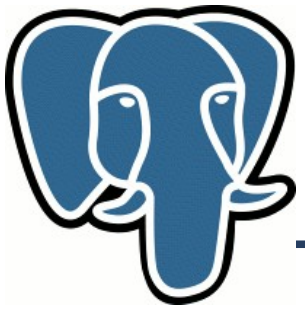
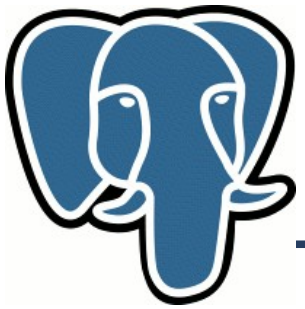


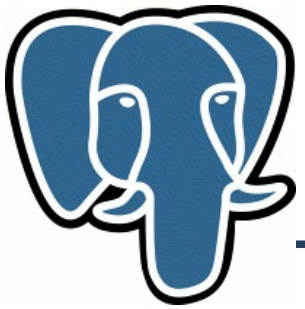
What is a FTS ?

- Find documents, which satisfy query
- optionally return them in some order
- Most common case:
  - Find documents containing **all** query terms
  - return them in order of their similarity to the query



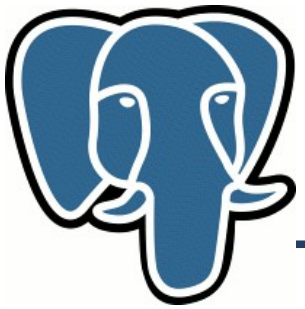
What's a document ?

- any text attribute
- combination of text attributes from one or many tables.
- Document must be identified by some unique key



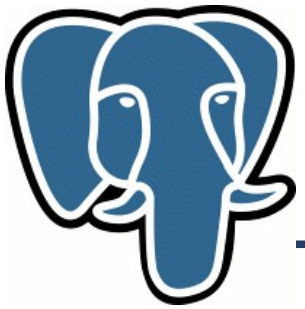
~,~*, LIKE, ILIKE for FTS

- Text search operators existed for years
  - No linguistics
  - No ordering (ranking)
  - Tends to be slow (no index support)



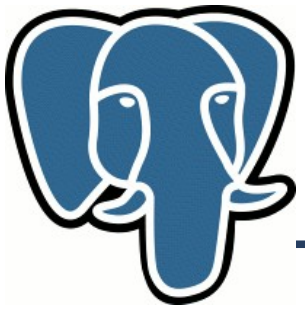
Improve FTS

- The idea is simple - preprocess document at index time to save time at search stage.
  - document parsing - (token, token type)
  - lingustic - normalize lexeme (depending on token type)
  - storage (sorted list of lexemes with positional information)



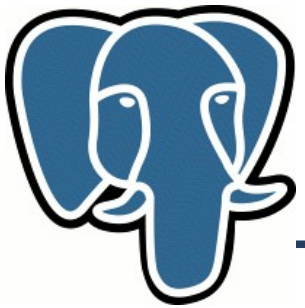
Tsearch2 comes

- Tsearch2 - is the full text engine for PostgreSQL. Main features (new in 8.2 are **bolded**):
  - Supports multiple table driven configurations
  - flexible and rich linguistic support (dictionaries, stop words), **thesaurus**
  - **UTF-8 support**
  - full integration with PostgreSQL



Tsearch2

- Sophisticated ranking functions with support of proximity and structure information (rank, **rank_cd**)
- Index support (GiST and **Gin**) with concurrency and recovery support
- Rich query language with **query rewriting** support
- It is mature (5 years of development)



