RUM index and its applications

Alexander Korotkov
Postgres Professional

PGConf.RU 2017, Moscow

Inverted Index for fulltext search

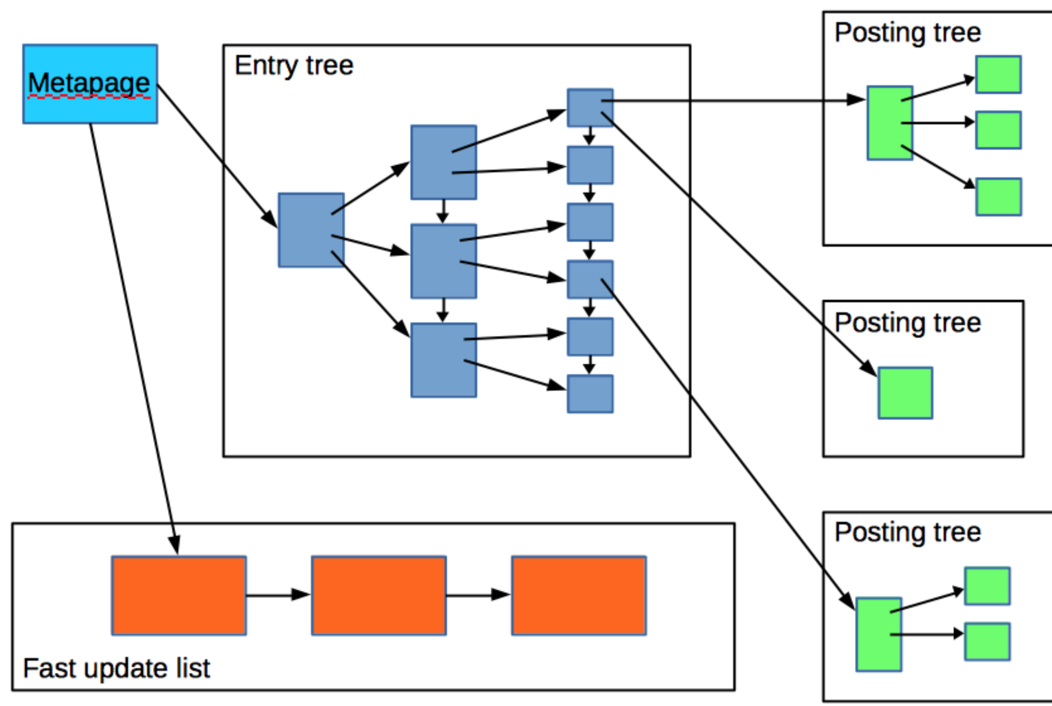
