Some recent advances in full-text search

Oleg Bartunov, Teodor Sigaev

Sternberg Astronomical Institute, Moscow State University, Russia
Talk roadmap

- Full-text search introduction
- Main topics
  - Phrase Search
  - Dictionaries API
- New features (already in 8.4)
- Future features
- Tips and Tricks
Full-text search in PostgreSQL

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

- **tsvector** – storage for document
  - sorted array of lexemes with optional positional and weight information
- **tsquery** – textual data type for query
  - Boolean operators - & | ! ()
  - 'telefonsvarer' => 
  - 'telefonsvarer' | 'telefon' & 'svar'
- **FTS operator**
  - `tsvector @@ tsquery`
- **to_tsvector, to_tsquery, plainto_tsquery**
- Indexes: GiST, GIN
Talk roadmap

- Full-text search introduction
- Main topics
  - **Phrase Search**
  - Dictionaries API
- New features (already in 8.4)
- Future features
- Tips and Tricks
Phrase search - definition

A $ B: \text{ word 'A' followed by 'B':}
- A & B (the same priority)
- exists at least one pair of positions $P_B$, $P_A$, so that $0 \leq P_B - P_A \leq 1$ (distance condition)

A $[n] B: \quad 0 \leq P_B - P_A \leq n$

Result of operation:
- false
- true and array of positions of left argument which satisfy distance condition (without positional information $\$ is equivalent to &)

$\$ is very similar to & except: A $ B \neq B \$ A
Phrase search - properties

'A \([n] B \([m] C' \rightarrow \'(A \([n] B) \([m] C' \rightarrow \\
matched phrase length \leq \max(n, m) \\
Note: 'A C B' matched by '\(A \([2] B) \$ C'

'A \([n] (B \([m] C') \rightarrow \\
matched phrase length \leq n + m \\
Note: Order is preserved for any n, m

'A \([0] B' matches the word with two different forms ( infinitives )

=\# \text{SELECT ts_lexize('ispell','bookings');}
  ts_lexize
-----------------
{booking,book}
to_tsvector('bookings') @@ 'booking \([0] book'::tsquery
Phrase search - practice

Phrase:
- 'A B C' → 'A $ (B $ C)'
- 'A B C' → '(A $ B) $[2] C'
- TSQUERY phraseto_tsearch([CFG[,] TEXT)


What shall we do with complex queries?
A $ ( B & ( C | ! D ) ) → ???
Phrase search has overhead, since it requires access and operations on posting lists.

To avoid slowdown of existing tsearch, executor of tsquery should not access positions without necessity. To facilitate this, any $ operations pushed down in query tree, so tsearch executor can call special phrase executor for the top $ operation, which will work only with query tree containing only $ operations.
Phrase search - transformation

$( (A \$ B) \$ (C \mid D) ) \& F$

Regular tree

Phrase top

Phrase tree
Phrase search - push down

\[ a \$ (b \& c) \Rightarrow (a \$ b) \& (a \$ c) \]

\[ (a \& b) \$ c \Rightarrow (a \$ c) \& (b \$ c) \]

\[ a \$ (b \| c) \Rightarrow (a \$ b) \| (a \$ c) \]

\[ (a \| b) \$ c \Rightarrow (a \$ c) \| (b \$ c) \]

\[ a \$ !b \Rightarrow a \& !(a \$ b) \]

there is no position of A followed by B

\[ !a \$ b \Rightarrow b \& !(a \$ b) \]

there is no position of B preceded by A
Phrase search - transformation

```
# select ' ( A | B ) $ ( D | C ) '::tsquery;

  tsquery

  'A' $ 'D' | 'B' $ 'D' | 'A' $ 'C' | 'B' $ 'C'

# select 'A $ ( B & ( C | ! D ) ) '::tsquery;

  tsquery

  ( 'A' $ 'B' ) & ( 'A' $ 'C' | 'A' & !( 'A' $ 'D' ) )
```
'PostgreSQL can be extended by the user in many ways' ->

# select phraseto_tsquery('PostgreSQL can be extended by the user in many ways');

Can be written by hand:


Difficult to modify, use phraseto_tsquery() function!
Phrase search - TODO

Ranking functions

Headline generation

Rewrite subsystem

Concatenation of two tsquery by $ operation: $$ ?
  - like other concatenations: &&, || and !!

