Full-Text Search in PostgreSQL

Oleg Bartunov
Moscow University
PostgreSQL Global Development Group
FTS in Database

• Full-text search
  – Find documents, which satisfy query
  – return results in some order (opt.)

• Requirements to FTS
  – Full integration with PostgreSQL
    • transaction support
    • concurrency and recovery
    • online index
  – Linguistic support
  – Flexibility
  – Scalability
What is a Document?

- Arbitrary textual attribute
- Combination of textual attributes
- Should have unique id
- Could be fully virtual
- It's a textual result of any SQL command

<table>
<thead>
<tr>
<th>id</th>
<th>Title</th>
<th>Abstract</th>
<th>Keywords</th>
<th>Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Did</td>
<td>Aid</td>
<td>Author</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Text Search Operators

- Traditional FTS operators for textual attributes: ~, ~*, LIKE, ILIKE

Problems
- No linguistic support, no stop-words
- No ranking
- Slow, no index support. Documents should be scanned every time.

Solution
- Preprocess document in advance
- Add index support
FTA in PostgreSQL

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

- **tsvector** – storage for document, optimized for search
  - sorted array of lexemes
  - positional information
  - weights information
- **tsquery** – textual data type for query
  - Boolean operators - & | ! ()
- **FTS operator**
  tsvector @@ tsquery
• FTS is consists of
  – set of rules, which define how document and query should be transformed to their FTS representations – tsvector, tsquery.
  – set of functions to obtain tsvector, tsquery from textual data types
  – FTS operators and indexes
  – ranking functions, headline

• OpenFTS - openfts.sourceforge.net
  – constructs tsvector, tsquery by itself
  – use FTS operator and indexes
FTS features

• Full integration with PostgreSQL
• 27 built-in configurations for 10 languages
• Support of user-defined FTS configurations
• Pluggable dictionaries (ispell, snowball, thesaurus), parsers
• Multibyte support (UTF-8)
• Relevance ranking
• Two types of indexes – GiST and GiN with concurrency and recovery support
• Rich query language with query rewriting support
Complete FTS reference

- Data types
  - tsvector, tsquery
- FTS operators
  - @@, @@@
- Basic functions
  - to_tsvector, setweight, to_tsquery, plainto_tsquery, rewrite, tsearch
- Additional functions
  - rank_cd, rank, headline
- Additional operators
  - @>, <@
- Debug functions
  - lexize, ts_debug, parse, token_type, numnode, querytree, stat
**Full-Text Search in PostgreSQL**

Oleg Bartunov

```sql
TO_TSVECTOR(DOC)
```

### Algorithm

1. **PARSER**
   - `(token, token_type)`

2. **dicts(token_type)**
   - `i = 0`

3. **ask DICT[i]**
   - **YES**
   - **NO**

4. **IS STOP?**
   - **YES**
   - **NO**

5. **i = i + 1**
   - **YES**
   - **NO**

6. **i < N**
   - **YES**
   - **NO**
to_tsvector(doc)

PARSER

(token, token_type)

dicts(token_type)

i=0

ask DICT[i]

YES

i=i+1

YES

IS STOP ?