ENTRY TREE

Report Index

A

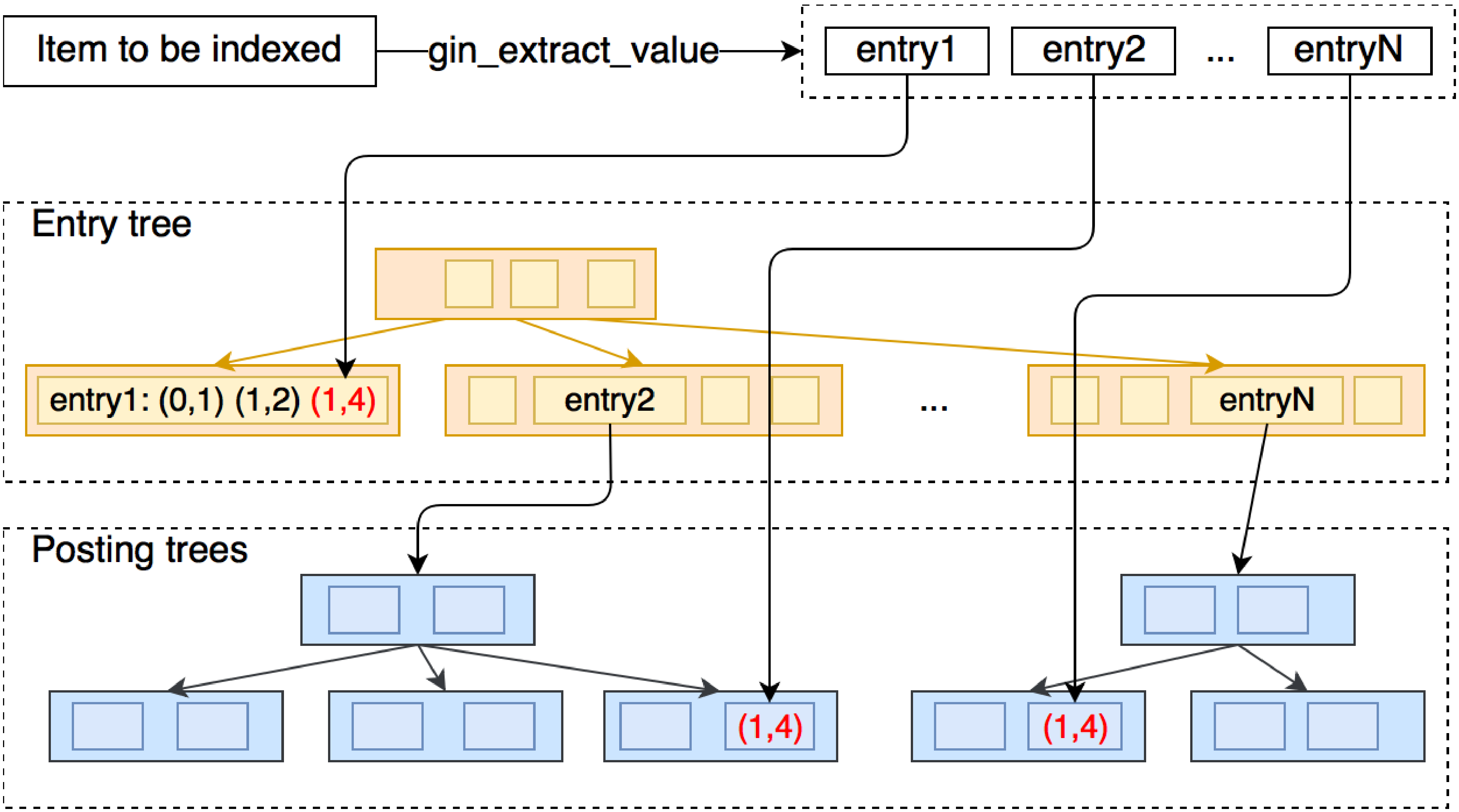
B

Posting list
Posting tree



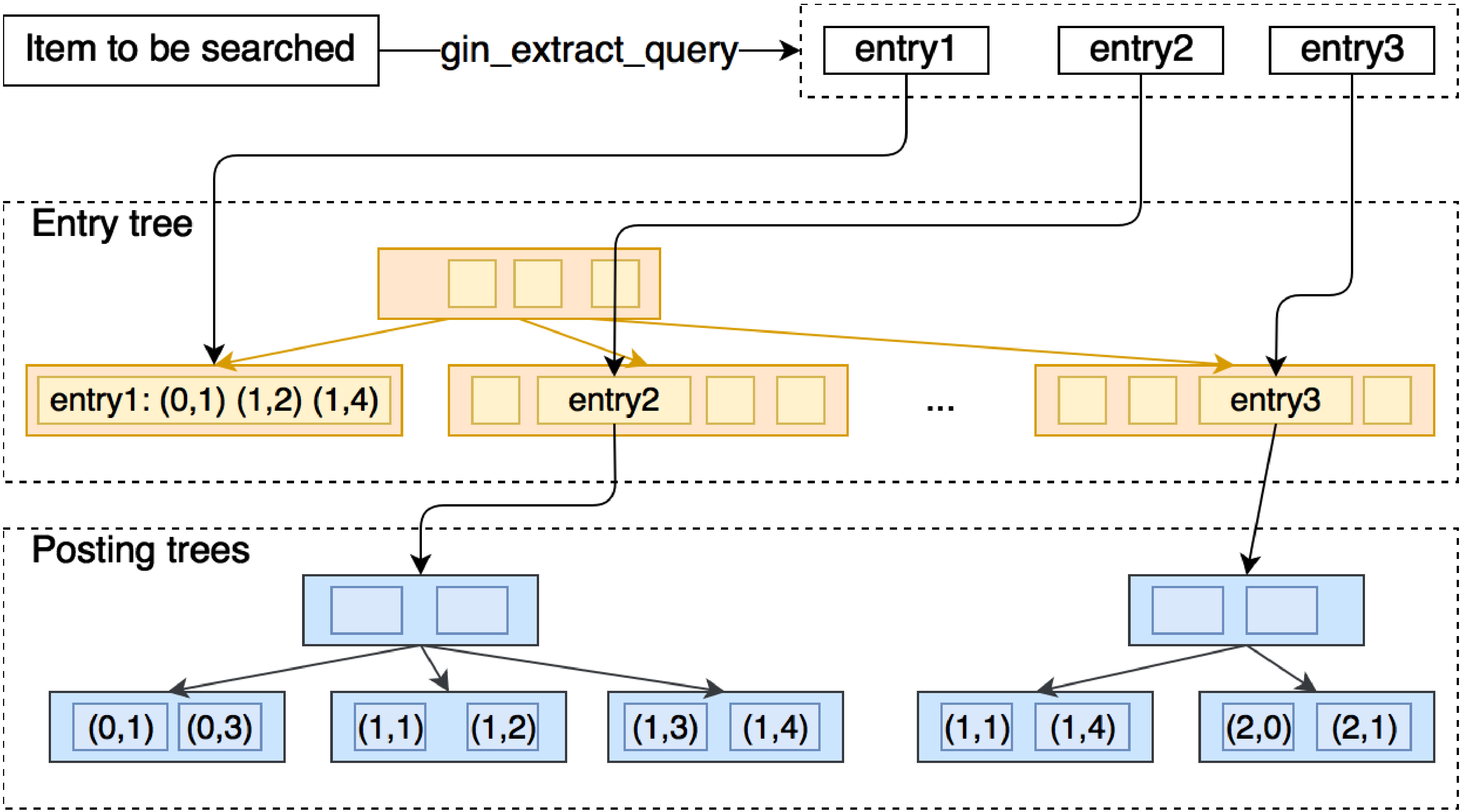
Generalized inverted index (1/3)

