Better Full Text Search in PostgreSQL

Oleg Bartunov
Postgres Professional, Moscow University
Artur Zakirov
Postgres Professional

PGConf.EU-2016, Tallinn
FTS in PostgreSQL

- FTS is a powerful built-in text search engine
- No new features since 2006!
- Popular complaints:
  - Slow ranking
  - No phrase search
  - No efficient alternate ranking
  - Working with dictionaries is tricky
  - Dictionaries are stored in the backend’s memory
  - FTS is flexible, but not enough
FTS in PostgreSQL

- **tsvector** — data type for document optimized for search
- **tsquery** — textual data type for rich query language
- **Full text search operator:** tsvector @@ tsquery
- **SQL interface** to FTS objects (CREATE, ALTER)
  - Configuration: {tokens, {dictionaries}}
  - Parser: {tokens}
  - Dictionary: tokens → lexeme{s}
- **Additional functions and operators**
- **Indexes:** GiST, GIN, RUM

```sql
to_tsvector('english','a fat cat sat on a mat and ate a fat rat')
@@
to_tsquery('english','(cats | rat) & ate & !mice');
```

http://www.postgresql.org/docs/current/static/textsearch.html
Some FTS problems: #1

156676 Wikipedia articles:

- Search is fast, ranking is slow.

```
SELECT docid, ts_rank(text_vector, to_tsquery('english', 'title')) AS rank
FROM ti2
WHERE text_vector @@ to_tsquery('english', 'title')
ORDER BY rank DESC
LIMIT 3;
```

Limit (actual time=476.106..476.107 rows=3 loops=1)
Buffers: shared hit=149804 read=87416
- Sort (actual time=476.104..476.104 rows=3 loops=1)
  Sort Key: (ts_rank(text_vector, '''titl'''::tsquery)) DESC
  Sort Method: top-N heapsort  Memory: 25kB
  Buffers: shared hit=149804 read=87416
- Bitmap Heap Scan on ti2 (actual time=6.894..469.215 rows=47855 loops=1)
  Recheck Cond: (text_vector @@ 'titl'::tsquery)
  Heap Blocks: exact=4913
  Buffers: shared hit=149804 read=87416
- Bitmap Index Scan on ti2_index (actual time=6.117..6.117 rows=47855 loops=1)
  Index Cond: (text_vector @@ 'titl'::tsquery)
  Buffers: shared hit=1 read=12

Planning time: 0.255 ms
Execution time: 476.171 ms
(15 rows)

HEAP IS SLOW 470 ms!
Some FTS problems: #2

- No phrase search

- “A & B” is equivalent to “B & A” »
  There are only 92 posts in -hackers with person 'Tom Good', but FTS finds 34039 posts

- FTS + regex is slow and can be used only for simple queries.
Some FTS problems:  #3

- Slow FTS with ordering by timestamp («fresh» results)

```sql
SELECT sent, subject from pglist
WHERE fts @@ to_tsquery('english', 'server & crashed')
  and sent < '2000-01-01'::timestamp
ORDER BY sent desc
LIMIT 5;
```

- Bitmap index scan by GIN (fts)
- Bitmap index scan by Btree (date)
- BitmapAND
- Bitmap Heap scan
- Sort
- Limit
- 10 ms
Inverted Index in PostgreSQL

No positions in index!
Improving GIN

• Improve GIN index
  • Store additional information in posting tree, for example, lexemes positions or timestamps
  • Use this information to order results
Improving GIN

9.6: CREATE AM GENERIC WAL

Create access methods RUM as extension!
CREATE INDEX ... USING RUM

- Use positions to calculate rank and order results
- Introduce distance operator `tsvector <=> tsquery`

CREATE INDEX ti2_rum_fts_idx ON ti2 USING rum(text_vector rum_tsvector_ops);

```
SELECT docid, ts_rank(text_vector, to_tsquery('english', 'title')) AS rank
FROM ti2
WHERE text_vector @@ to_tsquery('english', 'title')
ORDER BY
text_vector <=> plainto_tsquery('english','title') LIMIT 3;
```