YES

NO

PGDay 2007, Prato, July 2007

Oleg Bartunov

Full-Text Search in PostgreSQL

## token_type('default')

<table>
<thead>
<tr>
<th>tokid</th>
<th>alias</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lword</td>
<td>Latin word</td>
</tr>
<tr>
<td>2</td>
<td>nlword</td>
<td>Non-latin word</td>
</tr>
<tr>
<td>3</td>
<td>word</td>
<td>Word</td>
</tr>
<tr>
<td>4</td>
<td>email</td>
<td>Email</td>
</tr>
<tr>
<td>5</td>
<td>url</td>
<td>URL</td>
</tr>
<tr>
<td>6</td>
<td>host</td>
<td>Host</td>
</tr>
<tr>
<td>7</td>
<td>sfloat</td>
<td>Scientific notation</td>
</tr>
<tr>
<td>8</td>
<td>version</td>
<td>VERSION</td>
</tr>
<tr>
<td>9</td>
<td>part_hword</td>
<td>Part of hyphenated word</td>
</tr>
<tr>
<td>10</td>
<td>nlpart_hword</td>
<td>Non-latin part of hyphenated word</td>
</tr>
<tr>
<td>11</td>
<td>lpart_hword</td>
<td>Latin part of hyphenated word</td>
</tr>
<tr>
<td>12</td>
<td>blank</td>
<td>Space symbols</td>
</tr>
<tr>
<td>13</td>
<td>tag</td>
<td>HTML Tag</td>
</tr>
<tr>
<td>14</td>
<td>protocol</td>
<td>Protocol head</td>
</tr>
<tr>
<td>15</td>
<td>hword</td>
<td>Hyphenated word</td>
</tr>
<tr>
<td>16</td>
<td>lhword</td>
<td>Latin hyphenated word</td>
</tr>
<tr>
<td>17</td>
<td>nlhword</td>
<td>Non-latin hyphenated word</td>
</tr>
<tr>
<td>18</td>
<td>uri</td>
<td>URI</td>
</tr>
<tr>
<td>19</td>
<td>file</td>
<td>File or path name</td>
</tr>
<tr>
<td>20</td>
<td>float</td>
<td>Decimal notation</td>
</tr>
<tr>
<td>21</td>
<td>int</td>
<td>Signed integer</td>
</tr>
<tr>
<td>22</td>
<td>uint</td>
<td>Unsigned integer</td>
</tr>
<tr>
<td>23</td>
<td>entity</td>
<td>HTML Entity</td>
</tr>
</tbody>
</table>

(23 rows)
to тsvector(doc)

<table>
<thead>
<tr>
<th>Token</th>
<th>Dictionaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>host</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>hword</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>int</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>lhword</td>
<td>public.pg_dict, public.en_ispell, pg_catalog.en_stem</td>
</tr>
<tr>
<td>lpart_hword</td>
<td>public.pg_dict, public.en_ispell, pg_catalog.en_stem</td>
</tr>
<tr>
<td>lword</td>
<td>public.pg_dict, public.en_ispell, pg_catalog.en_stem</td>
</tr>
<tr>
<td>nlhword</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>nlpart_hword</td>
<td>pg_catalog.simple</td>
</tr>
</tbody>
</table>

```
Lexize('en_stem','stars')
{star}
```
Dictionaries

• **Dictionary** – is a program, which accepts token and returns
  – an array of lexemes, if it is known and not a stop-word
  – void array, if it is a stop-word
  – NULL, if it's unknown
• API for developing specialized dictionaries
• Built-in dictionary-templates:
  – ispell (works with ispell, myspell, hunspell dicts)
  – snowball stemmer
  – synonym, thesaurus
  – simple
Dictionaries

- Dictionary for integers

CREATE TEXT SEARCH DICTIONARY intdict
LEXIZE 'dlexize_intdict' INIT 'dinit_intdict'
OPTION 'MAXLEN=6,REJECTLONG=false'
;

select lexize('intdict', 11234567890);
lexize
--------
{112345}
Dictionaries

• Dictionary for roman numerals

```sql
=# select lexize('roman', 'XIX');

lexize
-------

{19}

=# select to_tsvector('roman', 'postgresql was born in XIX-century') @@ plainto_tsquery('roman','19 century');

?column?
-------

t
```
Dictionaries

- 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\d)((abcd)?) GRB$1$2
to_tsvector(cfg, doc)

DOCUMENT

PARSER

(token, token_type)

dicts(token_type)

i = 0

YES

NO

i = i + 1

i < N

YES

NO

IS STOP ?

YES

NO

tsvector

fts Configuration
Supernovae & stars

Foreach leaf node

PARSER

(token, token_type)

dicts (token_type)

i=0

? DICT[i]

i=i+1

i < N

YES

NO

IS STOP ?

YES

NO

QUERYTREE

TSQUERY

to_tsquery

Supernovae & sn

&

star

\{(supernova,sn)\}

\{supernova,sn\} & star

\{(supernova | sn) & star\}
to_tquery, plainto_tquery

- to_tquery expects *preparsed* text
  - tokens with boolean operators between - & (AND), | (OR), ! (NOT) and parentheses
  - tokens can have weight labels
    'fat:ab & rats & ! (cats | mice)'

- plainto_tquery accepts *plain text*

- Tip: quote text in to_tquery
  ```sql
  select to_tquery(''' supernovae stars''':ab & !crab');
  ---------------
  'sn':AB & !'crab'
  ```
Indexes