Need testing for agglutinative languages
  (norwegian, german, etc)
Talk roadmap

- Full-text search introduction
- Main topics
  - Phrase Search
  - Dictionaries API
- New features (already in 8.4)
- Future features
- Tips and Tricks
## Dictionaries

<table>
<thead>
<tr>
<th>Lexeme's type</th>
<th>Dict #1</th>
<th>Dict #2</th>
<th>Dict #N</th>
</tr>
</thead>
<tbody>
<tr>
<td>asciiword</td>
<td>synonym</td>
<td>en_ispell</td>
<td>en_stem</td>
</tr>
<tr>
<td>int</td>
<td>simple</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>real_dict</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# \dF+ english

Text search configuration "pg_catalog.english"

Parser: "pg_catalog.default"

<table>
<thead>
<tr>
<th>Token</th>
<th>Dictionaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>asciiihword</td>
<td>english_stem</td>
</tr>
<tr>
<td>asciiword</td>
<td>english_stem</td>
</tr>
<tr>
<td>email</td>
<td>simple</td>
</tr>
<tr>
<td>file</td>
<td>simple</td>
</tr>
</tbody>
</table>

....
Dictionaries - examples

Integers
'123456789' -> '123456'

Roman numbers
'XIX' -> '19'

Colours
'FFFFFFFF' -> 'white'

Regexp
H(\s|-)?(alpha|beta|gamma) h$2 — spectral lines of hydrogen
Dictionaries - interface

void* dictInit(List *dictoptions)
- list of dictoptions actually contains list of DefElem structures (see headers)
- returns pointer to the palloc'ed dictionary structure
- Can be expensive (ispell)

TSLexeme* dictLexize(
  void* dictData, // returned by dictInit()
  char* lexeme,  // not zero-terminated
  int lenlexeme, 
  DictSubState *substate // optional
);
typedef struct {
    uint16       nvariant; // optional
    uint16       flags;    // optional
    char         *lexeme;
} TSLexeme;

dictLexize returns NULL – dictionary doesn't recognize the lexeme

dictLexize returns array of TSLexeme
(last element TSLexeme->lexeme is NULL)

dictLexize returns empty array – dictionary recognizes the lexeme, but it's a stop-word
SELECT ts_lexize('en_ispell','bookings');

TSLexeme array:
#  nvariant flags  lexeme
0   0   0   booking
1   0   0   book
2   0   0   NULL
Agglutinative Languages

German, norwegian, ...
http://en.wikipedia.org/wiki/Agglutinative_language

Concatenation of words without space

Query - Fotballklubber
Document - Klubb on fotballfield

How to find document?

Split words and build search query
'fotbalklubber' =>
'( fotball & klubb ) | ( fot & ball & klubb )'
Agglutinative languages have several variants of word's splitting:

Word 'foobarcom' (imaginary)

<table>
<thead>
<tr>
<th>Lexeme</th>
<th>nvariant</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>bar</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>com</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>foob</td>
<td>2</td>
<td>-a- is an affix</td>
</tr>
<tr>
<td>rcom</td>
<td>2</td>
<td>(interfix)</td>
</tr>
</tbody>
</table>

tsvector: 'bar:1 com:1 foo:1 foob:1 rcom:1'
tsquery: '(foob & rcom) | (foo & bar & com)'
Each TSLexeme describes one normalized lexeme

TSLexeme->flags is an OR-ed:
- TSL_PREFIX indicates to use prefix search for this lexeme
  Note: dictionaries are planned for 8.5
- TSL_ADDPOS points to parser to increase position's counter
  Note: currently only thesaurus dictionary uses it
Dictionaries – output (imaginary)

Result of 'foobar'

Variant #1
- lexeme: foo
  - flags: 0
  - nvariant: 1
- Lexeme: first
  - Flags: 0
  - nvariant: 2

