Энциклопедия полнотекстового поиска

Oleg Bartunov
Postgres Professional, Moscow University
obartunov@postgrespro.ru

Moscow, PGConf.ru, Feb 5, 2020

Postgres developer/contributor since 1995

PROFESSIONAL
Postgres

Agenda

- What is a Text Search in PostgreSQL ?
- Why FTS in PostgreSQL ?
- Basics of FTS
- Mastering FTS
  - Parsers, dictionaries, configurations
  - SQL interface
  - FTS Functions
- Indexing
- FTS future

What is a Full Text Search ?

- Full text search
  - Find documents, which match a query
  - Sort them in some order (optionally)
- Typical Search
  - Find documents with **all words** from the query
  - Return them sorted by relevance

What is a document ?

- Arbitrary text attribute
- Combination of text attributes from the same or different tables (result of join[s])

```
msg (id, lid, subject, body);
lists (lid, list);

SELECT l.list || m.subject || m.body_plain as doc


Don’t forget about COALESCE (doc,’’)
```

What is a query ?

- Arbitrary text

  ‘open source’
- Text with some query language

```
'postgresql "open source * database" -die +most'
```

Why FTS in PostgreSQL ?

- Feed database content to external search engines
  - They are fast !

  BUT
- They can't index all documents - could be totally virtual
- They don't have access to attributes - no complex queries
- They have to be maintained — headache for DBA
- Sometimes they need to be certified
- They don't provide instant search (need time to download new data and reindex)
- They don't provide consistency — search results can be already deleted from database

Your system may looks like this

FTS in PostgreSQL

- **FTS requirements**
  - **Full integration with database engine**
    - Transactions
    - Concurrent access
    - Recovery
    - Online index
  - Configurability (parser, dictionary...)
  - Scalability

Text Search Operators

- Traditional text search operators
  (TEXT op TEXT, op - ~, ~*, LIKE, ILIKE)

```
=# select title  from apod where title ~* 'x-ray' limit 5;
                  title
-----------------------------------------
 The X-Ray Moon
 Vela Supernova Remnant in X-ray
 Tycho's Supernova Remnant in X-ray
 ASCA X-Ray Observatory
 Unexpected X-rays from Comet Hyakutake
(5 rows)

=# select title  from apod where title ilike '%x-ray%' limit 5;
                 title
-----------------------------------------
 The Crab Nebula in X-Rays
 X-Ray Jet From Centaurus A
 The X-Ray Moon
 Vela Supernova Remnant in X-ray
 Tycho's Supernova Remnant in X-ray
(5 rows)
```

Text Search Operators

- Traditional text search operators
  (TEXT op TEXT, op - ~, ~*, LIKE, ILIKE)
  - No linguistic support
    - What is a word ?
    - What to index ?
    - Word «normalization» ?
    - Stop-words (noise-words)
  - No ranking - all documents are equally similar to query
  - Slow, documents should be seq. scanned
    9.3+ index support of ~* (pg_trgm)

select * from man lines where man line ~* '(?:(?:p(?:ostgres(?:ql)?|g?sql)|sql)) (?:(?:(?:mak|us)e|do|is))';

```
One of (postgresql,sql,postgres,pgsql,psql) space One of (do,is,use,make)
```

FTS in PostgreSQL

- OpenFTS — 2000, Pg as a storage
- GiST index — 2000, thanks Rambler
- Tsearch — 2001, contrib:no ranking
- Tsearch2 — 2003, contrib:config
- GIN —2006, thanks, JFG Networks
- FTS — 2006, in-core, thanks,EnterpriseDB
- RUM –- 2016, extension, Postgres Pro

Team:
Teodor Sigaev, Oleg Bartunov, Alexander Korotkov, Arthur Zakirov

Basics of FTS

- **tsvector** – data type for document optimized for search
  - Sorted array of lexems
  - Positional information
  - Structural information (importance)
- **tsquery** – textual data type for query with boolean operators & | ! ()

- **Full text search operator:** tsvector @@ tsquery

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

FTS configuration

1)Parser breaks text on to (token, type) pairs

2)Tokens converted to the lexems using dictionaries specific for token type

- Extendability:
  - Pluggable parser and dictionaries
  - FTS configuration defines parser and dictionaries
  - FTS configurations used for document and query processing
- \dF{,p,d}[+] [pattern] — psql FTS
- SQL interface:

```
{CREATE | ALTER | DROP} TEXT SEARCH {CONFIGURATION | DICTIONARY | PARSER}
```

FTS in PostgreSQL

- Document to tsvector:
  - to_tsvector([cfg], text|json|jsonb)
    cfg — FTS configuration,
    GUC default_text_search_config

```
select to_tsvector('It is a very long story about true and false');
                to_tsvector
-------------------------------------
 'fals':10 'long':5 'stori':6 'true':8
(1 row)

select to_tsvector('simple', 'It is a very long story about true and false');
                                    to_tsvector
-------------------------------------------------------------------------------
 'a':3 'about':7 'and':9 'false':10 'is':2 'it':1 'long':5 'story':6 'true':8 'very':4
(1 row)
```