- Indexes speedup full-text operators
  - FTS should works without indexes!
- Two types of indexes
  - GiST index
    - fast update
    - not well scaled with #words, #documents
    - supports `fillfactor` parameter
      ```sql
      create index gist_idx on apod using gist(fts)
      with (fillfactor=50);
      ```
  - GiN index
    - slow update
    - good scalability
- Both indexes support concurrency and recovery
GiST index - Signatures

• Each word hashed to the bit position – word signature
  w1 -> S1: 01000000
  w2 -> S2: 00010000
  w3 -> S3: 10000000

• Document signature is a superposition of word signatures
  S: 11010000  S1 || S2 || S3 – bit-wise OR

• Query signature – the same way

• Bloom filter
  Q1: 00000001 – exact not
  Q2: 01010000  - may be contained in the document, false drop

• Signature is a **lossy** representation of a document
  – + fixed length, compact, + fast bit operations
  – - lossy (false drops), - saturation with #words grows
### Demo collections – latin proverbs

<table>
<thead>
<tr>
<th>id</th>
<th>proverb</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ars longa, vita brevis</td>
</tr>
<tr>
<td>2</td>
<td>Ars vitae</td>
</tr>
<tr>
<td>3</td>
<td>Jus vitae ac necisș</td>
</tr>
<tr>
<td>4</td>
<td>Jus generis humani</td>
</tr>
<tr>
<td>5</td>
<td>Vita nostra brevis</td>
</tr>
</tbody>
</table>
GiST Index

- Demo collections – latin proverbs
  - each word represented as a fixed-length bitmap

<table>
<thead>
<tr>
<th>word</th>
<th>signature</th>
<th>Document is a bitwise-OR of all signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac</td>
<td>00000011</td>
<td></td>
</tr>
<tr>
<td>ars</td>
<td>11000000</td>
<td>ars vitae =&gt; 11000000</td>
</tr>
<tr>
<td>brevis</td>
<td>00001010</td>
<td>00011000</td>
</tr>
<tr>
<td>generis</td>
<td>01000100</td>
<td>==</td>
</tr>
<tr>
<td>humili</td>
<td>00110000</td>
<td>11011000</td>
</tr>
<tr>
<td>jus</td>
<td>00010001</td>
<td></td>
</tr>
<tr>
<td>longa</td>
<td>00100100</td>
<td></td>
</tr>
<tr>
<td>necis</td>
<td>01001000</td>
<td></td>
</tr>
<tr>
<td>nostra</td>
<td>10000001</td>
<td></td>
</tr>
<tr>
<td>vita</td>
<td>01000001</td>
<td></td>
</tr>
<tr>
<td>vitaeae</td>
<td>00011000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>proverb</th>
<th>signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ars longa, vita brevis</td>
<td>11101111</td>
</tr>
<tr>
<td>2</td>
<td>Ars vitae</td>
<td>11010000</td>
</tr>
<tr>
<td>3</td>
<td>Jus vitae ac necis</td>
<td>01011011</td>
</tr>
<tr>
<td>4</td>
<td>Jus generis humani</td>
<td>01110101</td>
</tr>
<tr>
<td>5</td>
<td>Vita nostra brevis</td>
<td>11001011</td>
</tr>
</tbody>
</table>

false hit
### Full-Text Search in PostgreSQL

#### GiST index - RD-Tree

**Query**

<table>
<thead>
<tr>
<th>Level</th>
<th>Valid</th>
<th>FTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>t</td>
<td>130 true bits, 1886 false bits</td>
</tr>
<tr>
<td>4</td>
<td>t</td>
<td>95 unique words</td>
</tr>
<tr>
<td>4</td>
<td>t</td>
<td>33 unique words</td>
</tr>
<tr>
<td>4</td>
<td>t</td>
<td>61 unique words</td>
</tr>
</tbody>
</table>

(417366 rows)

**Internal nodes**

<table>
<thead>
<tr>
<th>Level</th>
<th>Valid</th>
<th>FTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>t</td>
<td>852 true bits, 1164 false bits</td>
</tr>
<tr>
<td>3</td>
<td>t</td>
<td>861 true bits, 1155 false bits</td>
</tr>
<tr>
<td>3</td>
<td>t</td>
<td>858 true bits, 1158 false bits</td>
</tr>
<tr>
<td>3</td>
<td>t</td>
<td>773 true bits, 1243 false bits</td>
</tr>
</tbody>
</table>

(17496 rows)