Variant #2
- lexeme: bar
  - flags: TSL_PREFIX
  - nvariant: 1
- Lexeme: oobar
  - Flags: TSL_PREFIX | TSL_ADDPOS
  - nvariant: 2

tsvector: 'foo:1 bar:1 first:1 oobar:2'

tquery: '(foo & bar:*)) | (first & oobar:*)'
Dictionaries - several words

typedef struct {
    bool isend; // in: marks end of text
    // (input lexeme is invalid!)
    bool getnext; // out: dictionary asks for
    // a next lexeme
    void *private; // internal state of
    // dictionary while it's
    // asking a next lexeme
} DictSubState;
Dictionaries – several words

1. Current lexeme
2. dictLexize
3. Result?
   - Y: Store result and current lexeme
   - N: Get next?
4. Get next?
   - Y: Has stored result?
     - Y: Go to the next dictionary with current lexeme
     - N: Store result and current lexeme
   - N: Set next lexeme as current lexeme
5. Has stored result?
   - Y: Go to the next dictionary with current lexeme
   - N: Store result and current lexeme
New TSLexeme->flags: TSL_FILTER

If dictionary returns only one lexeme with TSL_FILTER flag, then that lexeme will be used as an input for the subsequent dictionaries in the chain.
Filter dictionary – unaccent (8.5)

contrib/unaccent provides 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)

```sql
=# 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)

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

Finally, unaccent dictionary solves the known problem with headline!
( to_tsvector(remove_accent(document)) works with search, but
  has problem with highlighting )

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

- Full-text search introduction
- Main topics
  - Phrase Search
  - Dictionaries API
- New features (already in 8.4)
- Future features
- Tips and Tricks
New features and improvements

- ts_headline() enhancements (8.4)
- Prefix full-text search support (8.4)
- Devanagari script support (8.4)
- dict_xsyn improvement
- ts_stat() performance improvement (8.4)
- Fast approximated statistics (8.3, 8.4)
- GIN improvements: fast update (8.4), partial match support (8.4), multicolumn (8.4)
- contrib/btree_gin (8.4)
ts_headline enhancement

- New parameter MaxFragments by Sushant Sinha. Default is 0, ts_headline() generates one fragment

```sql
=# select ts_headline($$
    Text from http://www.postgresql.org/docs/8.3/static/history.html
$$,
    plainto_tsquery('postgresql postgres '), 'MaxFragments=3,
    MinWords=3, MaxWords=6');
    ts_headline
```

<b>PostgreSQL</b> is derived from the <b>POSTGRES</b> ... behind it, <b>PostgreSQL</b> ... <b>PostgreSQL</b> as "<b>Postgres</b>" (now rarely
Prefix full-text search support

- to_tsquery('supernov:*') - match all documents, which contains words with prefix 'supernov'
- to_tsquery('supernov:ab*') - the same, but only in titles (weight 'a') and keywords (weight 'b')
- Can use new GIN partial match feature to speedup search
- Can be useful if there is no stemmer available
PostgreSQL 8.3- has problem with Devanagari script (http://en.wikipedia.org/wiki/Devanagari - script for Hindi, Marathi, Nepali, Sanscrit,...).