QUERY PLAN
---

L Limit (actual time=54.676..54.735 rows=3 loops=1)
  Buffers: shared hit=355
  ->  Index Scan using ti2_rum_fts_idx on ti2 (actual time=54.675..54.733 rows=3 loops=1)
      Index Cond: (text_vector @@ '''titl'''::tsquery)
      Order By: (text_vector <=> '''titl'''::tsquery)
      Buffers: shared hit=355
Planning time: 0.225 ms
Execution time: 54.775 ms vs 476 ms !
CREATE INDEX ... USING RUM

• Top-10 (out of 222813) postings with «Tom Lane»
  • GIN index — 1374.772 ms

```
SELECT subject, ts_rank(fts, plainto_tsquery('english', 'tom lane')) AS rank
FROM pglist WHERE fts @@ plainto_tsquery('english', 'tom lane')
ORDER BY rank DESC LIMIT 10;
```

**QUERY PLAN**

<table>
<thead>
<tr>
<th>QUERY PLAN</th>
<th>Planning time: 0.287 ms</th>
<th>Execution time: 1374.772 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit (actual time=1374.277..1374.278 rows=10 loops=1)</td>
<td>(11 rows)</td>
<td></td>
</tr>
<tr>
<td>→ Sort (actual time=1374.276..1374.276 rows=10 loops=1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sort Key: (ts_rank(fts, ''tom'' &amp; ''lane''::tsquery)) DESC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sort Method: top-N heapsort Memory: 25kB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>→ Bitmap Heap Scan on pglist (actual time=98.413..1330.994 rows=222813 loops=1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recheck Cond: (fts @@ ''tom'' &amp; ''lane''::tsquery)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heap Blocks: exact=105992</td>
<td></td>
<td></td>
</tr>
<tr>
<td>→ Bitmap Index Scan on pglist_gin_idx (actual time=65.712..65.712 rows=222813 loops=1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Cond: (fts @@ ''tom'' &amp; ''lane''::tsquery)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CREATE INDEX ... USING RUM

- Top-10 (out of 222813) postings with «Tom Lane»
  - RUM index — 216 ms vs 1374 ms !!!

```
create index pglist_rum_fts_idx on pglist using rum(fts rum_tsvector_ops);

SELECT subject  FROM pglist WHERE fts @@ plainto_tsquery('tom lane')
ORDER BY fts <=> plainto_tsquery('tom lane') LIMIT 10;
```

QUERY PLAN

```
Limit (actual time=215.115..215.185 rows=10 loops=1)
  -> Index Scan using pglist_rum_fts_idx on pglist (actual time=215.113..215.183 rows=10 loops=1)
    Index Cond: (fts @@ plainto_tsquery('tom lane'::text))
    Order By: (fts <=> plainto_tsquery('tom lane'::text))
Planning time: 0.264 ms
Execution time: 215.833 ms
(6 rows)
```
CREATE INDEX ... USING RUM

- RUM uses new ranking function (ts_score) — combination of ts_rank and ts_rank_cd
  - ts_rank doesn't support logical operators
  - ts_rank_cd works poorly with OR queries

```sql
SELECT ts_rank(fts, plainto_tsquery('english', 'tom lane')) AS rank, ts_rank_cd (fts, plainto_tsquery('english', 'tom lane')) AS rank_cd, fts <=> plainto_tsquery('english', 'tom lane') as score, subject FROM pglist WHERE fts @@ plainto_tsquery('english', 'tom lane') ORDER BY fts <=> plainto_tsquery('english', 'tom lane') LIMIT 10;
```