Inserting of new item to the index.



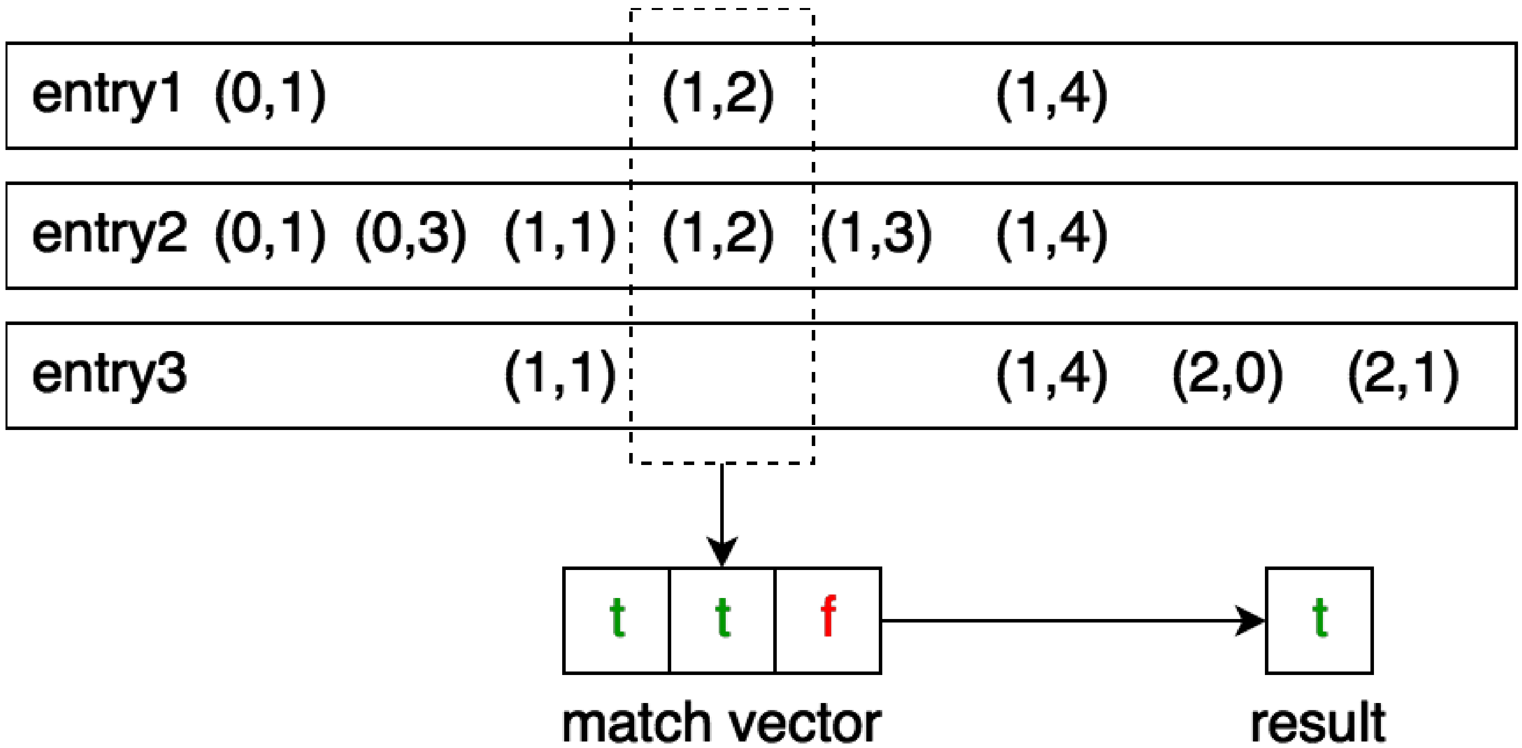
Generalized inverted index (2/3)