FTS in PostgreSQL

- JSON[b] to tsvector:
  - Notice, results are different for json and jsonb !
    Jsonb: keys are sorted, Json: spaces are preserved
  - Phrases are preserved

```
select to_tsvector(jb) from (values ('
{
    "abstract": "It is a very long story about true and false",
    "title": "Peace and War",
    "publisher": "Moscow International house"
 }
'::json[b])) foo(jb) as tsvector_json[b]
                                    tsvector_json
-------------------------------------------------------------------------------------
 'fals':10 'hous':18 'intern':17 'long':5 'moscow':16 'peac':12 'stori':6 'true':8 'war':14
(1 row)

                                    tsvector_jsonb
-------------------------------------------------------------------------------------
 'fals':14 'hous':18 'intern':17 'long':9 'moscow':16 'peac':1 'stori':10 'true':12 'war':3
(1 row)
```

Tsvector editing functions

- Different parts of document can be marked to use for ranking at search time.
  setweight(tsvector, «char», text[] - add label to lexemes from text[]

```
select setweight( to_tsvector('english', '20-th anniversary of PostgreSQL'),
'A',  '{postgresql,20}');
                   setweight
-----------------------------------------------
 '20':1A 'anniversari':3 'postgresql':5A 'th':2
(1 row)
```

- ts_delete(tsvector, text[]) - delete lexemes from tsvector

```
select ts_delete( to_tsvector('english', '20-th anniversary of PostgreSQL'),
'{20,postgresql}'::text[]);
       ts_delete
------------------------
 'anniversari':3 'th':2
(1 row)
```

Tsvector editing functions

- unnest(tsvector)

```
select * from unnest( setweight( to_tsvector('english',
'20-th anniversary of PostgreSQL'),'A',  '{postgresql,20}'));
   lexeme    | positions | weights
-------------+-----------+---------
 20          | {1}       | {A}
 anniversari | {3}       | {D}
 postgresql  | {5}       | {A}
 th          | {2}       | {D}
(4 rows)
```

- tsvector_to_array(tsvector) — tsvector to text[]
  array_to_tsvector(text[])

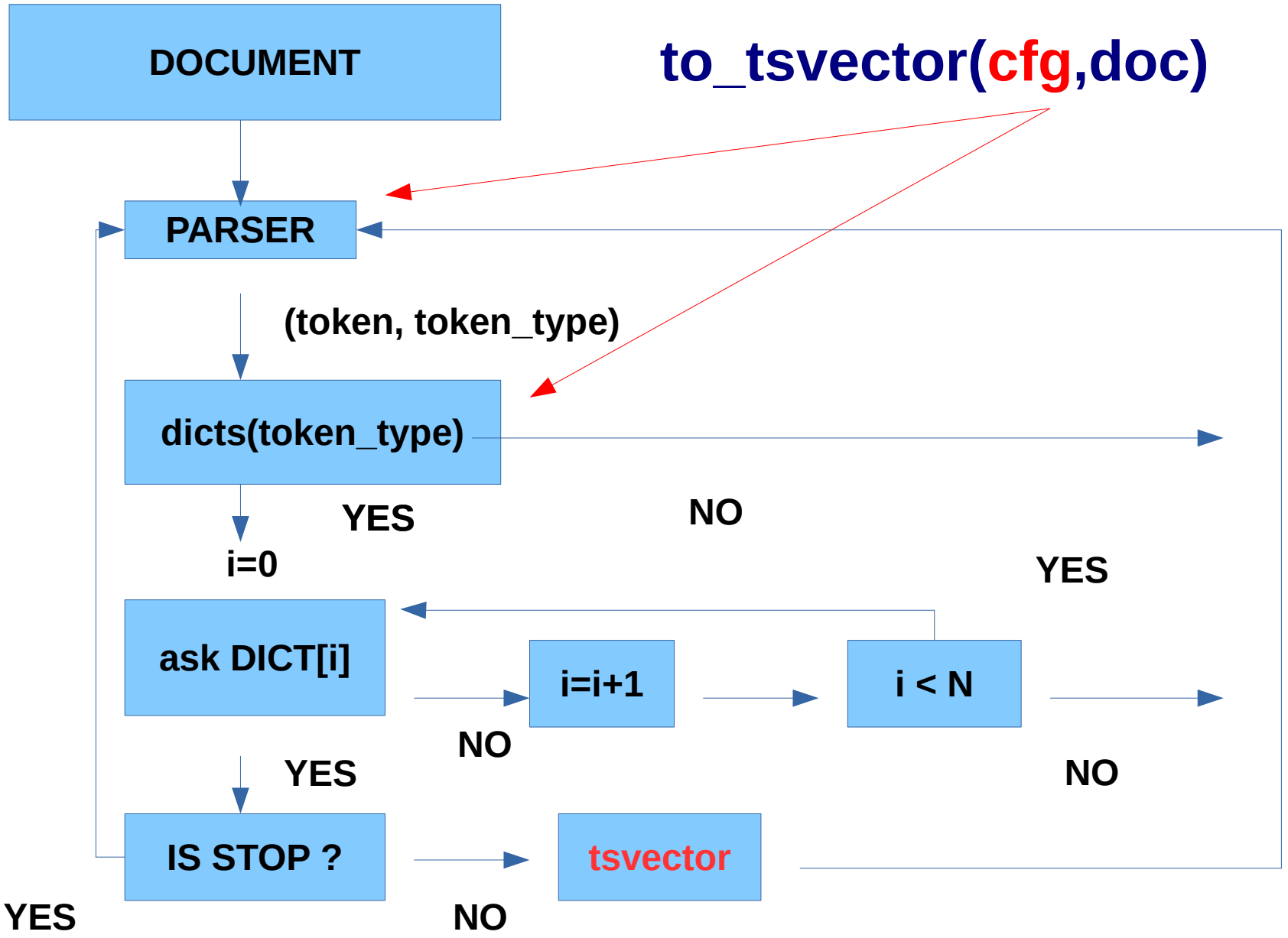
```
select tsvector_to_array( to_tsvector('english',
'20-th anniversary of PostgreSQL'));
       tsvector_to_array
-------------------------------
 {20,anniversari,postgresql,th}
(1 row)
```