```sql
select * from ts_parse('default', 'मदन पुरस्कार पुस्तकालय');
```

<table>
<thead>
<tr>
<th>2</th>
<th>मदन</th>
<th>Madan</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>पुरस्</td>
<td>Puraskar</td>
</tr>
<tr>
<td>12</td>
<td>कार</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>पुस्</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>तकालय</td>
<td></td>
</tr>
</tbody>
</table>

Virama sign (modifier, suppresses inherent vowel) – `punct` in np_NP locale. Breaks all parsers, which use locale.
### Devanagari Script Support

#### मदन पुरस्कार पुस्तकालय (Madan Puraskar Pustakalaya)

<table>
<thead>
<tr>
<th>character</th>
<th>byte</th>
<th>UTF-32</th>
<th>encoded as</th>
<th>glyph name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00092E</td>
<td>E0 A4 AE</td>
<td>म DEVANAGARI LETTER MA</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>000926</td>
<td>E0 A4 A6</td>
<td>द DEVANAGARI LETTER DA</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>000928</td>
<td>E0 A4 A8</td>
<td>न DEVANAGARI LETTER NA</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>000020</td>
<td>20</td>
<td>SPACE</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>00092A</td>
<td>E0 A4 AA</td>
<td>प DEVANAGARI LETTER PA</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>000941</td>
<td>E0 A5 81</td>
<td>DEVANAGARI VOWEL SIGN U</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>000930</td>
<td>E0 A4 B0</td>
<td>र DEVANAGARI LETTER RA</td>
</tr>
<tr>
<td>7</td>
<td>19</td>
<td>000938</td>
<td>E0 A4 B8</td>
<td>स DEVANAGARI LETTER SA</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>00094D</td>
<td>E0 A5 8D</td>
<td>DEVANAGARI SIGN VIRAMA</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>000915</td>
<td>E0 A4 95</td>
<td>क DEVANAGARI LETTER KA</td>
</tr>
<tr>
<td>10</td>
<td>28</td>
<td>00093E</td>
<td>E0 A4 BE</td>
<td>ा DEVANAGARI VOWEL SIGN AA</td>
</tr>
<tr>
<td>11</td>
<td>31</td>
<td>000930</td>
<td>E0 A4 B0</td>
<td>र DEVANAGARI LETTER RA</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
<td>000020</td>
<td>20</td>
<td>SPACE</td>
</tr>
<tr>
<td>13</td>
<td>35</td>
<td>00092A</td>
<td>E0 A4 AA</td>
<td>प DEVANAGARI LETTER PA</td>
</tr>
<tr>
<td>14</td>
<td>38</td>
<td>000941</td>
<td>E0 A5 81</td>
<td>DEVANAGARI VOWEL SIGN U</td>
</tr>
<tr>
<td>15</td>
<td>41</td>
<td>000938</td>
<td>E0 A4 B8</td>
<td>स DEVANAGARI LETTER SA</td>
</tr>
<tr>
<td>16</td>
<td>44</td>
<td>00094D</td>
<td>E0 A5 8D</td>
<td>DEVANAGARI SIGN VIRAMA</td>
</tr>
<tr>
<td>17</td>
<td>47</td>
<td>000924</td>
<td>E0 A4 A4</td>
<td>त DEVANAGARI LETTER TA</td>
</tr>
<tr>
<td>18</td>
<td>50</td>
<td>000915</td>
<td>E0 A4 95</td>
<td>क DEVANAGARI LETTER KA</td>
</tr>
<tr>
<td>19</td>
<td>53</td>
<td>00093E</td>
<td>E0 A4 BE</td>
<td>ा DEVANAGARI VOWEL SIGN AA</td>
</tr>
<tr>
<td>20</td>
<td>56</td>
<td>000932</td>
<td>E0 A4 B2</td>
<td>ल DEVANAGARI LETTER LA</td>
</tr>
<tr>
<td>21</td>
<td>59</td>
<td>00092F</td>
<td>E0 A4 AF</td>
<td>य DEVANAGARI LETTER YA</td>
</tr>
</tbody>
</table>
Devanagari script support

8.4 knows Virama signs
=# select * from ts_parse('default',
    'मदन पुरस्कार पु स्तकालय');

<table>
<thead>
<tr>
<th>tokid</th>
<th>token</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>मदन</td>
</tr>
<tr>
<td>12</td>
<td>पुरस्कार</td>
</tr>
<tr>
<td>2</td>
<td>पुरस्कार</td>
</tr>
<tr>
<td>12</td>
<td>पुरस्कार</td>
</tr>
<tr>
<td>2</td>
<td>पुस्तकालय</td>
</tr>
<tr>
<td>2</td>
<td>पुस्तकालय</td>
</tr>
</tbody>
</table>

(5 rows)

Thanks to Dibyendra Hyoju and Bal Krishna Bal for testing and valuable discussion
Devanagari script support

• TODO
  - Port stemmer for nepali to snowball
  - Improve Hunspell support (recognize more flags in affix file)
cat $SHAREDIR/tsearch_data/synonym_sample.syn
postgres  pgsql
postgresql  pgsql
postgre  pgsql
gogle  googl
indices index*

=# create text search dictionary syn
( template=synonym,synonyms='synonym_sample');
=# select ts_lexize('syn','indices');

----------
{index}
 Synonym dictionary with prefix search support (8.5)

```sql
=# create text search configuration tst ( copy=simple);
=# alter text search configuration tst alter mapping
    for asciiword with syn;

=# select to_tsquery('tst','indices');

to_tsquery

--------
'index':*

=# select 'indexes are very useful'::tsvector @@
    to_tsquery('tst','indices');

?column?