GIN search: finding corresponding posting lists/trees

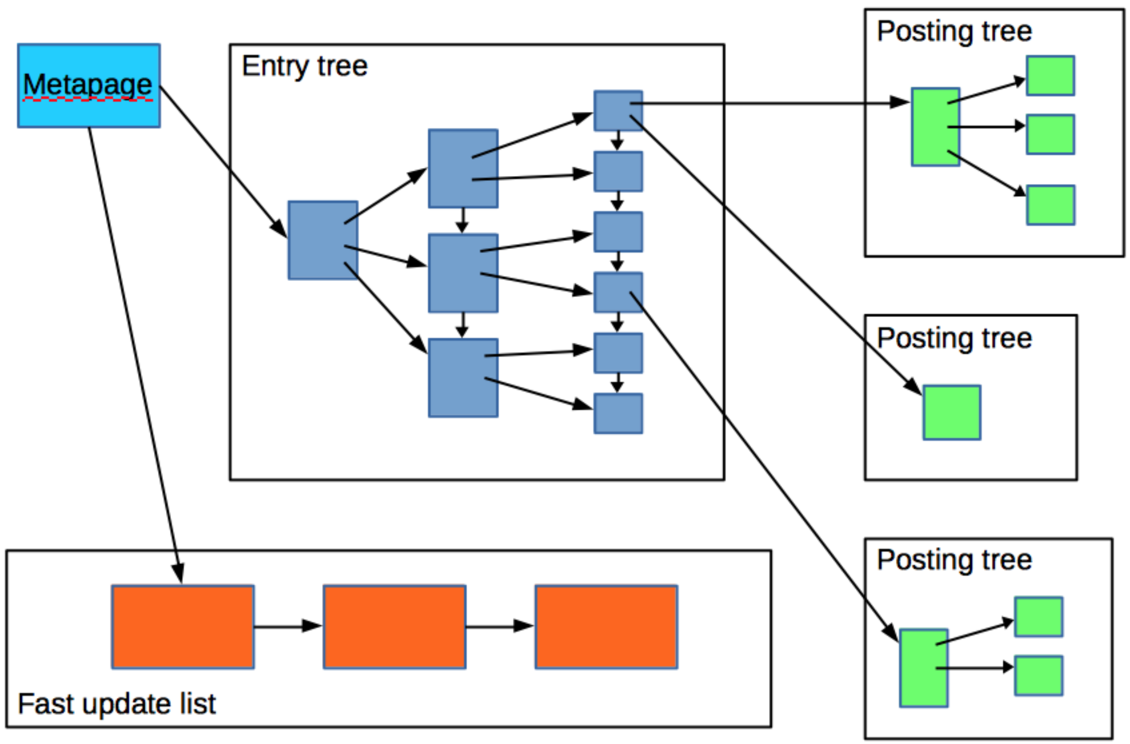


Generalized inverted index (3/3)