**Leaf nodes**

<table>
<thead>
<tr>
<th>Query</th>
<th>Level</th>
<th>Valid</th>
<th>FTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>11011000</td>
<td>11011001</td>
<td>11011000</td>
<td>10010010</td>
</tr>
</tbody>
</table>

**Contrib module**: Gevel
### GIN Index

Demo collections – latin proverbs

<table>
<thead>
<tr>
<th>id</th>
<th>proverb</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ars longa, vita brevis</td>
</tr>
<tr>
<td>2</td>
<td>Ars vitae</td>
</tr>
<tr>
<td>3</td>
<td>Jus vitaeae ac neciș</td>
</tr>
<tr>
<td>4</td>
<td>Jus generis humanī</td>
</tr>
<tr>
<td>5</td>
<td>Vita nostra brevis</td>
</tr>
</tbody>
</table>
GIN Index

Demo collections – latin proverbs

Inverted Index

Entries tree

- word
  - ac
  - ars
  - brevis
  - generis
  - humani
  - jus
  - longa
  - neeis
  - nostra
  - vita
  - vitae

Postings tree

- posting
  - {3}
  - {1, 2}
  - {1, 5}
  - {4}
  - {3, 4}
  - {1}
  - {3}
  - {5}
  - {1, 5}
  - {2, 3}

• Fast search
• Slow update
GiN or GiST?

Direct comparison of performance on abstracts from e-print archives

Total number of abstracts - 405690.
Desktop PC, P4 2.4Ghz, 2Gb RAM, Linux 2.6.19.1, Slackware, PostgreSQL 8.2.4.

**postgresql.conf:**
shared_buffers = 256MB
work_mem = 8MB
maintenance_work_mem = 64MB
checkpoint_segments = 9
effective_cache_size = 256MB

```
arxiv=# select pg_relation_size('papers');
  pg_relation_size
------------------
  1054081024

arxiv=# select count(*) from wordstat;
  count
-----
  459841
```
query 'gamma & ray & burst & !supernovae' – 2764 hits

<table>
<thead>
<tr>
<th>Index</th>
<th>creation(ms)</th>
<th>size (b)</th>
<th>count(*)</th>
<th>rank query</th>
</tr>
</thead>
<tbody>
<tr>
<td>GiN</td>
<td>532310.368</td>
<td>305864704</td>
<td>38.739</td>
<td>130.488</td>
</tr>
<tr>
<td>GIST90</td>
<td>176267.543</td>
<td>145989632</td>
<td>111.891</td>
<td>188.992</td>
</tr>
<tr>
<td>GIST100</td>
<td>189321.561</td>
<td>130465792</td>
<td>120.730</td>
<td>215.153</td>
</tr>
<tr>
<td>GIST50</td>
<td>164669.614</td>
<td>279306240</td>
<td>122.101</td>
<td>200.963</td>
</tr>
</tbody>
</table>

Updating:
<table>
<thead>
<tr>
<th>index (nlev)</th>
<th>95</th>
<th>1035</th>
<th>10546</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIN</td>
<td>3343.881</td>
<td>36337.733</td>
<td>217577.424</td>
</tr>
<tr>
<td>GIST90  (4)</td>
<td>280.072</td>
<td>1835.485</td>
<td>29597.235</td>
</tr>
<tr>
<td>GIST100 (4)</td>
<td>232.674</td>
<td>2460.621</td>
<td>27852.507</td>
</tr>
<tr>
<td>GIST50 (5)</td>
<td>238.101</td>
<td>2952.362</td>
<td>33984.443</td>
</tr>
</tbody>
</table>

Conclusions:
- creation time - GiN takes 3x time to build than GiST
- size of index - GiN is 2-3 times bigger than GiST
- search time - GiN is 3 times faster than GiST
- update time - GiN is about 10 times slower than GiST
FTS new features

- FTS configuration - schema support
- FTS operator for textual data types
- Correct dump/restore (*)
- SQL interface to FTS configuration
- `psql` commands to display info about FTS objects
- changes of FTS objects are immediate
- `ispell` supports `ispell`, `myspell`, `hunspell` dicts
- improved `ts_debug`
- relative paths for dictionary files ($PGROOT/share)
Simple FTS