Tsvector editing functions

- ts_filter(tsvector,text[]) - fetch lexemes with specific label{s}

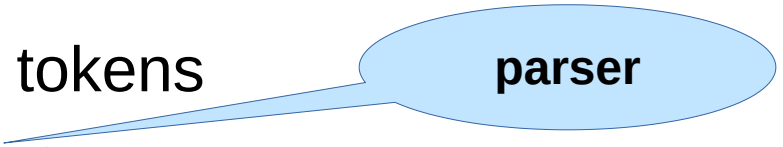
```
select ts_filter($$'20':2A 'anniversari':4C 'postgresql':1A,6A 'th':3$$::tsvector,
'{C}');
    ts_filter
------------------
 'anniversari':4C
(1 row)

select ts_filter($$'20':2A 'anniversari':4C 'postgresql':1A,6A 'th':3$$::tsvector,
'{C,A}');
                   ts_filter
-----------------------------------------------
 '20':2A 'anniversari':4C 'postgresql':1A,6A
(1 row)
```

FTS PostgreSQL



FTS in PostgreSQL

- Parser breaks document into tokens




```
=# select * from ts_token_type('default');
 tokid |      alias      |                description
-------+-----------------+-------------------------------------------
     1 | asciiword       | Word, all ASCII
     2 | word            | Word, all letters
     3 | numword         | Word, letters and digits
     4 | email           | Email address
     5 | url             | URL
     6 | host            | Host
     7 | sfloat          | Scientific notation
     8 | version         | Version number
     9 | hword_numpart   | Hyphenated word part, letters and digits
    10 | hword_part      | Hyphenated word part, all letters
    11 | hword_asciipart | Hyphenated word part, all ASCII
    12 | blank           | Space symbols
    13 | tag             | XML tag
    14 | protocol        | Protocol head
    15 | numhword        | Hyphenated word, letters and digits
    16 | asciihword      | Hyphenated word, all ASCII
    17 | hword           | Hyphenated word, all letters
    18 | url_path        | URL path
    19 | file            | File or path name
    20 | float           | Decimal notation
    21 | int             | Signed integer
    22 | uint            | Unsigned integer
    23 | entity          | XML entity
(23 rows)
```

Dictionaries

- **Dictionary** – is a **program**, which accepts token on input and returns an array of lexems, NULL if token doesn‘t recognized and empty array for stop-word.
- ts_lexize(dictionary)

```
SELECT ts_lexize('english_hunspell','a') as stop,
ts_lexize('english_hunspell','elephants') AS elephants,
ts_lexize('english_hunspell','elephantus') AS unknown;
 stop |  elephants | unknown
------+------------+---------
 {}   | {elephant} | (null)
(1 row)
```

- Dictionary API allows to develop any custom dictionaries
  - Truncate too long numbers
  - Convert colors
  - Convert URLs to canonical way

    http://a.in/a/./index.html → http://a.in/a/index.html

Dictionaries

- Dictionary — is a program !

```
=# select ts_lexize('intdict', 11234567890);
  ts_lexize
-----------
 {112345}
=# select ts_lexize('roman', 'XIX');
 ts_lexize
------------
 {19}

=# select ts_lexize('colours','#FFFFFF');
ts_lexize
------------
{white}
```