GIN search: filtering results

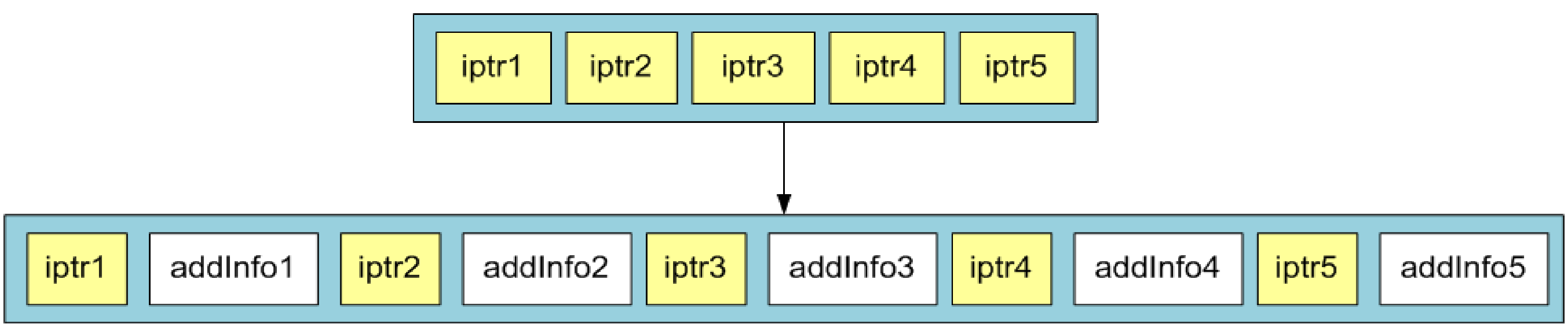


Improving GIN



9.6: CREATE AM
GENERIC WAL

Create access methods
RUM as extension !



How did things start?

I was trying this starting from 2012!!!

From:	Alexander Korotkov <aekorotkov(at)gmail(dot)com>
To:	pgsql-hackers <pgsql-hackers(at)postgresql(dot)org>
Subject:	WIP: store additional info in GIN index
Date:	2012-11-18 21:54:53
Message-ID:	CAPpHfdtSt47PpRQBK6OawHePLJk8PF-wNhswaUpre7_+cc_kmA@mail.gmail.com (view raw)

```
Hackers,

Attached patch enables GIN to store additional information with item
pointers in posting lists and trees.
Such additional information could be positions of words, positions of
trigrams, lengths of arrays and so on.
This is the first and most huge patch of serie of GIN improvements which
was presented at PGConf.EU
http://wiki.postgresql.org/images/2/25/Full-text_search_in_PostgreSQL_in_milliseconds-extended-
version.pdf

Patch modifies GIN interface as following:
1) Two arguments are added to extractValue
Datum **addInfo, bool **addInfoIsNull
2) Two arguments are added to consistent
Datum addInfo[], bool addInfoIsNull[]
3) New method config is introduced which returns datatype oid of addtional
information (analogy with SP-GiST config method).

Patch completely changes storage in posting lists and leaf pages of posting
trees. It uses varbyte encoding for BlockNumber and OffsetNumber.
BlockNumber are stored incremental in page. Additionally one bit of
OffsetNumber is reserved for additional information NULL flag. To be able
to find position in leaf data page quickly patch introduces small index in
the end of page.

------
With best regards,
Alexander Korotkov.
```

RUM applications

- Fulltext indexing with positional information (offsets of lexemes inside document)
- Jsonb indexing with positional information (offsets of elements in array)
- Inversed fulltext search (find queries matching given document)
- Inversed regex search (find regexes matching given
- Similarity indexing with lengths of arrays

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

```
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

GIN indexing: ranking from heap

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;

HEAP IS SLOW
470 ms !

```
 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
                     Index Cond: (text_vector @@ '''titl'''::tsquery)
                     Buffers: shared hit=1 read=12
 Planning time: 0.255 ms
 Execution time: 476.171 ms
(15 rows)
```

RUM indexing: ranking from index

- 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 !
(8 rows)
```

GIN indexing: ranking from heap

- 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
------------------------------------------------------------------------------------------
 Limit (actual time=1374.277..1374.278 rows=10 loops=1)
   ->  Sort (actual time=1374.276..1374.276 rows=10 loops=1)
         Sort Key: (ts_rank(fts, '''tom'' & ''lane'''::tsquery)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  Bitmap Heap Scan on pglist (actual time=98.413..1330.994 rows=222813 loops=1)
               Recheck Cond: (fts @@ '''tom'' & ''lane'''::tsquery)
               Heap Blocks: exact=105992
               ->  Bitmap Index Scan on pglist_gin_idx (actual time=65.712..65.712
rows=222813 loops=1)
                     Index Cond: (fts @@ '''tom'' & ''lane'''::tsquery)
 Planning time: 0.287 ms
 Execution time: 1374.772 ms
(11 rows)
```

RUM indexing: ranking from heap

- 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)
```