<table>
<thead>
<tr>
<th>rank</th>
<th>rank_cd</th>
<th>score</th>
<th>subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.999637</td>
<td>2.02857</td>
<td>0.487904</td>
<td>Re: ATTN: Tom Lane</td>
</tr>
<tr>
<td>0.999224</td>
<td>1.97143</td>
<td>0.492074</td>
<td>Re: Bug #866 related problem (ATTN Tom Lane)</td>
</tr>
<tr>
<td>0.99798</td>
<td>1.97143</td>
<td>0.492074</td>
<td>Tom Lane</td>
</tr>
<tr>
<td>0.99653</td>
<td>1.57143</td>
<td>0.523388</td>
<td>happy birthday Tom Lane ...</td>
</tr>
<tr>
<td>0.99697</td>
<td>2.18825</td>
<td>0.570404</td>
<td>For Tom Lane</td>
</tr>
<tr>
<td>0.99638</td>
<td>2.12208</td>
<td>0.571455</td>
<td>Re: Favorite Tom Lane quotes</td>
</tr>
<tr>
<td>0.99188</td>
<td>1.68571</td>
<td>0.593533</td>
<td>Re: disallow LOCK on a view - the Tom Lane remix</td>
</tr>
<tr>
<td>0.99188</td>
<td>1.68571</td>
<td>0.593533</td>
<td>Re: disallow LOCK on a view - the Tom Lane remix</td>
</tr>
<tr>
<td>0.99188</td>
<td>1.68571</td>
<td>0.593533</td>
<td>Re: disallow LOCK on a view - the Tom Lane remix</td>
</tr>
<tr>
<td>0.99188</td>
<td>1.68571</td>
<td>0.593533</td>
<td>Re: [HACKERS] disallow LOCK on a view - the Tom Lane remix</td>
</tr>
<tr>
<td>(10 rows)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Phrase Search (8 years old!)

- Queries 'A & B'::tsquery and 'B & A'::tsquery produce the same result.
- Phrase search - preserve order of words in a query. Results for queries 'A & B' and 'B & A' should be different!
- Introduce new FOLLOWED BY (<>-) operator:
  - Guarantee an order of operands
  - Distance between operands

\[ a <n> b == a \& b \& (\exists \ i,j : \text{pos}(b)i - \text{pos}(a)j = n) \]
Phrase search - definition

- FOLLOWED BY operator returns:
  - false
  - true and array of positions of the `right` operand, which satisfy distance condition
- FOLLOWED BY operator requires positions

```
select 'a b c'::tsvector @@ 'a <-> b'::tsquery; -- false, there no positions
?column?
---------
f
(1 row)
select 'a:1 b:2 c'::tsvector @@ 'a <-> b'::tsquery;
?column?
---------
t
(1 row)
```
Phrase search - properties

- 'A <-> B' = 'A<1>B'
- 'A <0> B' matches the word with two different forms (infinitives)

```sql
=# SELECT ts_lexize('ispell','bookings');
  ts_lexize
 ---------------
 {booking,book}
 to_tsvector('bookings') @@ 'booking <0> book'::tsquery
```
Phrase search - properties

- Precendence of tsquery operators - '!' <-> & |'

Use parenthesis to control nesting in tsquery

```sql
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

- **TSQUERY phraseto_tsquery([[CFG,] TEXT])**
  Stop words are taken into account.

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

- **It’s possible to combine tsquery’s**

```sql
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)
```
Phrase search - internals

- Phrase search has overhead, since it requires access and operations on posting lists

\[(A \leftrightarrow B) \leftrightarrow (C \mid D) \land F\]

- We want to avoid slowdown FTS operators (\& |), which do not need positions.
- Rewrite query, so any \leftrightarrow operators pushed down in query tree and call phrase executor for the top \leftrightarrow operator.
Phrase search - transformation

\[( (A \leftrightarrow B) \leftrightarrow (C \mid D) ) \& F \]
Phrase search - push down

\[ a \leftrightarrow (b\&c) \Rightarrow a\leftrightarrow b \& a\leftrightarrow c \]
\[ (a\&b) \leftrightarrow c \Rightarrow a\leftrightarrow c \& b\leftrightarrow c \]
\[ a \leftrightarrow (b\mid c) \Rightarrow a\leftrightarrow b \mid a\leftrightarrow c \]
\[ (a\mid b) \leftrightarrow c \Rightarrow a\leftrightarrow c \mid b\leftrightarrow c \]
\[ a \leftrightarrow !b \Rightarrow a \& !(a\leftrightarrow b) \]
\[ \text{there is no position of A followed by B} \]
\[ !a \leftrightarrow b \Rightarrow !(a\leftrightarrow b) \& b \]
\[ \text{there is no position of B preceded by A} \]
Phrase search - transformation

```sql
# select '( A | B ) <-> ( D | C )'::tsquery;

# select 'A <-> ( B & ( C | ! D ) )'::tsquery;
```
Phrase search

1.1 mln postings (postgres mailing lists)

- Phrase search has overhead