Astronomical dictionary

Dictionary with regexp support (pcre library)

```
# Messier objects
(M|Messier)(\s|-)?((\d){1,3}) M$3
# catalogs
(NGC|Abell|MKN|IC|H[DHR]|UGC|SAO|MWC)(\s|-)?((\d){1,6}[ABC]?) $1$3
(PSR|PKS)(\s|-)?([JB]?)(\d\d\d\d)\s?([+-]\d\d)\d? $1$4$5
# Surveys
OGLE(\s|-)?((I){1,3}) ogle
2MASS twomass
# Spectral lines
H(\s|-)?(alpha|beta|gamma) h$2
(Fe|Mg|Si|He|Ni)(\s|-)?((\d)|([IXV])+) $1$3
# GRBs
gamma\s?ray\s?burst(s?) GRB
GRB\s?(\d\d\d\d\d\d)([abcd]?) GRB$1$2
```

```
SELECT ts_lexize('regex', 'ngc 1234');
 ts_lexize
-----------
 {ngc1234}
(1 row)
```

Built-in Dictionaries

Dictionary templates:

1. Simple
   - convert the input token to lower case
   - exclude stop words
2. Synonym (also, contrib/xsyn)
   - replace word with a synonym

   Example of .syn file:

```
postgres    pgsql
postgresql pgsql
postgre     pgsql
```

Built-in Dictionaries

3. Thesaurus
   - replace phrase by indexed phrase

   Example of .ths file:

```
booking tickets : order invitation cards
booking ? tickets : order invitation Cards
```

4. Snowball stemmer
   - reduce words by stemming algorithms
   - recognizes everything
   - exclude stop words

```
SELECT ts_lexize('portuguese_stem','responsáveis');
 ts_lexize
-----------
 {respons}
(1 row)
```

Built-in Dictionaries

- Portuguese snowball stemmer dictionary

```
 viva  | vivo  | viver
-------+-------+-------
 {viv} | {viv} | {viv}
```

```
select ts_lexize('portuguese_stem','responsáveis');
 ts_lexize
-----------
 {respons}
(1 row)
```

- Available as a part of PostgreSQL core

Built-in Dictionaries

5. Ispell
   - normalize different linguistic forms of a word into the same lexeme. Try to reduce an input word to its infinitive form
   - support dictionary file formats: Ispell, MySpell, Hunspell
   - exclude stop words

```
      viva          |      vivo       |   viver
--------------------+-----------------+----------
 {viva,vivo,viver} | {vivo,viver} | {viver}
```

Filter dictionary – unaccent

contrib/unaccent - unaccent text search dictionary and function to remove accents (suffix tree, ~ 25x faster *translate*() solution)

```
1. Unaccent dictionary does nothing and returns NULL.
(lexeme 'Hotels' will be passed to the next dictionary if any)

=# select ts_lexize('unaccent','Hotels') is NULL;
 ?column?
----------
 t

2. Unaccent dictionary removes accent and returns 'Hotel'.
(lexeme 'Hotel' will be passed to the next dictionary if any)

=# select ts_lexize('unaccent','Hôtel');
 ts_lexize
-----------
 {Hotel}
```

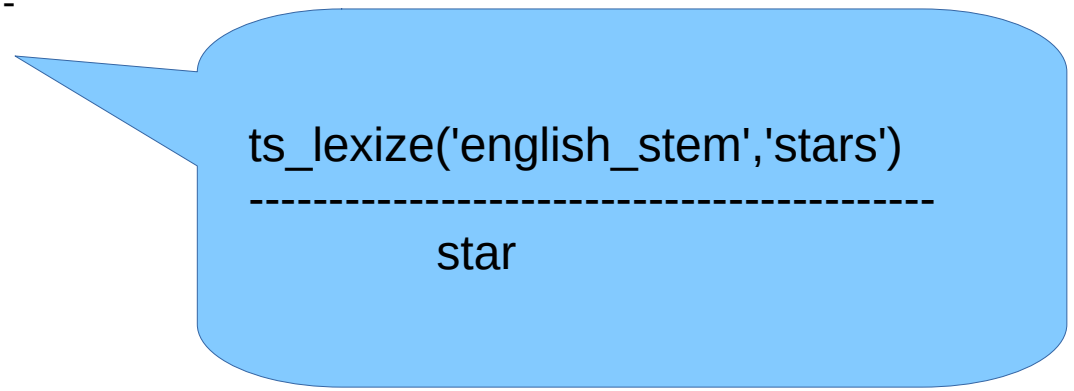
Filter dictionary - unaccent

```
CREATE TEXT SEARCH CONFIGURATION fr ( COPY = french );
ALTER TEXT SEARCH CONFIGURATION fr ALTER MAPPING FOR hword, hword_part, word
    WITH unaccent, french_stem;


=# select to_tsvector('fr','Hôtel de la Mer') @@ to_tsquery('fr','Hotels');
 ?column?
----------
 t



=# select ts_headline('fr','Hôtel de la Mer',to_tsquery('fr','Hotels'));
      ts_headline
------------------------
  <b>Hôtel</b> de la Mer
```