- FTS operator supports text data types
  - easy FTS without ranking
  - use other ordering

```sql
arxiv=# \d papers
Table "public.papers"
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>oai_id</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>datestamp</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>modification_date</td>
<td>date</td>
<td></td>
</tr>
</tbody>
</table>

arxiv=# create index title_idx on papers using gin(title);

arxiv=# select title from papers p where title @@ to_tsquery('supernovae & (Ia | Ib)')
order by modification_date desc limit 5;
```
FTS without tsvector column

- Use functional index (GiST or GiN)
  - no ranking, use other ordering

```sql
create index gin_text_idx on test using gin (coalesce(to_tsvector(title),'') || coalesce(to_tsvector(body),''));
```

```sql
apod=# select title from test where (coalesce(to_tsvector(title),'') || coalesce(to_tsvector(body),'')) @@ to_tsquery('supernovae') order by sdate desc limit 10;
```
A POD example

- curl -O http://www.sai.msu.su/~megera/postgres/fts/apod.dump.gz
- zcat apod.dump.gz | psql postgres
- psql postgres

```
postgres=# \d apod
        Table "public.apod"
Column | Type         | Modifiers
---------------+-------------+-----------
id       | integer     | not null
           title | text        |
           body | text        |
           sdate | date        |
           keywords | text      |

postgres=# show tsearch_conf_name;
  tsearch_conf_name
-------------------------
p_catalog.russian_utf8
```
### APOD example

```sql
postgres=# \dF+ pg_catalog.russian_utf8
Configuration "pg_catalog.russian_utf8"
Parser name: "pg_catalog.default"
Locale: 'ru_RU.UTF-8' (default)

<table>
<thead>
<tr>
<th>Token</th>
<th>Dictionaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>file</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>float</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>host</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>hword</td>
<td>pg_catalog.ru_stem_utf8</td>
</tr>
<tr>
<td>int</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>lhword</td>
<td>pg_catalog.en_stem</td>
</tr>
<tr>
<td>lpart_hword</td>
<td>pg_catalog.en_stem</td>
</tr>
<tr>
<td>lword</td>
<td>pg_catalog.en_stem</td>
</tr>
<tr>
<td>nlhword</td>
<td>pg_catalog.ru_stem_utf8</td>
</tr>
<tr>
<td>nlpart_hword</td>
<td>pg_catalog.ru_stem_utf8</td>
</tr>
<tr>
<td>nlword</td>
<td>pg_catalog.ru_stem_utf8</td>
</tr>
<tr>
<td>part_hword</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>sfloat</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>uint</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>uri</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>url</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>version</td>
<td>pg_catalog.simple</td>
</tr>
<tr>
<td>word</td>
<td>pg_catalog.ru_stem_utf8</td>
</tr>
</tbody>
</table>
```
Full-Text Search in PostgreSQL

Oleg Bartunov

postgres=# alter table apod add column fts tsvector;
postgres=# update apod  set fts=
setweight( coalesce( to_tsvector(title),""),'B') ||
setweight( coalesce( to_tsvector(keywords),""),'A') ||
setweight( coalesce( to_tsvector(body),""),'D');

postgres=# create index apod_fts_idx on apod using gin(fts);
postgres=# vacuum analyze apod;

postgres=# select title from apod where fts  @@ plainto_tsquery('supernovae stars') limit 5;

-------------------------------------------
title
-------------------------------------------
Runaway Star
Exploring The Universe With IUE 1978-1996
Tycho Brahe Measures the Sky
Unusual Spiral Galaxy M66
COMPTEL Explores The Radioactive Sky
### APOD example: Search

```
postgres=# select title, rank_cd(fts, q) from apod, to_tsquery('supernovae & x-ray') q
where fts @@ q order by rank_cd desc limit 5;
```

<table>
<thead>
<tr>
<th>title</th>
<th>rank_cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supernova Remnant E0102-72 from Radio to X-Ray</td>
<td>1.59087</td>
</tr>
<tr>
<td>An X-ray Hot Supernova in M81</td>
<td>1.47733</td>
</tr>
<tr>
<td>X-ray Hot Supernova Remnant in the SMC</td>
<td>1.34823</td>
</tr>
<tr>
<td>Tycho's Supernova Remnant in X-ray</td>
<td>1.14318</td>
</tr>
<tr>
<td>Supernova Remnant and Neutron Star</td>
<td>1.08116</td>
</tr>
</tbody>
</table>

(5 rows)

Time: 1.965 ms

rank_cd uses only local information!