--------
t
```
dict_xsyn improvement

• How to search for 'William' and any synonyms 'Will', 'Bill', 'Billy'? We can:
  - Index only synonyms
  - Index synonyms and original name
  - Index only original name - replace all synonyms. Index size is minimal, but search for specific name is impossible.
dict_xsyn improvement

- Old version of dict_xsyn can return only list of synonyms. It's possible to prepare synonym file to support other options:

  William Will Bill Billy
  Will William Bill Billy
  Bill William Will Billy
  Billy William Will Bill

- New dict_xsyn (Sergey Karpov) allows better control:

  ```sql
  CREATE TEXT SEARCH DICTIONARY xsyn
  (RULES='xsyn_sample', KEEPORIG=false|true, mode='SIMPLE|SYMMETRIC|MAP');
  ```
dict_xsyn improvement

- Mode SIMPLE - accepts the original word and returns all synonyms as OR-ed list. This is default mode.

- Mode SYMMETRIC - accepts the original word or any of its synonyms, and return all others as OR-ed list.

- Mode MAP - accepts any synonym and returns the original word.
dict_xsyn improvement

EXAMPLES:

=# ALTER TEXT SEARCH DICTIONARY xsyn (RULES='xsyn_sample',
KEEPORIG=false, mode='SYMMETRIC');

=# select ts_lexize('xsyn','Will') as Will,
    ts_lexize('xsyn','Bill') as Bill,
    ts_lexize('xsyn','Billy') as Billy;

<table>
<thead>
<tr>
<th>will</th>
<th>bill</th>
<th>billy</th>
</tr>
</thead>
<tbody>
<tr>
<td>{william,bill,billy}</td>
<td>{william,will,billy}</td>
<td>{william,will,bill}</td>
</tr>
</tbody>
</table>

Mode='MAP'

<table>
<thead>
<tr>
<th>will</th>
<th>bill</th>
<th>billy</th>
</tr>
</thead>
<tbody>
<tr>
<td>{william}</td>
<td>{william}</td>
<td>{william}</td>
</tr>
</tbody>
</table>
ts_stat() performance!

- ts_stat() function gathers words statistics from tsvectors – now uses binary tree instead of sorted arrays (probably, better to use rbtree to defense against skewed data)

Dataset with geonames, total 5,793,013 rows with 2,404,197 unique names:

```sql
=# select * into ts_stat2
   from ts_stat('select fts from spots');
```

8.3: 66405972.737 ms
CVS HEAD: 25506.736 ms 2600x faster!
Fast approximated statistics

- Gevel extension — GiST/GIN indexes explorer (http://www.sai.msu.su/~megera/wiki/Gevel)
- **Fast** — uses only GIN index (no table access)
- **Approximated** — no table access, which contains visibility information, approx. for long posting lists
- For mostly **read-only** data error is small
Fast approximated statistics

- Top-5 most frequent words (463,873 docs)

```sql
=# SELECT * FROM gin_stat('gin_idx') as t(word text, ndoc int)
order by ndoc desc limit 5;

<table>
<thead>
<tr>
<th>word</th>
<th>ndoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>340858</td>
</tr>
<tr>
<td>figur</td>
<td>240366</td>
</tr>
<tr>
<td>use</td>
<td>148022</td>
</tr>
<tr>
<td>model</td>
<td>134442</td>
</tr>
<tr>
<td>result</td>
<td>129010</td>
</tr>
</tbody>
</table>