ts_score ranking

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

```
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;

   rank    | rank_cd |  score   |                          subject
----------+---------+----------+------------------------------------------------------------
 0.999637 | 2.02857 | 0.487904 | Re: ATTN: Tom Lane
 0.999224 | 1.97143 | 0.492074 | Re: Bug #866 related problem (ATTN Tom Lane)
  0.99798 | 1.97143 | 0.492074 | Tom Lane
 0.996653 | 1.57143 | 0.523388 | happy birthday Tom Lane ...
 0.999697 | 2.18825 | 0.570404 | For Tom Lane
 0.999638 | 2.12208 | 0.571455 | Re: Favorite Tom Lane quotes
 0.999188 | 1.68571 | 0.593533 | Re: disallow LOCK on a view - the Tom Lane remix
 0.999188 | 1.68571 | 0.593533 | Re: disallow LOCK on a view - the Tom Lane remix
 0.999188 | 1.68571 | 0.593533 | Re: disallow LOCK on a view - the Tom Lane remix
 0.999188 | 1.68571 | 0.593533 | Re: [HACKERS] disallow LOCK on a view - the Tom Lane remix
(10 rows)
```

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

$$a <n> b == a \& b \& (\exists\ i,j : pos(b)i - pos(a)j = n)$$

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)

```
=# 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

```
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.

```
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

```
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

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)

                                   <->(s)  |  & (s)
-----------------------------------+---------
Sequential Scan:   2.6  | 2.2
GIN index:         1.1  | 0.48       - significant overhead
RUM index:         0.5  | 0.48       - solves the problem !
```

Alternative posting lists/trees ordering

- FTS with ordering by timestamp («fresh» results)
  - Store timestamps in additional information
  - Order posting lists/trees by timestamp
  - **No sort needed!**

```
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)

0.08 ms (RUM no sort) vs **10 ms** (GIN + sort)!

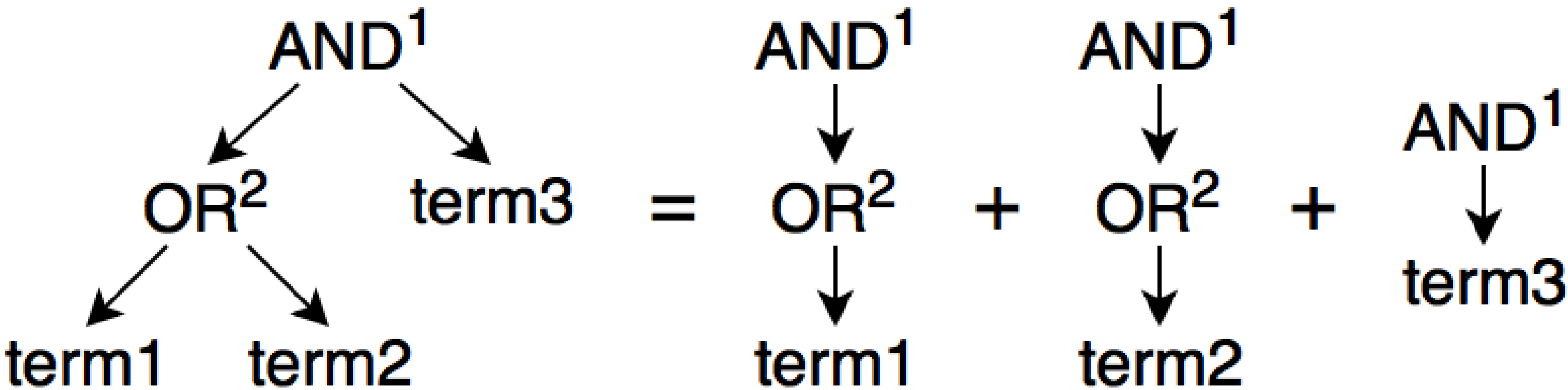
Inverse FTS (FQS)

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

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


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)



- term1: (AND 1, OR 2)
- term2: (AND 1, OR 2)
- term3: (AND 1)

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

```
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;
   id   |                        subject                        | cnt
--------+-----------------------------------------------+------
 248443 | Packages patch                                | 4472
 282668 | Re: release.sgml, minor pg_autovacuum changes | 4184
 282512 | Re: release.sgml, minor pg_autovacuum changes | 4151
 282481 | release.sgml, minor pg_autovacuum changes     | 4104
 243465 | Re: [HACKERS] Re: Release notes               | 3989
(5 rows))
```