\[0 < \frac{\text{rank}}{\text{rank}+1} < 1\]

```
rank_cd(\{'0.1, 0.2, 0.4, 1.0\'},fts, q)
```

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
</table>

postgres=# select headline(body,q,'StartSel=<,StopSel=>,MaxWords=10,MinWords=5'), rank_cd(fts, q) from apod, to_tsquery('supernovae & x-ray') q where fts @@ q order by rank_cd desc limit 5;

<table>
<thead>
<tr>
<th>headline</th>
<th>rank_cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;supernova&gt; remnant E0102-72, however, is giving astronomers a clue</td>
<td>1.59087</td>
</tr>
<tr>
<td>&lt;supernova&gt; explosion. The picture was taken in &lt;X&gt;-&lt;rays&gt;</td>
<td>1.47733</td>
</tr>
<tr>
<td>&lt;X&gt;-&lt;ray&gt; glow is produced by multi-million degree</td>
<td>1.34823</td>
</tr>
<tr>
<td>&lt;X&gt;-&lt;rays&gt; emitted by this shockwave made by a telescope</td>
<td>1.14318</td>
</tr>
<tr>
<td>&lt;X&gt;-&lt;ray&gt; glow. Pictured is the &lt;supernova&gt;</td>
<td>1.08116</td>
</tr>
</tbody>
</table>

(5 rows)

Time: 39.298 ms

Slow, use subselects! See tips
### APOD example

- Different searches with one full-text index
  - title search

```sql
=# select title, rank_cd(fts, q) from apod,
    to_tsquery('supernovae:b & x-ray') q
where fts @@ q order by rank_cd desc limit 5;
```

<table>
<thead>
<tr>
<th>title</th>
<th>rank_cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supernova Remnant E0102-72 from Radio to X-Ray</td>
<td>1.59087</td>
</tr>
<tr>
<td>An X-ray Hot Supernova in M81</td>
<td>1.47733</td>
</tr>
<tr>
<td>X-ray Hot Supernova Remnant in the SMC</td>
<td>1.34823</td>
</tr>
<tr>
<td>Tycho's Supernova Remnant in X-ray</td>
<td>1.14318</td>
</tr>
<tr>
<td>Supernova Remnant and Neutron Star</td>
<td>1.08116</td>
</tr>
</tbody>
</table>

(5 rows)

`to_tsquery('supernovae:ab')` - title and keywords search
FTS tips

- headline() function is slow – use \texttt{subselect}

\begin{verbatim}
select id, headline(body, q), rank(fts, q) as rank
from apod, to_tsquery('stars') q
where fts @@ q order by rank desc limit 10;
\end{verbatim}

Time: 723.634 ms

\begin{verbatim}
select id, headline(body, q), rank from (select id, body, q, rank(fts, q) as rank
from apod, to_tsquery('stars') q
where fts @@ q order by rank desc limit 10
) as foo;
\end{verbatim}

Time: 21.846 ms

\begin{verbatim}
=\#select count(*)from apod where fts @@ to_tsquery('stars');
\end{verbatim}

count

\begin{verbatim}
-\-
790
\end{verbatim}
• Fuzzy search with contrib/pg_trgm - trigram statistics

```
=# select show_trgm('supyrnova');
show_trgm
--------------------------
{" s"," su",nov,ova,pyr,rno,sup,upy," va ",yrn}
```

```
=# select * into apod_words from stat('select fts from apod') order by ndoc desc, nentry desc, word;

=# \d apod_words
Table "public.apod_words"
Column | Type    | Modifiers
--------+---------+-----------
word   | text    |           |
ndoc   | integer |           |
nentry | integer |           |

=# create index trgm_idx on apod_words using gist(word gist_trgm_ops);
=# select word, similarity(word, 'supyrnova') AS sml
from apod_words where word % 'supyrnova' order by sml desc, word;

word    |   sml
-----------+----------
supernova | 0.538462
```

FTS tips

collect statistics
To be or not to be ...