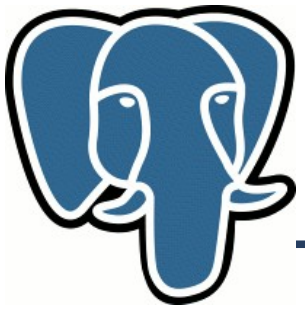
Tsearch2

We introduced two new data types and operator for FTS

tsvector @@ **tsquery**
tsquery @@ tsvector

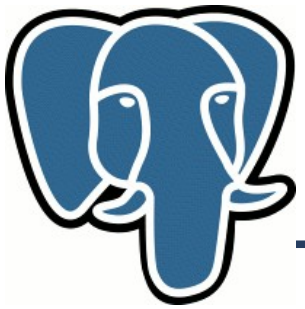
@@ operator returns TRUE if tsvector **contains** tsquery.

```
'fat & cat'::tsquery @@
'a fat cat sat on the mat'::tsvector;
```



Tsvector

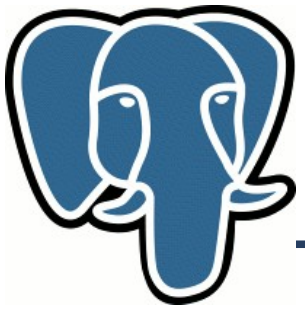
- data type, which represents document, optimized for FTS
- It's a sorted list of lexemes - search is faster than standard ~,LIKE operators.
- Lexeme, could have positional information with optional labels (4 groups)
- select 'a:1 fat:2 cat:3A'::tsvector
- tsvector || tsvector



Tsquery

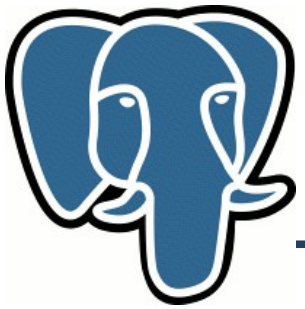
data type for textual queries with support of boolean operators

- Tsquery consists of lexemes (optionally labelled by letter[s]) with boolean operator between ('fat & cat'::tsquery)
- Concatenation
  - tsquery && tsquery
  - tsquery || tsquery



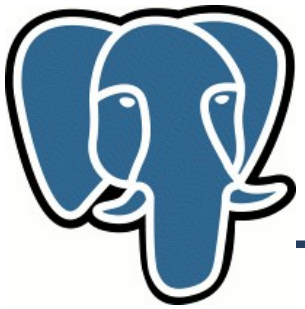
Limits

- Length of lexeme < 2K
- Length of tsvector (lexemes + positions) < 1Mb
- The number of lexemes < 4^32
- 0< Positional information < 16383
- No more than 256 positions per lexeme
- The number of nodes (lexemes + operations) in tsquery < 32768



Some statistics

- PostgreSQL 8.1 documentation
  - total 335420 lexemes
  - 10441 unique lexemes
  - 'postgresql' mentioned 6127 times in 655 documents
- PostgreSQL mailing list archive:
  - total 57,491,343 lexemes in 461020 msgs
  - 910989 uniquelexemes



Tsearch2 configurations
Four tables control FTS

- We want to control document-tsvector convertation
- We want to do that in various ways

That means, we want to define how to parse document, what lexemes to index and how to process them.

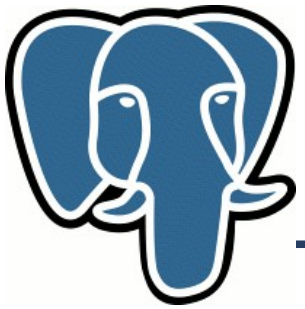
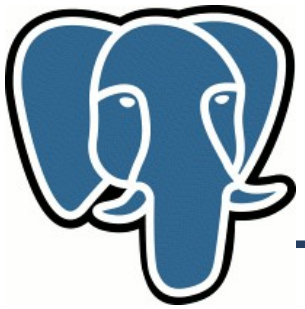