FTS in PostgreSQL

- Each token processed by a set of dictionaries

```
=# \dF+ russian
Text search configuration "pg_catalog.russian"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | russian_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | russian_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | russian_stem
```

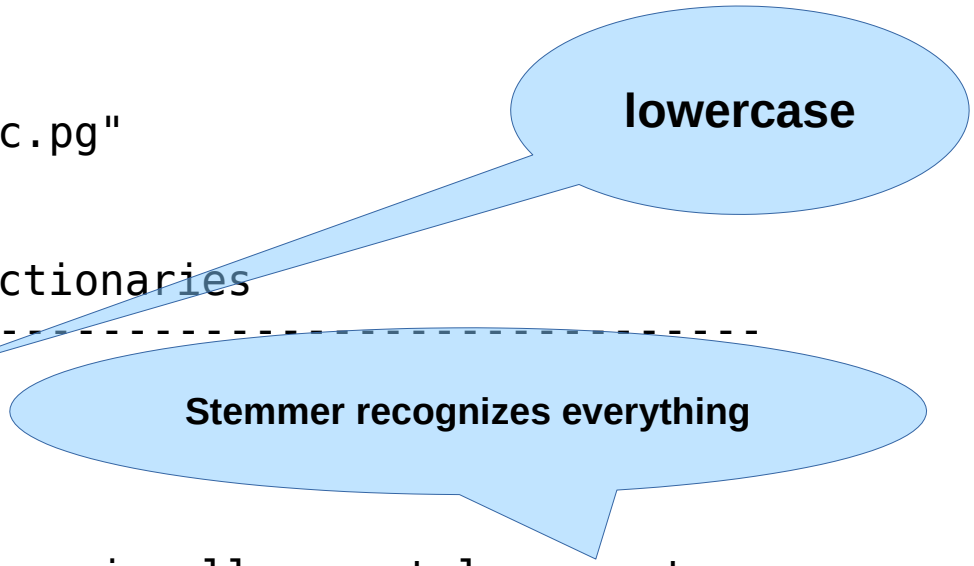


FTS in PostgreSQL

- Token processed by dictionaries until it recognized
- It is discarded, if it's not recognized

Rule: from «specific» dictionary to a «common» dictionary

```
=# \dF+ pg
                 Configuration "public.pg"
Parser name: "pg_catalog.default"
Locale: 'ru_RU.UTF-8' (default)
    Token     |                       Dictionaries
--------------+-------------------------------------------------------
 file         | pg_catalog.simple
 host         | pg_catalog.simple
 hword        | pg_catalog.simple
 int          | pg_catalog.simple
 lhword       | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 lpart_hword  | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 Lword        | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 nlhword      | pg_catalog.simple
 nlpart_hword | pg_catalog.simple
```