Two FTS configurations: with and without stop-words
To be or not to be ...

```
hamlet=# \dFd+ en_stem

List of fulltext dictionaries

<table>
<thead>
<tr>
<th>Schema</th>
<th>Name</th>
<th>Init method</th>
<th>Lexize method</th>
<th>Init options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pg_catalog</td>
<td>en_stem</td>
<td>dsnb_en_init</td>
<td>dsnb_lexize</td>
<td>dicts_data/english.stop</td>
<td>English stemmer. Snowball.</td>
</tr>
</tbody>
</table>

CREATE TEXT SEARCH DICTIONARY en_stem_nostop OPTION NULL LIKE en_stem;
CREATE TEXT SEARCH CONFIGURATION hamlet LIKE english WITH MAP;
ALTER TEXT SEARCH CONFIGURATION hamlet
    ALTER MAPPING lhword,lpart_hword,lword WITH en_stem_nostop;

update text set fts=coalesce(to_tsvector('hamlet',txt),'');
```

```sql
hamlet=# select headline('hamlet',txt,q,'StartSel=<,StopSel=>') from text,
    plainto_tsquery('hamlet','to be or not to be') q where fts @@ q;
```

-------------------------------------------------------------
| Ham. <To> <be>, <or> <not> <to> <be>, that is the Question: |
FTS tips – Query rewriting

• Online rewriting of query
  – Query expansion
    • synonyms (new york => Gotham, Big Apple, NYC ...)
  – Query narrowing (submarine Kursk went down)
    • Kursk => submarine Kursk
• Similar to synonym (thesaurus) dictionary, but doesn't require reindexing
FTS tips – Query rewriting

```
create table aliases( t tsquery primary key, s tsquery);
insert into aliases values(to_tsquery('supernovae'),
to_tsquery('supernovae|sn'));
apod=# select rewrite(to_tsquery('supernovae'),
'select * from aliases');
   rewrite
-------------------
   'supernova' | 'sn'
```
FTS tips – Query rewriting

```
apod=# select title, rank_cd(fts,q,1) as rank
        from apod, to_tsquery('supernovae') q
       where fts @@ q  order by rank desc limit 10;
```

<table>
<thead>
<tr>
<th>title</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Mysterious Rings of Supernova 1987A</td>
<td>0.669633</td>
</tr>
<tr>
<td>Tycho's Supernova Remnant in X-ray</td>
<td>0.598556</td>
</tr>
<tr>
<td>Tycho's Supernova Remnant in X-ray</td>
<td>0.598556</td>
</tr>
<tr>
<td>Vela Supernova Remnant in Optical</td>
<td>0.591655</td>
</tr>
<tr>
<td>Vela Supernova Remnant in Optical</td>
<td>0.591655</td>
</tr>
<tr>
<td>Galactic Supernova Remnant IC 443</td>
<td>0.590201</td>
</tr>
<tr>
<td>Vela Supernova Remnant in X-ray</td>
<td>0.589028</td>
</tr>
<tr>
<td>Supernova Remnant: Cooking Elements In The LMC</td>
<td>0.585033</td>
</tr>
<tr>
<td>Cas A Supernova Remnant in X-Rays</td>
<td>0.583787</td>
</tr>
<tr>
<td>Supernova Remnant N132D in X-Rays</td>
<td>0.579241</td>
</tr>
</tbody>
</table>
### FTS tips – Query rewriting

```sql
apod=# select id, title, rank_cd(fts,q,1) as rank
from apod, rewrite(to_tsquery('supernovae'), 'select * from aliases') q
where fts @@ q  order by rank desc limit 10;
```

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1162701</td>
<td>The Mysterious Rings of Supernova 1987A</td>
<td>0.90054</td>
</tr>
<tr>
<td>1162717</td>
<td>New Shocks For Supernova 1987A</td>
<td>0.738432</td>
</tr>
<tr>
<td>1163673</td>
<td>Echos of Supernova 1987A</td>
<td>0.658021</td>
</tr>
<tr>
<td>1163593</td>
<td>Shocked by Supernova 1987a</td>
<td>0.621575</td>
</tr>
<tr>
<td>1163395</td>
<td>Moving Echoes Around SN 1987A</td>
<td>0.614411</td>
</tr>
<tr>
<td>1161721</td>
<td>Tycho's Supernova Remnant in X-ray</td>
<td>0.598556</td>
</tr>
<tr>
<td>1163201</td>
<td>Tycho's Supernova Remnant in X-ray</td>
<td>0.598556</td>
</tr>
<tr>
<td>1163133</td>
<td>A Supernova Star-Field</td>
<td>0.595041</td>
</tr>
<tr>
<td>1163611</td>
<td>Vela Supernova Remnant in Optical</td>
<td>0.591655</td>
</tr>
<tr>
<td>1161686</td>
<td>Vela Supernova Remnant in Optical</td>
<td>0.591655</td>
</tr>
</tbody>
</table>

```sql
apod=# select title, rank_cd(fts,q,1) as rank from apod,
    to_tsquery('supernovae') q where fts @@ q and id=1162717;
```

