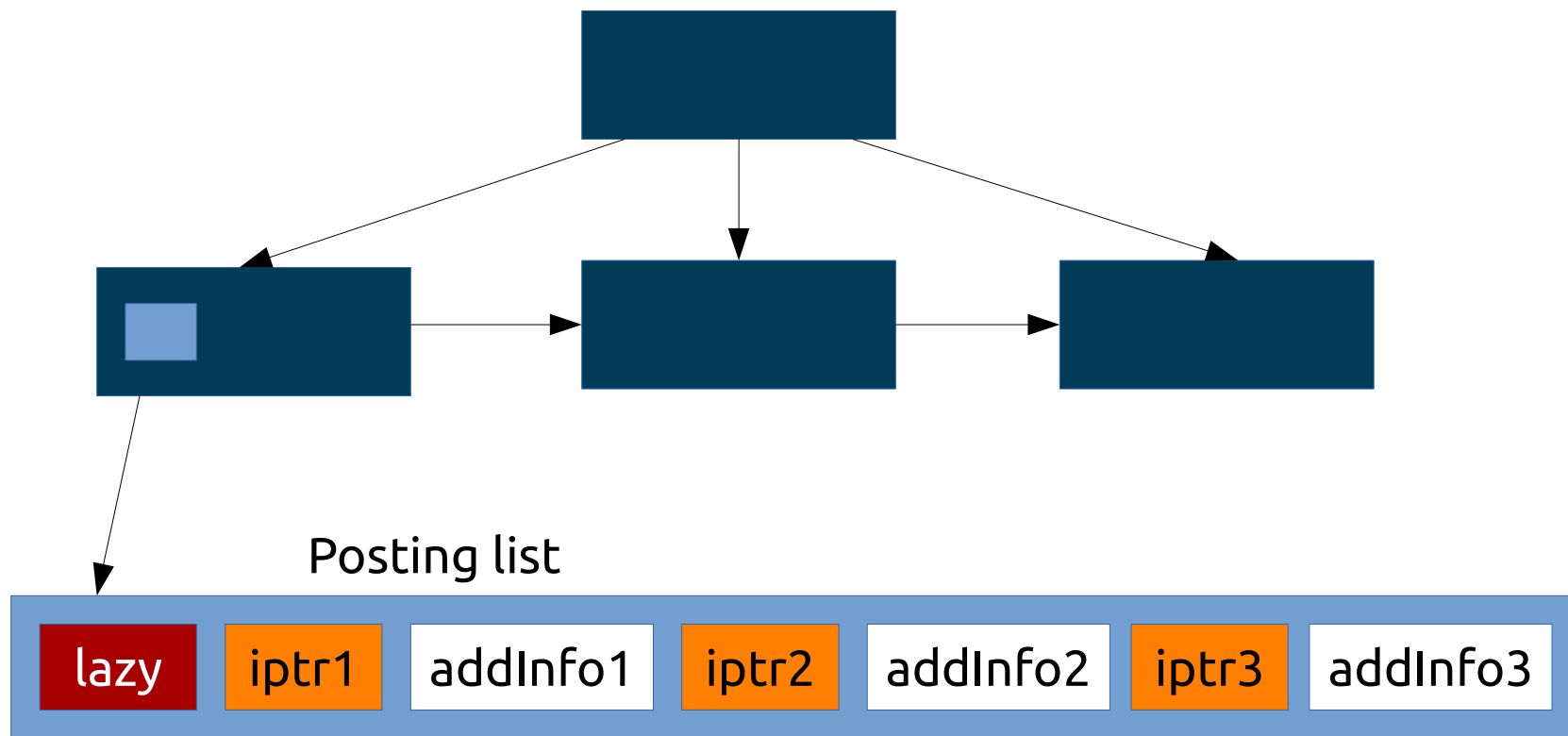
```
Limit (actual time=170.572..170.575 rows=10 loops=1)
-> Sort (actual time=170.570..170.572 rows=10 loops=1)
    Sort Key: (ts_rank(fts, '''text' <-> 'search' '::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 25kB
-> Bitmap Heap Scan on pglist (actual time=24.027..168.358
rows=3803 loops=1)
    Recheck Cond: (fts @@ '''text' <-> 'search' '::tsquery)
    Rows Removed by Index Recheck: 6121
    Heap Blocks: exact=8125
-> Bitmap Index Scan on pglist_idx (actual
time=21.755..21.755 rows=9924 loops=1)
    Index Cond: (fts @@ '''text' <-> 'search' '::tsquery)
Planning time: 0.470 ms
Execution time: 170.683 ms
```

3 - Медленная сортировка по другим полям (например, date)

```
=# SELECT id, sent  
FROM pglist  
WHERE fts @@ to_tsquery('english', 'index & scan')  
ORDER BY abs(extract(epoch from (sent - now())))  
LIMIT 10;
```

```
Limit (actual time=150.496..150.499 rows=10 loops=1)  
  -> Sort (actual time=150.493..150.495 rows=10 loops=1)  
        Sort Key: (abs(date_part('epoch'::text, ((sent)::timestamp with  
time zone - now()))))  
        Sort Method: top-N heapsort  Memory: 25kB  
        -> Bitmap Heap Scan on pglist (actual time=42.855..134.649  
rows=34068 loops=1)  
              Recheck Cond: (fts @@ '''index' & 'scan'::tsquery)  
              Heap Blocks: exact=25206  
              -> Bitmap Index Scan on pglist_idx (actual  
time=34.810..34.810 rows=34068 loops=1)  
                    Index Cond: (fts @@ '''index' & 'scan'::tsquery)  
Planning time: 0.741 ms  
Execution time: 150.747 ms
```