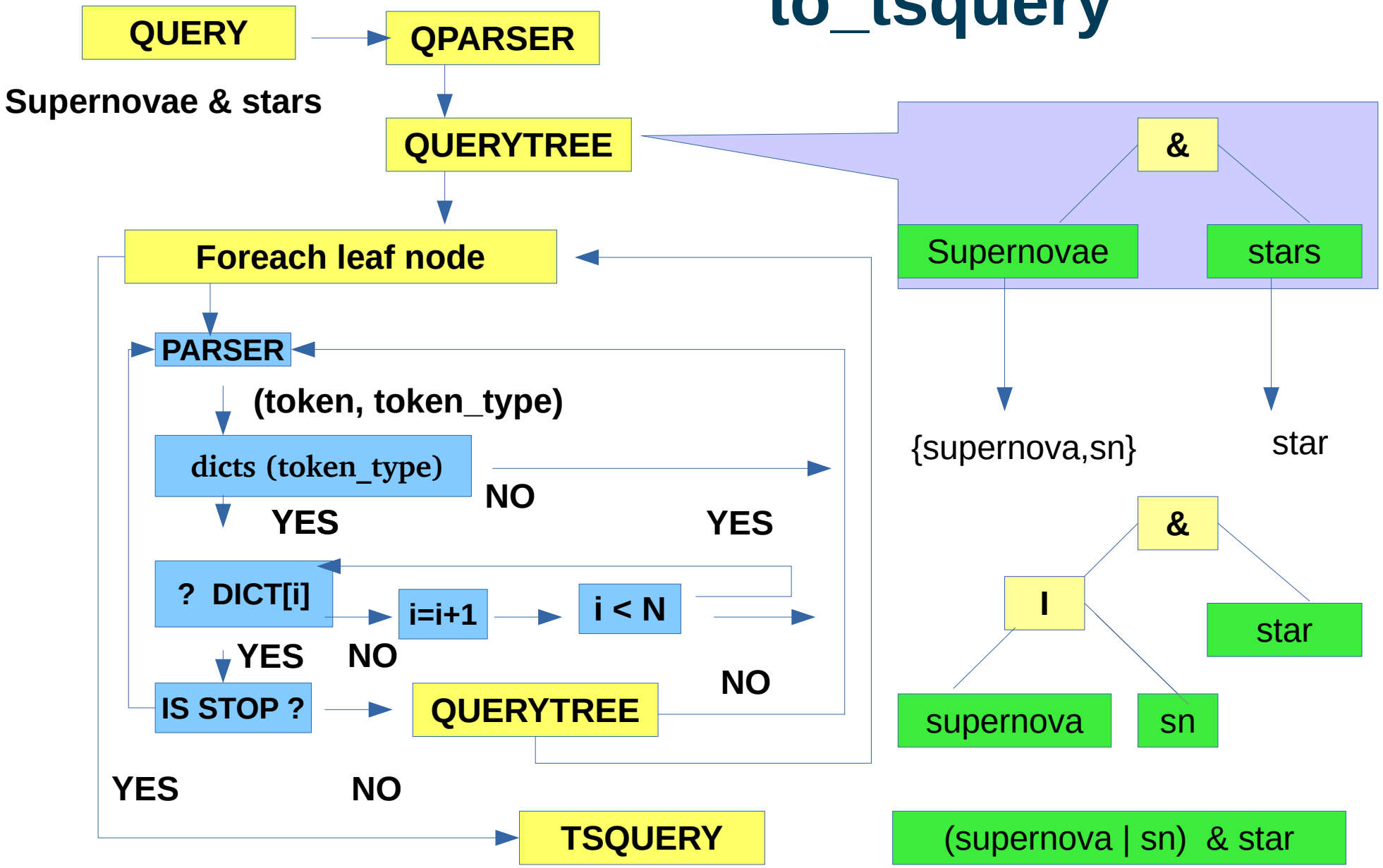
FTS in PostgreSQL

What is the benefit ?

Document processed only once when inserting to table, no overhead in search

- Document parsed into tokens using pluggable parser
- Tokens converted to lexems using pluggable dictionaries
- Words positions and importance are stored and used for ranking
- Stop-words ignored

FTS PostgreSQL
to_tsquery



Query processing

- Query to tsquery:
  - to_tsquery([cfg], text)

    .

    Better, always specify *cfg* (immutable vs stable) !

```
select to_tsquery('supernovae & stars');
      to_tsquery
-----------------------
 'supernova' & 'star'
(1 row)
```

  - plainto_tsquery([cfg],text) – words are AND-ed

```
select plainto_tsquery('supernovae  stars');
   plainto_tsquery
-----------------------
 'supernova' & 'star'
(1 row)
```

Query processing

- Queries 'A & B'::tsquery and 'B & A'::tsquery are equivalent

```
select 'a:1 b:2'::tsvector @@ 'a & b'::tsquery,
       'a:1 b:2'::tsvector @@ 'b & a'::tsquery;
 ?column? | ?column?
----------+----------
 t        | t
```

- Phrase query: FOLLOWED BY operators <n>,<->
- Guarantee an order (and distance) of operands
- Precendence of tsquery operators - '! <-> & |'

```
select 'a:1 b:2'::tsvector @@ 'a <-> b'::tsquery,
       'a:1 b:2'::tsvector @@ 'b <-> a'::tsquery;
 ?column? | ?column?
----------+----------
 t        | f
```

Phrase search - properties

- Precendence of tsquery operators - '! <-> & |'
  Use parenthesis to control nesting in tsquery

```
select 'a & b <-> c'::tsquery;
      tsquery
--------------------
 'a' & 'b' <-> 'c'

select 'b <-> c & a'::tsquery;
      tsquery
--------------------
 'b' <-> 'c' & 'a'

 select 'b <-> (c & a)'::tsquery;
           tsquery
----------------------------
 'b' <-> 'c' & 'b' <-> 'a'
```

Phrase search - example

- phraseto_tsquery([CFG,] TEXT)

```
select phraseto_tsquery('english','PostgreSQL can be extended
by the user in many ways');
                        phraseto_tsquery
-----------------------------------------------------------
 'postgresql' <3> 'extend' <3> 'user' <2> 'mani' <-> 'way'
(1 row)
```

Stop words are taken into account !