Table driven configuration

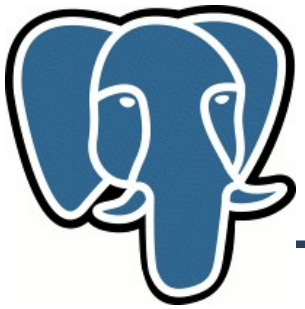
- pg_ts_cfg - configurations
- pg_ts_dict - dictionaries
- pg_ts_parser - document parsers
- pg_ts_cfgmap - map configurations, lexems and dictionaries



pg_ts_cfg

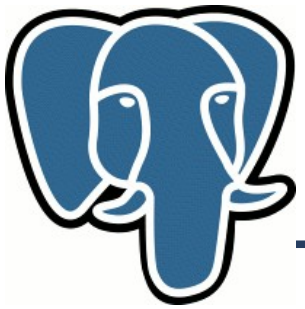
- Each configuration has unique name (ts_name), parser (prs_name), and locale name
- Locale is used to identify default configuration.

```
=# select * from pg_ts_cfg;
     ts_name      | prs_name |    locale
------------------+----------+--------------
 default          | default  | C
 default_russian  | default  | ru_RU.KOI8-R
 utf8_russian     | default  | ru_RU.UTF-8
 simple           | default  |
```



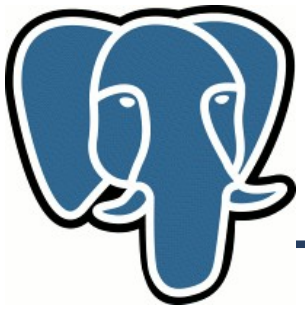
Dictionary

- Dictionary is a **program**, which accepts lexeme(s) on input and returns:
  - array of lexeme(s) if input lexeme is known to the dictionary
  - void array - dictionary knows lexeme, but it's stop word.
  - NULL - dictionary doesn't recognized input lexeme



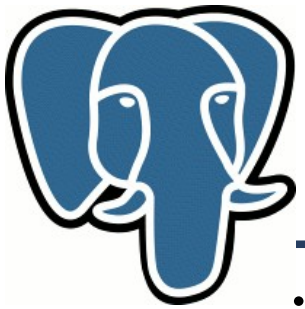
Normalization

- Linguistic normalization – ispell, stemmer
- Special
  - http://www.pgsql.ru/db/mw/index.html
  - http://www.pgsql.ru/db/mw/
  - http://www.pgsql.ru/db/../db/mw
  - red,green,blue, magenta – FF0000, 00FF00, 0000FF, FF00FF
  - **3.14**1592653589, **2.71**828182846



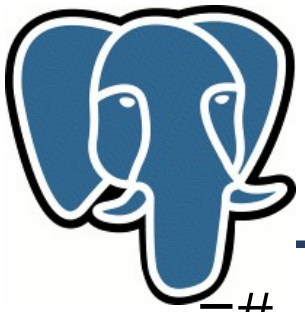
pg_ts_dict

- pg_ts_dict is a dictionaries registry
- Tsearch2 provides templates for several dictionaries to simplify registering of new dictionaries



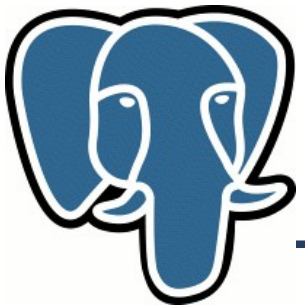
Dictionaries templates

- simple - returns lowercased lexeme (recognize everything)
- ispell - returns normalized lexeme(s) - morphology, compound words support
- snowball stemmer - returns lexeme stem (recognize everything)
- synonym - simple lexeme-to-lexeme replacement
- thesaurus - phrase-to-phrase replacement
-



Parser

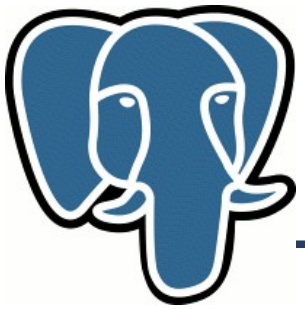
```
=# select * from token_type();
 tokid |    alias     |                descr
-------+--------------+---------------------
     1 | lword        | Latin word
     2 | nlword       | Non-latin word
     3 | word         | Word
     4 | email        | Email
     5 | url          | URL
     6 | host         | Host
               .........
    23 | entity       | HTML Entity
```