(5 rows)
Time: 520.714 ms
Fast approximated statistics

• `gin_stat()` vs `ts_stat()`

```sql
# select * into stat from ts_stat('select fts from papers') order by ndoc desc, nentry desc, word;

...wait.... 68704,182 ms

# SELECT a.word, b.ndoc as exact, a.estimation as estimation, round ((a.estimation-b.ndoc)*100.0/a.estimation,2)||'%' as error FROM (SELECT * FROM gin_stat('gin_x_idx') as t(word text, estimation int) order by estimation desc limit 5 ) as a, stat b WHERE a.word = b.word;

<table>
<thead>
<tr>
<th>word</th>
<th>exact</th>
<th>estimation</th>
<th>error</th>
</tr>
</thead>
<tbody>
<tr>
<td>page</td>
<td>340430</td>
<td>340858</td>
<td>0.13%</td>
</tr>
<tr>
<td>figur</td>
<td>240104</td>
<td>240366</td>
<td>0.11%</td>
</tr>
<tr>
<td>use</td>
<td>147132</td>
<td>148022</td>
<td>0.60%</td>
</tr>
<tr>
<td>model</td>
<td>133444</td>
<td>134442</td>
<td>0.74%</td>
</tr>
<tr>
<td>result</td>
<td>128977</td>
<td>129010</td>
<td>0.03%</td>
</tr>
</tbody>
</table>

(5 rows)
Time: 550.562 ms
```
GIN improvements

- GIN fast update (8.4)
- GIN partial match support (8.4)
- GIN multicolumn index (8.4)
- contrib/btree_gin (8.4) – provides GIN operator classes, that implement B-tree for all data types. Useful to use with GIN multicolumn feature:

```
CREATE index fts_idx ON papers USING gin(timestamp, fts_tsvector);
```
Talk roadmap

- Full-text search introduction
- Main topics
  - Phrase Search
  - Dictionaries API
- New features (already in 8.4)
- Future features
- Tips and Tricks
Future features

Red-Black tree experiment to replace binary tree in GIN – better defense against skewed data.
Future features

- Red-Black tree experiment to replace binary tree in GIN – better defense against skewed data – motivational example by Sergey Burladyan
  

```sql
create table a (i1 int, i2 int, i3 int, i4 int, i5 int, i6 int);
insert into a select n, n, n, n, n, n from generate_series(1, 100000) as n;
create index arr_gin on a using gin ( (array[i1, i2, i3, i4, i5, i6]) );

truncate a;
drop index arr_gin ;
create index arr_gin on a using gin ( (array[i1, i2, i3, i4, i5, i6]) );
insert into a select n, n, n, n, n, n from generate_series(1, 100000) as n;
```
Red-Black Tree

- **8.3.5** - binary tree
- **8.4beta1** - binary tree + limit
- **8.4beta1+Red-Black tree**

(self-balancing binary search tree, the longest path from any node to a leaf is no more than twice the shortest path)

<table>
<thead>
<tr>
<th>Version</th>
<th>Index (bulk)</th>
<th>Index+Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.5</td>
<td>123276.419</td>
<td>3415.676</td>
</tr>
<tr>
<td>8.4beta1</td>
<td>2686.435</td>
<td>2900.268</td>
</tr>
<tr>
<td>8.4beta1+rbtree</td>
<td>2075.634</td>
<td>2708.512</td>
</tr>
</tbody>
</table>
Red-Black Tree

```
select array_to_string(ARRAY(select '' || c || '.' || b from generate_series(1,50) b, ' ')::tsvector AS i INTO a
FROM generate_series(1,100000) c;
create index arr_gin on a using gin (i);

drop table a;
create table a ( i tsvector);
create index arr_gin on a using gin (i);
insert into a select array_to_string(ARRAY(select '' || c || '.' || b from generate_series(1,50) b, ' ')::tsvector AS i
FROM generate_series(1,100000) c;

8.3.5               8.4beta1       8.4beta1+rbtree

----------------------------------------------------------------------------------------------------
index (bulk):      inf (>1night)  228564.291   152569.763
index+insert:    410300.855   314332.507   251015.830
Epaper archive:                                  81714.308          86312.517
```

Oleg Bartunov, Teodor Sigaev
PGCon, Ottawa, May 21-22, 2009
Downloads (CVS HEAD)

- Phrase search
  - http://www.sigaev.ru/misc/phrase_search-0.7.gz

- Filter dictionary support
  - http://www.sigaev.ru/misc/filter_dict-0.2.gz
  - http://www.sigaev.ru/misc/unaccent-0.2.tar.gz

- Synonym dictionary with prefix search
  - http://www.sigaev.ru/misc/synonym_prefix.gz