- It’s possible to combine tsquery’s

```
select phraseto_tsquery('PostgreSQL can be extended by the user in many ways') ||
                to_tsquery('oho<->ho & ik');
                                ?column?
-------------------------------------------------------------------------------------------
 'postgresql' <3> 'extend' <3> 'user' <2> 'mani' <-> 'way' | 'oho' <-> 'ho' & 'ik'
(1 row)
```

Query processing

- websearch_to_tsquery([cfg], text)
  - Recognizes “phrases”, AND, OR, *, +word, -word

```
select websearch_to_tsquery('english','postgresql "open source *
database" -die +most');
                    websearch_to_tsquery
-------------------------------------------------------------
 'postgresql' & 'open' <-> 'sourc' <2> 'databas' & !'die'
(1 row)


select to_tsvector('english', 'PostgreSQL: The Worlds Most Advanced
Open Source Relational Database') @@
websearch_to_tsquery('english','postgresql "open source * database" -
die +most');
 ?column?
----------
 t
(1 row)
```

FTS: additional functions

- ts_debug(cfg, text) – good for debugging FTS configuration
- ts_stat – word frequencies
- ts_parse(parser, text) – produces (token_type, token) from a text
- ts_rewrite – rewrite query online, no reindexing needed
- ts_headline – pieces of documents with words from query
- Ordering result of FTS:
  - ts_rank – the more occurrences of words,the bigger rank
    good for OR queries, no query language
  - ts_rank_cd – the closer words, the bigger rank
    good for AND queries, supports query language
  - rum_ts_score (requires RUM extension) – combination of the above, the best (NIST TREC, AD-HOC coll.)

FTS summary

- FTS in PostgreSQL is a flexible search engine,

- It is a «collection of bricks» you can build your search engine using
  - Custom parser
  - Custom dictionaries

  + All power of SQL (FTS+Spatial+Temporal)

Index — silver bullet !

the only weapon that is effective against a werewolf, witch, or other monsters.

Indexes !

- Index is a search tree with tuple pointers in the leaves
- Index has no visibility information (MVCC !)
- Indexes used only for accelerations:
  Index scan should produce the same results as sequence scan with filtering
- Indexes can be: **partial** (where price > 0.0), **functional** (to_tsvector(text)), **multicolumn** (timestamp, tsvector)
- Indexes not always useful !
  - Low selectivity
  - Maintainance overhead

FTS indexes

- CREATE INDEX … USING GIST/GIN/RUM (tsvector)
- GiST — Generalized Search Tree
  - document, query as a signature, documents → signature tree, Bloom filter used for search
- GIN — inverted tree, basically it“s a B-tree
  - Optimized for storing a lot of duplicate keys
  - Duplicates are ordered by heap TID
- RUM (extension)
  - GIN with addtional information (words positions, timestamp, ...)

FTS indexes

- GiST
  - document, query as a signature, documents → signature tree, Bloom filter used for search
  - Fast insert, small size, good for small collections
- GIN — inverted tree, basically it‘s a B-tree
  - Optimized for storing a lot of duplicate keys
  - Duplicates are ordered by heap TID
  - Not as fast as GiST for updates, good performance and scalability
- RUM (extension) — GIN++
  - Slow for updating, big size, high WAL traffic, best for mostly read-only workload, very fast for ranking, good for phrase search, no need tsvector column

Ispell shared dictionaries

- Working with dictionaries can be difficult and slow
  - Installing dictionaries can be complicated
  - Dictionaries are loaded into memory for every session (slow first query symptom) and eat memory.

```
time for i in {1..10}; do echo $i; psql postgres -c "select
ts_lexize('english_hunspell', 'evening')" > /dev/null;  done
1
2
3
4
5
6
7
8
9
10

real   0m0.656s
user   0m0.015s
sys0m0.031s
```

For russian hunspell dictionary:

real 0m3.809s
user0m0.015s
sys 0m0.029s

Each session «eats» 20MB !

Dictionaries as extensions

- Easy installation of hunspell dictionaries

```
CREATE EXTENSION hunspell_ru_ru; -- creates russian_hunspell dictionary
CREATE EXTENSION hunspell_en_us; -- creates english_hunspell dictionary
CREATE EXTENSION hunspell_nn_no; -- creates norwegian_hunspell dictionary
SELECT ts_lexize('english_hunspell', 'evening');
   ts_lexize
----------------
 {evening,even}
(1 row)

Time: 57.612 ms
SELECT ts_lexize('russian_hunspell', 'туши');
       ts_lexize
-----------------------
 {туша,тушь,тушить,туш}
(1 row)

Time: 382.221 ms
SELECT ts_lexize('norwegian_hunspell','fotballklubber');
          ts_lexize
------------------------------
 {fotball,klubb,fot,ball,klubb}
(1 row)

Time: 323.046 ms
```