Parser

- parse([parser_name], text)
- set_curprs(parser_name) – default parser

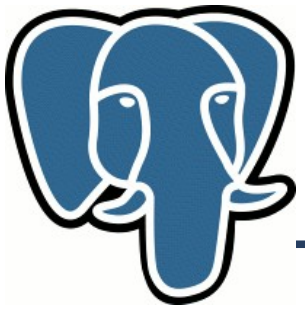
```
=# select * from parse('<b>Fat</b> cat');
 tokid | token
-------+-------
    13 | <b>
     1 | Fat
    13 | </b>
    12 |
     1 | cat
```



pg_ts_cfgmap

- tsname – configuration name
- lexeme type – {dict1, dict2,...,dictN}

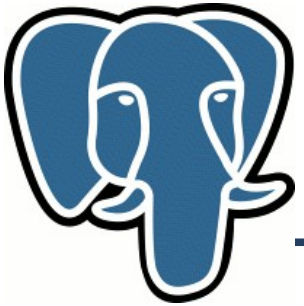
```
    ts_name      |  tok_alias    |    dict_name
-----------------+---------------+-----------------
.....................................................
default_russian | lword          | {en_ispell,en_stem}
.....................................................
default_russian | nlword         | {ru_ispell,ru_stem_koi8}
```



pg_ts_cfgmap

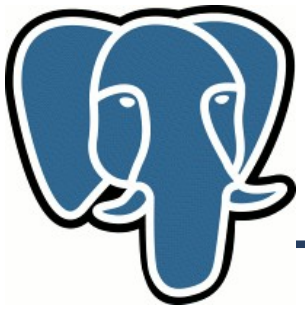
- Lexeme will not be indexed if:
- Lexeme's type is not in pg_ts_cfgmap
- or
- Dictionary stack is NULL

```
update pg_ts_cfgmap set dict_name=NULL where
ts_name='default_russian' and tok_alias='uri';
```



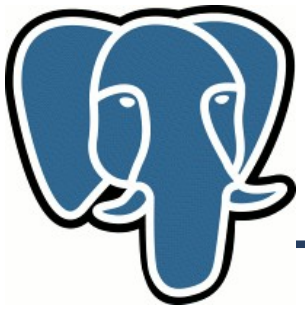
Tsquery

- to_tsquery ([ts_name], text)
- plainto_tsquery([ts_name],text)



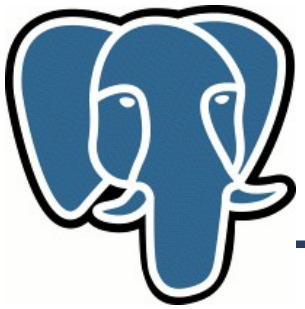
Tsquery – restricted search

- It's possible to use labels, stored in tsvector, to limit search region.
- Flexibility - Several searches using one tsvector
  - 'supernovae & stars'::tsquery - search everywhere
  - 'supernovae:a & stars'::tsquery - search only titles
  - 'supernovae:ab & stars'::tsquery - search titles and abstracts



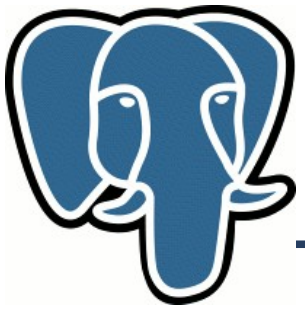
query rewriting

- Query rewriting is a set of functions and operators for tsquery type. Control search at query time without reindexing
  - Expand search using synonyms: new york, big apple, nyc,gotham
  - help search popular topic (online!):
  - submarine kursk went down August 12, 2000 year: 'kursk' rewritten to 'submarine kursk'