RUM vs GIN

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

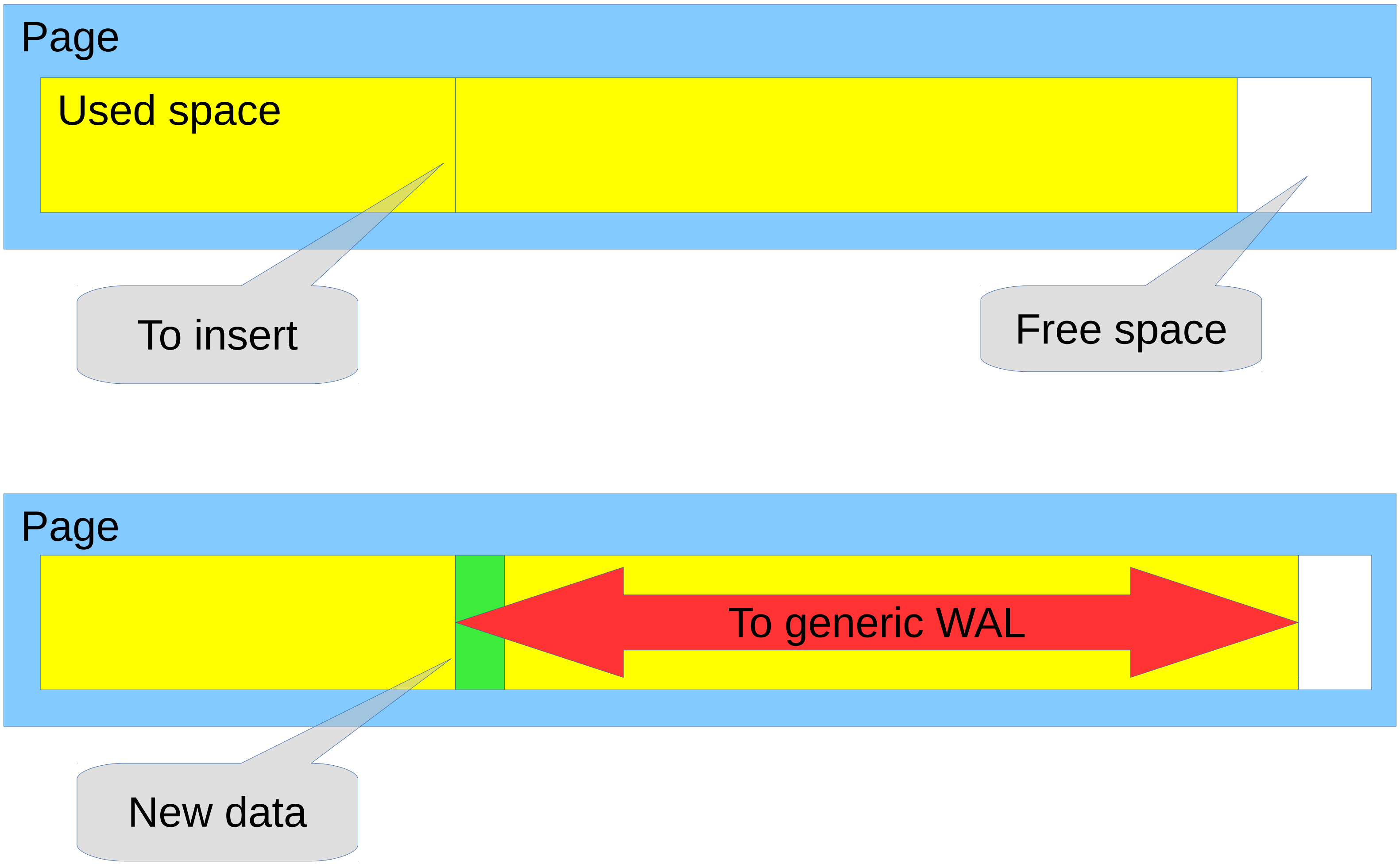
  Insert 1 mln messages shows no significant overhead:

  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



RUM vs GIN

- CREATE INDEX
  - GENERIC WAL(9.6) generates too big WAL traffic.

    It currently doesn't supports shift.
    rum(fts, ts+order) generates 186 Gb of WAL !
  - RUM writes WAL AFTER creating index

	table	gin	rum (fts	rum(fts,ts)	rum(fts,ts+order
Create time		147 s	201	209	215
Size(mb)	2167/1302	534	980	1531	1921
WAL (Gb)		0.9	0.68	1.1	1.5

RUM for jsonb

```
 ctid  |              value
-------+----------------------------
 (0,1) | '{"array": [1, 2, 3]}'
 (0,2) | '{"array": [2, 3]}'
```

GIN — no information about array elements positions!

```
array.#.1: (0,1)
array.#.2: (0,1); (0,2)
array.#.3: (0,1); (0,2)
```