Slow first query syndrom

Dictionaries in shared memory

```
CREATE EXTENSION shared_ispell;
CREATE TEXT SEARCH DICTIONARY english_shared (
    TEMPLATE = shared_ispell,
    DictFile = en_us,
    AffFile = en_us,
    StopWords = english
);
CREATE TEXT SEARCH DICTIONARY russian_shared (
    TEMPLATE = shared_ispell,
    DictFile = ru_ru,
    AffFile = ru_ru,
    StopWords = russian
);
time for i in {1..10}; do echo $i; psql postgres -c "select ts_lexize('russian_shared', 'туши')" > /dev/null;  done
1
2
…..
10
```

```
real  0m0.170s
user  0m0.015s
sys   0m0.027s
```

VS

```
real 0m3.809s
user0m0.015s
sys  0m0.029s
```

Search Mailing list archive

- https://postgrespro.com/list
- Custom parser — fixes several problems in default parser

```
select * from ts_parse('default','1914-1999');
 tokid | token
-------+-------
    22 | 1914
    21 | -1999
(2 rows)
```

```
select * from ts_parse('tsparser','1914-1999');
 tokid |   token
-------+-----------
    15 | 1914-1999
     9 | 1914
    12 | -
     9 | 1999
(4 rows)
```

```
select * from ts_parse('default','pg_catalog');
 tokid |  token
-------+---------
     1 | pg
    12 | _
     1 | catalog
(3 rows)
```

```
select * from ts_parse('tsparser','pg_catalog');
 tokid |   token
-------+------------
    16 | pg_catalog
    11 | pg
    12 | _
    11 | catalog
(4 rows)
```

Search Mailing list archive

- https://postgrespro.com/list
- Faceted search - grouping search results by lists
- Strip citation from posts
- Uses pg_trgm for suggestions
- Advanced query language
  - Support «phrase» search

PROFESSIONAL Postgres

Search Mailing list archive

server cras | Mailing lists | Search

- **server crash**
- **server crash**es
- **server crash**ing
- **server crash**ed
- **server crash**me

server crash - Search results in mailing lists

server crash

List: All lists

Post date: anytime

Sort by: Date

Search

pgsql-general (1037)

2018-10-16 21:25:54 | postgres **server** process **crashes** when using odbc_fdw (Ravi Krishna)

server. I also created foreign table. When I run a sql 'select * from odbctest' postgres **crashes**

Thread >> Search in thread (12)

2018-09-26 14:46:10 | Re: Setting up continuous archiving (Stephen Frost)

server crashes or there's some kind of issue with it after the rsync finishes

Thread

2018-08-29 04:02:45 | WAL replay issue from 9.6.8 to 9.6.10 (Dave Peticolas)

server to 9.6.8 and I was able to replay WAL past the point where 9.6.10 would PANIC and **crash**

Thread

2018-08-24 19:07:41 | Re: unorthodox use of PG for a customer (David Gauthier)

crash them. Of course any DB running would die too and have to be restarted/recovered. So the place for the DB is really elsewhere, on an external **server**

Thread

pgsql-hackers (1199)

2018-10-23 21:06:49 | Re: [HACKERS] Transactions involving multiple postgres foreignservers, take 2 (Masahiko Sawada)

PG13 for FTS

- Предвычисление immutable function использованных как таблица:
  Теперь можно писать спокойно без опаски Function Scan on q

```
SELECT rank(txtsample, q), * FROM
    test_tsquery, to_tsquery('english', 'new') q
WHERE txtsample @@ q
ORDER BT 1 desc;
```

PG13 for FTS

- Avoid full index scan for GIN ('w' & '!star')

References

- Slides of this talk
  http://www.sai.msu.su/~megera/postgres/talks/fts-pgconf.ru-2020.pdf
- FTS talk at PGConf.EU 2018
  http://www.sai.msu.su/~megera/postgres/talks/pgconf.eu-2018-fts.pdf
- Dictionaries as extensions
  https://github.com/postgrespro/hunspell_dicts
- Improved text search parser
  https://github.com/postgrespro/pg_tsparser
- RUM access method
  https://github.com/postgrespro/rum
- Shared ispell template
  https://github.com/postgrespro/shared_ispell
- Full text search example
  https://github.com/postgrespro/apod_fts
- Setrank - TF*IDF ranking
  https://github.com/obartunov/setrank

References

- Dictionary for regular expressions
  https://github.com/obartunov/dict_regex
- Dictionary for roman numbers
  https://github.com/obartunov/dict_roman
- Faceted search in one query
  http://akorotkov.github.io/blog/2016/06/17/faceted-search/
- FTS real example: Search mailing list archives
  https://postgrespro.com/list
- FTS slides with a lot of info
  http://www.sai.msu.su/~megera/postgres/talks/fts_postgres_by_authors_2.pdf
- Pg_trgm documentation
  https://www.postgresql.org/docs/11/static/pgtrgm.html
- My postings about FTS
  https://obartunov.livejournal.com/tag/fts

Спасибо за внимание !