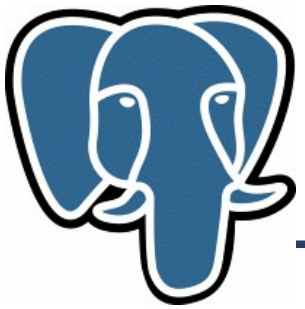
Query rewriting

- rewrite (tsquery, tsquery, tsquery)
- rewrite (ARRAY[tsquery,tsquery,tsquery])
- rewrite (tsquery, text)
  - rewrite (tsquery, 'select tsquery,tsquery from test'::text) - table driven
  - tsquery @ (~) tsquery operators, index support – using gist (keyword gist_tp_tsquery_ops)



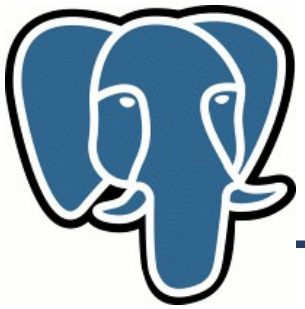
Getting results - ranking

- Ranking attempts to measure how documents are relevant to particular query.
- rank, rank_cd – different algorithms
- rank([{weights}], tsvector, tsquery, norm.)
  - weights, proximity
  - doc. length normalization
  - only local information is used, no way to have **true** 0-1 rank. Cheat: r/r+1



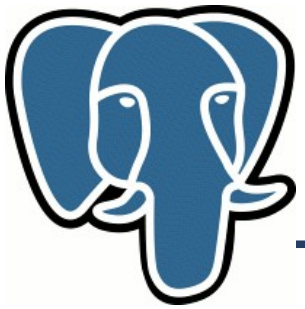
Getting results - headline

- Headline is a fragment of document with query terms.
- headline([ts_name], document, tsquery, options)
- Headline is slow (read document from disk), use subselect !