- Red-Black tree
  - http://www.sigaev.ru/misc/rbtree-0.2.gz
Polygons

A @> B = TRUE

A && B = TRUE

http://www.sigaev.ru/misc/polygon-0.1.gz
Talk roadmap

- Full-text search introduction
- Main topics
  - Phrase Search
  - Dictionaries API
- New features (already in 8.4)
- Future features
- **Tips and Tricks**
Full-text search tips

- Aggregate for tsvector
- Stable to_tsquery
- Find documents with specific token type
- Getting words from tsvector
- Confuse with text search
CREATE AGGREGATE tsvector_sum(tsvector) (  
   SFUNC = tsvector_concat, 
   STYPE = tsvector, 
   INITCOND = "" 
);

=\# SELECT tsvector_sum( t.fts) FROM ( select ('1 2 ' || generate_series(3,10,1))::tsvector AS fts ) AS t;
   tsvector_sum

------------------------------
'1' '2' '3' '4' '5' '6' '7' '8' '9' '10'
Stable to_tsquery

Result of to_tsquery() can't be used as a cache key, since ts_query() does preserve an order, which isn't good for caching.

Little function helps:

```sql
CREATE OR REPLACE FUNCTION stable_ts_query(tsquery) RETURNS tsquery AS
$$
  SELECT ts_rewrite($1, 'dummy_word', 'dummy_word');
$$
LANGUAGE SQL RETURNS NULL ON NULL INPUT IMMUTABLE;
```

Note: Remember about text search configuration to have really good cache key!
How to find documents, which contain emails?

```sql
CREATE OR REPLACE FUNCTION document_token_types(text) RETURNS _text AS $$
SELECT ARRAY (
    SELECT
        DISTINCT alias
    FROM
        ts_token_type('default') AS tt,
        ts_parse('default', $1) AS tp
    WHERE
        tt.tokid = tp.tokid
);
$$ LANGUAGE SQL immutable;
```
Find documents with specific token type

=# SELECT document_token_types(title) FROM papers
   LIMIT 10;

   document_token_types

   {asciihword, asciihword, blank, hword_asciipart}
   {asciihword, blank}
   {asciihword, blank}
   {asciihword, blank}
   {asciihword, blank}
   {asciihword, blank}
   {asciihword, blank, float, host}
   {asciihword, blank}
   {asciihword, blank, hword_asciipart, int, numword, uint}
   {asciihword, blank}
   {asciihword, blank}
   (10 rows)

CREATE INDEX fts_types_idx ON papers USING gin( document_token_types (title) );
Find documents with specific token type

How to find documents, which contain emails?

SELECT comment FROM papers
WHERE document_token_types(title) && '{email}';

The list of available token types:

SELECT * FROM ts_token_type('default');
Creating the function `ts_stat`:

```sql
CREATE OR REPLACE FUNCTION ts_stat(tsvector, OUT word text,
OUT ndoc integer, OUT nentry integer)
RETURNS SETOF record AS $$
SELECT ts_stat('SELECT ' || quote_literal( $1::text ) || '::tsvector');
$$ LANGUAGE SQL RETURNS NULL ON NULL INPUT IMMUTABLE;
```

Example usage:

```sql
SELECT id, (ts_stat(fts)).* FROM apod WHERE id=1;
```

<table>
<thead>
<tr>
<th>id</th>
<th>word</th>
<th>ndoc</th>
<th>nentry</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>io</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>may</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>new</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>red</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>two</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Confuse with text search

One expected **true** here, but result is disappointing **false**

```sql
=# select to_tsquery('ob_1','inferences') @@
   to_tsvector('ob_1','inference');

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

Use `ts_debug()` to understand the problem

```plaintext
'inferences':
{french_ispell,french_stem} | french_stem | {inferent}

'inference':
{french_ispell,french_stem} | french_ispell | {inference}
```
Confuse with text search

- Use synonym dictionary as a first dictionary
  \{synonym,french_ispell,french_stem\}
  with rule 'inferences inference'
  - Don't forget to reindex!
- Use ts_rewrite()
  - Don't need to reindex
• Our work was supported by
  - Russian Foundation for Basic Research
  - EnterpriseDB
  - jfg://networks

THANKS !