```
select count(*) from pglist where fts @@ to_tsquery('english', 'tom <-> lane');
count
-------
 222777
(1 row)
```

<table>
<thead>
<tr>
<th>&lt;-&gt;(s)</th>
<th>&amp; (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential Scan: 2.6</td>
<td>2.2</td>
</tr>
<tr>
<td>GIN index: 1.1</td>
<td>0.48</td>
</tr>
<tr>
<td>RUM index: 0.5</td>
<td>0.48</td>
</tr>
</tbody>
</table>

- significant overhead
- solves the problem!
Some FTS problems: #3

- Slow FTS with ordering by timestamp ('fresh' results)
- Store timestamps in additional information in timestamp order!

```sql
create index pglist_fts_ts_order_rum_idx on pglist using rum(fts rum_tsvector_timestamp_ops, sent) WITH (attach = 'sent', to = 'fts', order_by_attach = 't');

select sent, subject from pglist
where fts @@ to_tsquery('server & crashed')
order by sent <= '2000-01-01'::timestamp limit 5;
```

- Index Scan by RUM (fts, sent)
- Limit
- **0.08 ms vs 10 ms**!
RUM vs GIN

- 6 mln classifies, real fts queries, concurrency 24, duration 1 hour
  - GIN — 258087 qph
  - RUM — 1885698 qph (7x speedup)
- RUM has no pending list (not implemented) and stores more data.

Insert 1 mln messages shows no significant overhead:

Time(min): GiST(10), GIN(10), GIN_no_fast(21), RUM(34)
WAL(GB): GiST(3.5), GIN(7.5), GIN_no_fast(24), RUM(29)
RUM vs GIN

- CREATE INDEX
  - GENERIC WAL (9.6) generates too big WAL traffic
**Inverse FTS (FQS)**

- Find queries, which match given document
- Automatic text classification

```sql
SELECT * FROM queries;
q | tag
-----------------------------------+-------
'supernova' & 'star' | sn
'black' | color
'big' & 'bang' & 'black' & 'hole' | bang
'spiral' & 'galaxi' | shape
'black' & 'hole' | color
(5 rows)
```

```sql
SELECT * FROM queries WHERE to_tsvector('black holes never exists before we think about them') @@ q;
q         | tag
------------------+-------
'black'          | color
'black' & 'hole' | color
(2 rows)
```
Inverse FTS (FQS)

- RUM index supported – store branches of query tree in addinfo

Find queries for the first message in postgres mailing lists

\d pg_query
Table "public.pg_query"
  Column | Type | Modifiers
----------+-------+----------
    q     | tsquery|          |
 count   | integer|          |
Indexes:
   "pg_query_rum_idx" rum (q) 33818 queries

select q from pg_query pgq, pglist where q @@ pglist.fts and pglist.id=1;
  q
--------------------------
'one' & 'one'
'postgresql' & 'freebsd'
(2 rows)
Inverse FTS (FQS)

- RUM index supported – store branches of query tree in addinfo

Find queries for the first message in postgres mailing lists

```sql
create index pg_query_rum_idx on pg_query using rum(q);
select q from pg_query pgq, pglist where q @@ pglist.fts and pglist.id=1;
```

**QUERY PLAN**

```
Nested Loop (actual time=0.719..0.721 rows=2 loops=1)
  ->  Index Scan using pglist_id_idx on pglist
      (actual time=0.013..0.013 rows=1 loops=1)
        Index Cond: (id = 1)
  ->  Bitmap Heap Scan on pg_query pgq
      (actual time=0.702..0.704 rows=2 loops=1)
        Recheck Cond: (q @@ pglist.fts)
        Heap Blocks: exact=2
        ->  Bitmap Index Scan on pg_query_rum_idx
            (actual time=0.699..0.699 rows=2 loops=1)
              Index Cond: (q @@ pglist.fts)
Planning time: 0.212 ms
Execution time: 0.759 ms
(10 rows)
```
Inverse FTS (FQS)

- RUM index supported – store branches of query tree in addinfo

Monstrous postings

```
select id, t.subject, count(*) as cnt into pglist_q from pg_query,
(select id, fts, subject from pglist) t where t.fts @@ q
group by id, subject order by cnt desc limit 1000;

select * from pglist_q order by cnt desc limit 5;