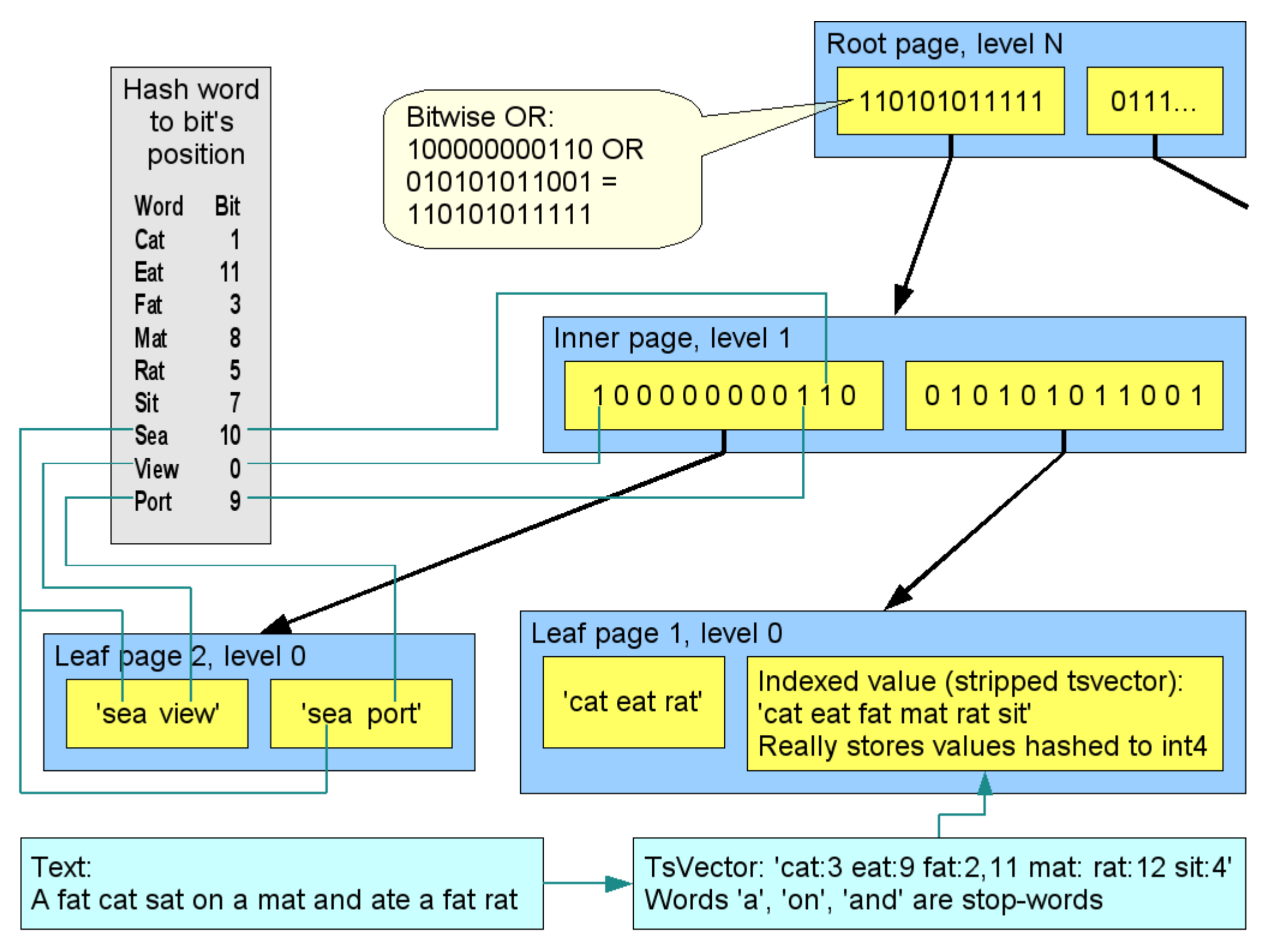
Indexes

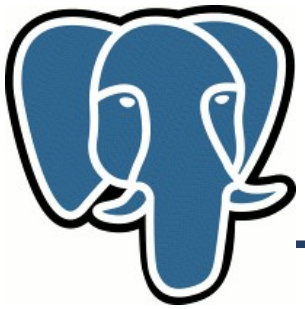
- Tsearch2 provides indexed AM for tsvector (indexes are not mandatory for FTS !)
- Signature tree – GiST
  - =# create index fts_idx on apod using gist(fts);
- Inverted index – Gin
  - =# create index fts_idx on apod using gin(fts);



Indexes - GiST

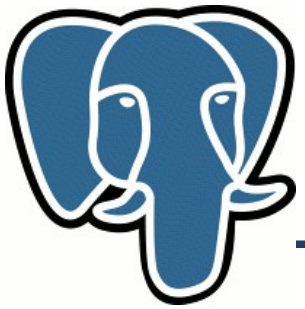
- Document represented as a bit string with '1' in positions to which words are hashed
- These bit strings are stored in RD-tree, where parent is 'OR'-ed bit-strings of all children
- This index is lossy, so we need to check results, it could be very expensive


Root page, level N
110101011111
0111...
Bitwise OR:
100000000110 OR
010101011001 =
110101011111
Hash word to bit's position
Word Bit
Cat 1
Eat 11
Fat 3
Mat 8
Rat 5
Sit 7
Sea 10
View 0
Port 9
Inner page, level 1
1 0 0 0 0 0 0 0 0 1 1 0
0 1 0 1 0 1 0 1 1 0 0 1
Leaf page 2, level 0
'sea view'
'sea port'
Leaf page 1, level 0
'cat eat rat'
Indexed value (stripped tsvector):
'cat eat fat mat rat sit'
Really stores values hashed to int4
Text:
A fat cat sat on a mat and ate a fat rat
TsVector: 'cat:3 eat:9 fat:2,11 mat: rat:12 sit:4'
Words 'a', 'on', 'and' are stop-words