<table>
<thead>
<tr>
<th>title</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Shocks For Supernova 1987A</td>
<td>0.533312</td>
</tr>
</tbody>
</table>
```
FTS tips – Partition your data

• Problem:
  – FTS on very big collection of documents

• Solution:
  – Partition data
    • Table inheritance + Constraint Exclusion – current and one or more archive tables
    • GiST index for current table
    • GiN index for archive table(s)
FTS tips – Partition your data

• Create parent class
  CREATE TABLE papers_class (  
    id integer,  
    ............  
    creation_date date,  
    fts tsvector  
  );

• Create current and archive tables
  CREATE TABLE papers (  
    CHECK (creation_date >= '2007-01-01'::date)  
  ) INHERITS ( papers_class );

  CREATE TABLE paper_archive (  
    CHECK (creation_date < '2007-01-01'::date)  
  ) INHERITS ( papers_class );
FTS tips – Partition your data

• Create GiST index for *current* table
  CREATE INDEX gist_idx ON paper USING gist(fts);
  – Not so big
  – Frequently updated
  – GiST is good for updates and fast enough

• Create GIN index for *archive* table
  CREATE INDEX gin_idx ON paper_archive USING gin(fts);
  – May be very big
  – Static
  – GIN is very well scaled

• Don't forget to enable constraint exclusion
  SET constraint_exclusion=on;
FTS tips – Partition your data

• All queries will not use tables which doesn't match CHECK constraint on creation_date

```
arxiv=# explain select title from papers_class where
    fts @@ to_tsquery('stars') and creation_date > '05-01-2007'::date;
```

```
QUERY PLAN

Result (cost=0.00..63.47 rows=3 width=73)
  ->  Append (cost=0.00..63.47 rows=3 width=73)
     ->  Seq Scan on papers_class (cost=0.00..14.95 rows=1 width=32)
         Filter: ((fts @@ 'star'::tsquery) AND (creation_date > '2007-05-01'::date))
     ->  Bitmap Heap Scan on papers_papers_class (cost=40.53..48.52 rows=2 width=73)
         Recheck Cond: (creation_date > '2007-05-01'::date)
         Filter: (fts @@ 'star'::tsquery)
         ->  BitmapAnd (cost=40.53..40.53 rows=2 width=0)
             ->  Bitmap Index Scan on gist_idx (cost=0.00..4.42 rows=18 width=0)
                 Index Cond: (fts @@ 'star'::tsquery)
             ->  Bitmap Index Scan on creation_date_papers_idx (cost=0.00..35.86 rows=2081 width=0)
                 Index Cond: (creation_date > '2007-05-01'::date)

Big table paper_archive was excluded!
select dblink_connect('pgweb','dbname=pgweb hostaddr='XXX.XXX.XXX.XXX');

select * from dblink('pgweb',
'select tid, title, rank_cd(fts_index, q) as rank from pgweb,
  to_tsquery('table') q
where q @@ fts_index and tid >= 6000
order by rank desc limit 10' )
as t1 (tid integer, title text, rank real)
union all
select tid, title, rank_cd(fts_index, q) as rank from pgweb,
  to_tsquery('table') q
where q @@ fts_index and tid < 6000 and tid > 0
order by rank desc limit 10
) as foo
order by rank desc limit 10;
References

• Documentation
  – http://www.sai.msu.su/~megera/wiki/tsearch2 - tsearch2 Wiki
  – http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2 - tsearch2 home page
  – http://www.sai.msu.su/~megera/postgres/talks/ - presentations about PostgreSQL

• Data
  – http://www.sai.msu.su/~megera/postgres/fts/apod.dump.gz

• Acknowledgements
  – Russian Foundation for Basic Research
  – -hackers, EnterprizeDB PostgreSQL Development Fund, Mannheim University, jfg:networks, Georgia Public Library Service, Rambler Internet Holding
Questions ?
FTS tips

- GIN_FUZZY_SEARCH_LIMIT - maximum number of returned rows
  - GIN_FUZZY_SEARCH_LIMIT=0, disabled on default