RUM — with information about array elements positions!

```
array.#.1: (0,1) | 0
array.#.2: (0,1) | 1; (0,2) | 0
array.#.3: (0,1) | 2; (0,2) | 1
```

RUM vs GIN for jsonb

```
# EXPLAIN (ANALYZE, BUFFERS) SELECT count(*) FROM js  -- GIN
  WHERE js @@ 'tags.#16.term = "design"'::jsquery;
 Aggregate  (cost=4732.10..4732.11 rows=1 width=8) (actual time=101.047..101.047 rows=1 loo
   Buffers: shared hit=55546
   ->  Bitmap Heap Scan on js  (cost=33.71..4728.97 rows=1253 width=0) (actual time=35.495
         Recheck Cond: (js @@ '"tags".#16."term" = "design"'::jsquery)
         Rows Removed by Index Recheck: 64490
         Heap Blocks: exact=55525
         Buffers: shared hit=55546
         ->  Bitmap Index Scan on js_gin_idx  (cost=0.00..33.40 rows=1253 width=0) (actual
               Index Cond: (js @@ '"tags".#16."term" = "design"'::jsquery)
               Buffers: shared hit=21
 Planning time: 0.104 ms
 Execution time: 101.447 ms

# EXPLAIN (ANALYZE, BUFFERS) SELECT count(*) FROM js  -- RUM
  WHERE js @@ 'tags.#16.term = "design"'::jsquery;
 Aggregate  (cost=4732.10..4732.11 rows=1 width=8) (actual time=5.818..5.818 rows=1 loops=1
   Buffers: shared hit=71
   ->  Bitmap Heap Scan on js  (cost=33.71..4728.97 rows=1253 width=0) (actual time=5.804.
         Recheck Cond: (js @@ '"tags".#16."term" = "design"'::jsquery)
         Heap Blocks: exact=10
         Buffers: shared hit=71
         ->  Bitmap Index Scan on js_rum_idx  (cost=0.00..33.40 rows=1253 width=0) (actual
               Index Cond: (js @@ '"tags".#16."term" = "design"'::jsquery)
               Buffers: shared hit=61
 Planning time: 0.057 ms
 Execution time: 5.860 ms
```

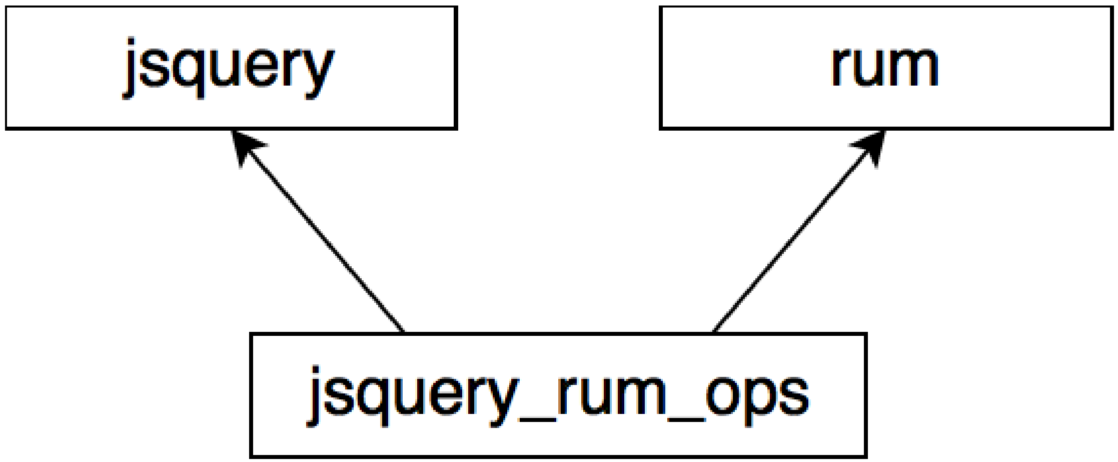
17 times faster!!!

RUM vs GIN for jsonb

```
 Object     | Size    | Build time
-----------+---------+------------
 Table      | 1369 MB |
 GIN index  |  411 MB |   80,2 sec
 RUM index  |  516 MB |   86,6 sec
```

RUM for jsonb appears to be not much bigger (25%) than GIN for jsonb.

rum_jsquery_ops: extension dependencies



RUM for jsonb is coming soon!

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


Thanks !

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“nt works.

```
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

```
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
1
2
3
4
5
6
7
8
9
10

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

For russian hunspell dictionary:

real 0m3.809s
user0m0.015s
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

```
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
```

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)

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

Tsvector editing functions

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

```
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
  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)
```