Индексы. RUM



- PostgreSQL 9.6+
- github.com/postgrespro/rum
- Использует Generic WAL Records
(www.postgresql.org/docs/9.6/static/generic-wal.html)

1 - Ранжирование

```
=# SELECT id, fts <=> to_tsquery('english', 'text &
search') AS rank
FROM pglist
WHERE fts @@ to_tsquery('english', 'text & search')
ORDER BY rank
LIMIT 10;
```

```
Limit (actual time=62.845..62.959 rows=10 loops=1)
  -> Index Scan using pglist_rum_idx on pglist (actual
time=62.843..62.955 rows=10 loops=1)
    Index Cond: (fts @@ '''text'&'search'''::tsquery)
    Order By: (fts <=> '''text'&'search'''::tsquery)
Planning time: 0.739 ms
Execution time: 63.209 ms
```

2 - Фразовый поиск

```
=# SELECT id, fts <=> to_tsquery('english', 'text <->
search') AS rank
FROM pglist
WHERE fts @@ to_tsquery('english', 'text <-> search')
ORDER BY rank
LIMIT 10;
```

```
Limit (actual time=52.727..52.851 rows=10 loops=1)
-> Index Scan using pglist_rum_idx on pglist (actual
time=52.724..52.846 rows=10 loops=1)
    Index Cond: (fts @@ '''text'<->'search''::tsquery)
    Order By: (fts <=> '''text'<->'search''::tsquery)
Planning time: 0.787 ms
Execution time: 53.540 ms
```

3 - Сортировка по другим полям (например, date)

```
=# SELECT id, sent  
FROM pglist  
WHERE fts @@ to_tsquery('english', 'text & search')  
ORDER BY sent <=> '2000-01-01'::timestamp  
LIMIT 10;
```

Limit (actual time=39.907..39.926 rows=10 loops=1)

-> Index Scan using pglist_date_idx on pglist (actual time=39.905..39.924 rows=10 loops=1)

Index Cond: (fts @@ '''text' & 'search''::tsquery)

Order By: (sent <=> '2000-01-01 00:00:00'::timestamp
without time zone)

Planning time: 0.720 ms

Execution time: **40.273 ms**

- Более медленное построение индекса
- Вставка данных медленнее
- Размер индекса больше
- Большой WAL трафик при вставке данных, но не при построении индекса. Это происходит из-за использования Generic WAL Records

Сравнение GIN и RUM

Создание индекса для таблицы размером 1290 Мб.

	GIN	RUM	RUM+ts	RUM+ts+order
Время создания, сек	169	189	196	255
Размер, Мб	535	952	1489	1869
WAL, Мб	927	681	1125	1476

- Несколько дополнительных информации (позиции лексем + timestamp)
- Классы операторов для массивов, integer и float
- Улучшение функции ранжирования для поддержки TF/IDF
- Улучшение времени вставки в индекс
- Улучшение производительности Generic WAL Records

github.com/postgrespro/rum

Пример сайта с полнотекстовым поиском

- Поиск с помощью RUM
- Ранжирование с помощью разных функций:
ts_rank(), ts_rank_cd(), ts_score()
- Фасетный поиск
- Отображение сгенерированного запроса

tsdemo.postgrespro.ru

github.com/postgrespro/apod_fts

Пример сайта с полнотекстовым ПОИСКОМ

apod

msg_id
title
text
lang
sections
keywords
date

keywords

key_id
name
status_id

sections

sect_id
name
status_id

Пример сайта с полнотекстовым поиском. Словари

Ispell словари:

- нормализация нескольких форм слова в одну лексему
- поддержка форматов Ispell, MySpell, Hunspell
- исключение стоп-слов
- настройка с помощью файлов .affix и .dict

Пример сайта с полнотекстовым поиском. Словари

Установка словарей в PostgreSQL:

- Загрузить LibreOffice расширения:
<http://extensions.libreoffice.org/extension-center?getCategories=Dictionary>
- Извлечь файлы .dic и .aff из .oxt архива
- Переименовать файлы в .dict и .affix
- Конвертация файлов в UTF-8 кодировку
- Скопировать файлы в папку
\$SHAREDIR/tsearch_data/
- Загрузка файлов в PostgreSQL с помощью команды

```
CREATE TEXT SEARCH DICTIONARY en_hunspell (  
    TEMPLATE = ispell,  
    DictFile = en_us,  
    AffFile = en_us);
```

Советы и приемы

- `setweight(tsvector, char, text[])` - добавить метки лексемам из `text[]`

```
=# SELECT setweight(to_tsvector('english', '22-nd
anniversary of PostgreSQL'), 'A', '{postgresql,22}');
      setweight
```

```
-----
'22':1A 'anniversari':3 'nd':2 'postgresql':5A
```

- `ts_delete(tsvector, text[])` - удалить лексемы из `tsvector`

```
=# SELECT ts_delete(to_tsvector('english', '22-nd
anniversary of PostgreSQL'), '{22,postgresql}'::text[]);
      ts_delete
```

```
-----
'anniversari':3 'nd':2
```

- `unnest(tsvector)` - преобразовать `tsvector` в таблицу
`=# SELECT * FROM unnest(setweight(to_tsvector('english',
'22-nd anniversary of PostgreSQL'), 'A',
'{postgresql,22}'));`

lexeme	positions	weights
22	{1}	{A}
anniversari	{3}	{D}
nd	{2}	{D}
postgresql	{5}	{A}

- `tsvector_to_array(tsvector)`, `array_to_tsvector(text[])`

```
=# SELECT tsvector_to_array(to_tsvector('english', '22-  

nd anniversary of PostgreSQL'));
```

```
tsvector_to_array
```

```
{22,anniversari,nd,postgresql}
```

Спасибо за внимание!