<table>
<thead>
<tr>
<th>id</th>
<th>subject</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>248443</td>
<td>Packages patch</td>
<td>4472</td>
</tr>
<tr>
<td>282668</td>
<td>Re: release.sgml, minor pg_autovacuum changes</td>
<td>4184</td>
</tr>
<tr>
<td>282512</td>
<td>Re: release.sgml, minor pg_autovacuum changes</td>
<td>4151</td>
</tr>
<tr>
<td>282481</td>
<td>release.sgml, minor pg_autovacuum changes</td>
<td>4104</td>
</tr>
<tr>
<td>243465</td>
<td>Re: [HACKERS] Re: Release notes</td>
<td>3989</td>
</tr>
</tbody>
</table>
```

(5 rows)
RUM vs GIN

CREATE INDEX

- GENERIC WAL(9.6) generates too big WAL traffic. It currently doesn't support shift. rum(fts, ts+order) generates 186 Gb of WAL!
- RUM writes WAL AFTER creating index

<table>
<thead>
<tr>
<th></th>
<th>gin</th>
<th>rum (fts)</th>
<th>rum(fts, ts)</th>
<th>rum(fts, ts+order)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create time</td>
<td>147 s</td>
<td>201</td>
<td>209</td>
<td>215</td>
</tr>
<tr>
<td>Size (mb)</td>
<td>2167/1302</td>
<td>534</td>
<td>980</td>
<td>1531</td>
</tr>
<tr>
<td>WAL (Gb)</td>
<td>0.9</td>
<td>0.68</td>
<td>1.1</td>
<td>1.5</td>
</tr>
</tbody>
</table>
RUM Todo

- Allow multiple additional info (lexemes positions + timestamp)
- Add support for arrays
- Improve ranking function to support TF/IDF
- Improve insert time (pending list ?)
- Improve GENERIC WAL to support shift

Availability:
- 9.6+ only: https://github.com/postgrespro/rum
Better FTS configurability

• The problem
  • Search multilingual collection requires processing by several language-specific dictionaries. Currently, logic of processing is hidden from user and example would not work.

  ```sql
  ALTER TEXT SEARCH CONFIGURATION multi_conf
  ALTER MAPPING FOR asciiword, asciihword, hword_asciipart, word, hword, hword_part
  WITH unaccent, german_ispell, english_ispell, simple;
  ```

• Logic of tokens processing in FTS configuration
  • Example: German-English collection

  ```sql
  ALTER TEXT SEARCH CONFIGURATION multi_conf
  ALTER MAPPING FOR asciiword, asciihword, hword_asciipart, word, hword, hword_part
  WITH unaccent THEN (german_ispell AND english_ispell) OR simple;
  ```
Some FTS problems #4

- 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
```

<table>
<thead>
<tr>
<th></th>
<th>real</th>
<th>user</th>
<th>sys</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0m0.656s</td>
<td>0m0.015s</td>
<td>0m0.031s</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For russian hunspell dictionary:

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

Each session «eats» 20MB of RAM !
Dictionaries in shared memory

- Now it’s easy (Artur Zakirov, Postgres Professional + Thomas Vondra)
  https://github.com/postgrespro/shared_ispell

```sql
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       VS       real 0m3.809s
user 0m0.015s       VS       user0m0.015s
sys  0m0.027s       VS       sys  0m0.029s
Dictionaries as extensions

- Now it's easy (Artur Zakirov, Postgres Professional)
  https://github.com/postgrespro/hunspell_dicts

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)

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

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

Time: 57.612 ms

Slow first query syndrom
Tsvector editing functions

- Stas Kelvich (Postgres Professional)
- `setweight(tsvector, 'char', text[])` - add label to lexemes from text[] array

```sql
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

```sql
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)

```sql
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

```sql
select tsvector_to_array( to_tsvector('english', '20-th anniversary of PostgreSQL'));

<table>
<thead>
<tr>
<th>tsvector_to_array</th>
</tr>
</thead>
<tbody>
<tr>
<td>{20,anniversari,postgresql,th}</td>
</tr>
</tbody>
</table>
```

(1 row)
Tsvector editing functions

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

```sql
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)
```
Thanks !