Indexes – GiST

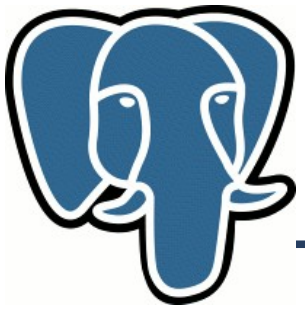
- good for online indexing
- support multicolumn indices
- not well scaled with the number of distinct words and the number of documents



Indexes - GIN

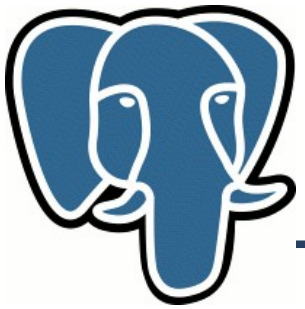
An inverted index is an index structure storing a set of (key, posting list) pairs, where 'posting list' is a set of document id in which the key occurs.

- weak dependence on the size of vocabulary, good scalability
- fast bulk indexing
- slow update



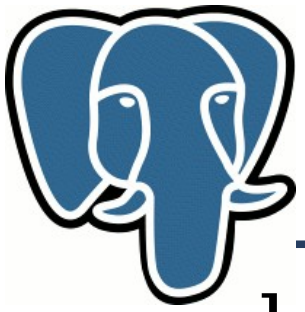
Indexes - Usage

- GiST index for online documents
- GIN – for archives
- Cron jobs for archiving



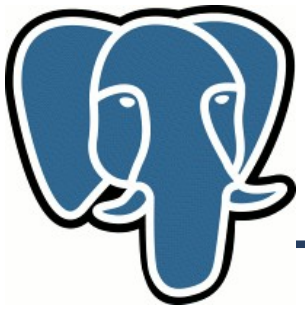
Acknowledgements

- ABC Startsiden - compound words support
- PostGIS community - GiST Concurrency and Recovery
- jfg://networks - GIN
- University of Mannheim - UTF-8
- Georgia Public Library Service and LibLime, Inc. - Thesaurus support
- Russian Foundation for Basic Research



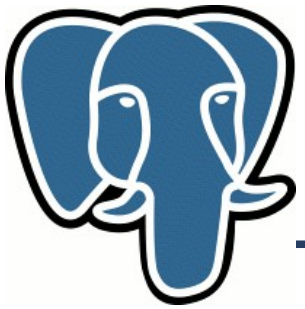
Todo - Tsearch2

- phrase search, exact search, wildcard search
- configurable number of weight groups in tsvector
- parser and dictionaries could define lexeme's weight
- built-in support pg_trgm (it can be used with GIN)
- configurable length of signature in GiST index



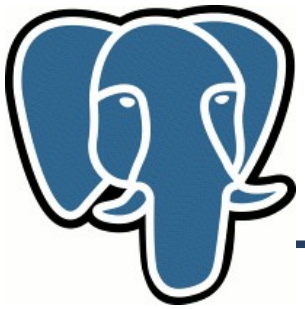
Todo – GiST

- Better interaction of GiST with optimizer and planner
- cost functions for several popular extensions (intarray, ltree, tsearch2,,,)
- Extend GiST interface to support SP-GIST



Todo – Gin

- Increase the number of strategies. Currently – only one (full match)
  - entries B-tree: <,<=,>,>=,prefix
- Extend Gin to support new data types.
  - replace entries B-tree by GiST similar tree (requres support of unique values in GiST).
  - This further increase the number of possible strategies.
- Optimize insert operations (background index insertion)



Todo – Gin

- Extend pgsql's interface to use GIN for ranking:
  - store position information in GIN
  - propagate ranking from index to final sort of tuples
- Better interaction of GIN with optimizer and planner, developing cost functions (tsearch2, built-in support for array,,)