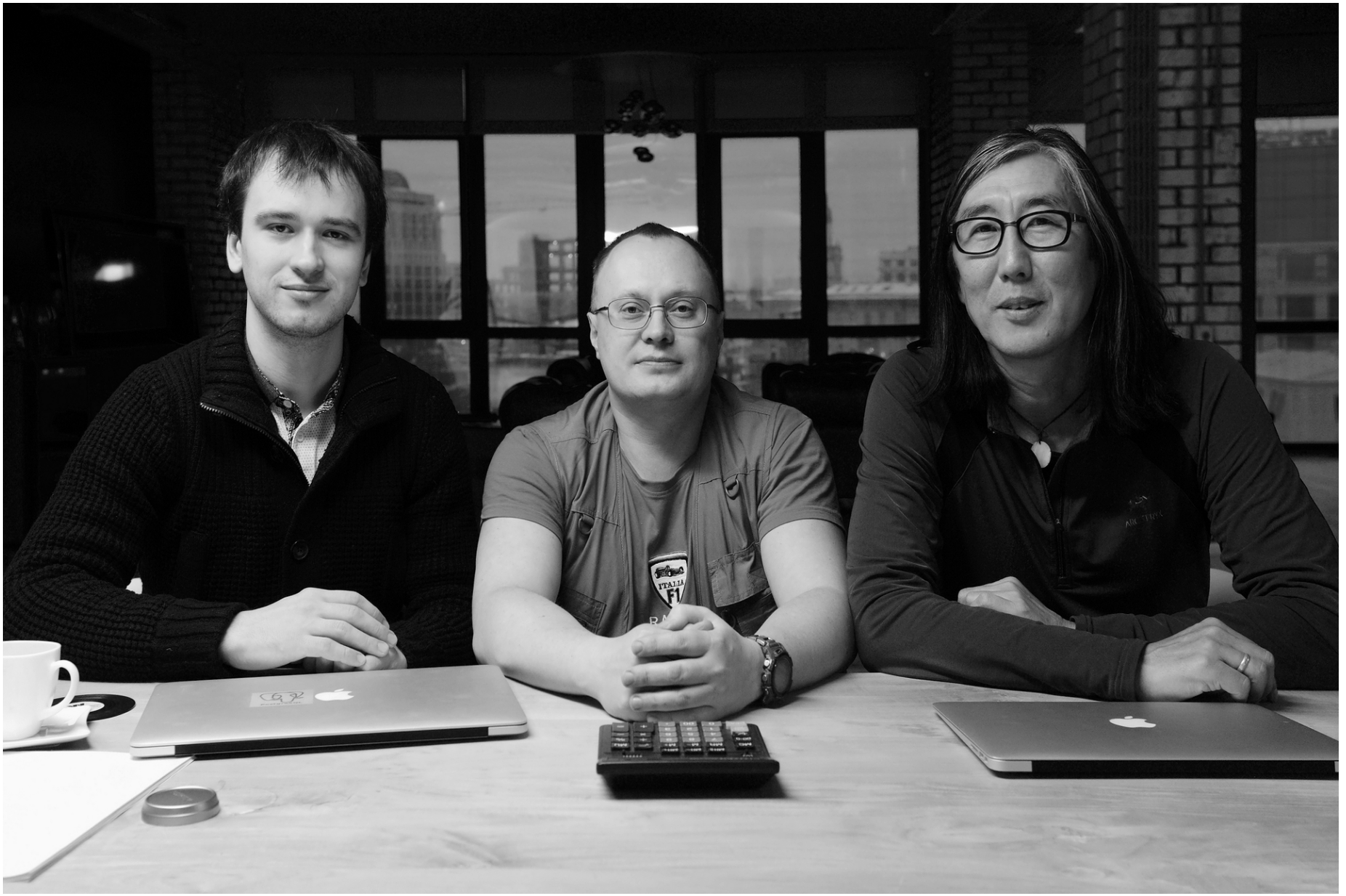
Пишем расширение для PostgreSQL на примере полнотекстового поиска

Александр Коротков
Федор Сигаев

www.postgrespro.ru

Postgres PROFESSIONAL

Мы



Agenda

- Deep Inspection
- Example
- Tips & tricks

FTS in Databases

Обычные поисковые машины не могут индексировать базы данных
Web-site – интерфейс к БД
Базы данных как часть **Hidden, Invisible, Dark, Deep** Web

- динамические страницы
- страницы, на которые никто не ссылается
- ограничение доступа
- javascipt, flash генерируемые линки
- бинарный контент

FTS in Database

Полнотекстовый поиск

- найти документы удовлетворяющие запросу
- отсортировать их в некотором порядке

Найти документы содержащие **все** слова из запроса и вернуть их отсортированными по похожести

Требования к FTS

- **полная интеграция с СУБД**
  - транзакционность
  - конкурентный доступ
  - восстановление после сбоев
  - online индекс
- Конфигурируемость (парсеры, словари,
- Масштабируемость

Наиболее привычный вид поиска

Что такое Документ?

Произвольный текстовый атрибут

Комбинация текстовых атрибутов из одной или разных (join) таблиц

Title || Abstract || Keywords || Body || Author

Test Search Operators

Традиционные операции текстового поиска
(TEXT op TEXT, op - ~, ~*, LIKE, ILIKE)

```
=# select title  from apod where title ~* 'x-ray' limit 5;
                 title
---------------------------------------
 The X-Ray Moon
 Vela Supernova Remnant in X-ray
 Tycho's Supernova Remnant in X-ray
 ASCA X-Ray Observatory
 Unexpected X-rays from Comet Hyakutake
(5 rows)

=# select title  from apod where title ilike '%x-ray%' limit 5;
```

What's wrong?

Нет поддержки лингвистики

- что есть слово ?
- что индексировать ?
- «нормализация» слов
- стоп-слова (noise-words)

Нет релевантности

- все документы одинаково «похожи»

Медленно, документы каждый раз сканируются

В 9.3+ появилась индексная поддержка (pg_trgm)

```
select * from man_lines where man_line ~* '(?:(?:p(?:ostgres(?:ql)?|g?sql)|sql)) (?:(?:(?:mak|us)e|do|is))';
```

FTS in PostgreSQL

- OpenFTS — 2000, Pg as a storage
- GiST index — 2000, thanks Rambler
- Tsearch — 2001, contrib:no ranking
- Tsearch2 — 2003, contrib:config
- GIN —2006, thanks, JFG Networks
- FTS — 2006, in-core, thanks, EnterpriseDB
- Now — Postgres Professional
  - Phrase search – PostgreSQL 9.6
  - RUM index – WIP

FTS in PostgreSQL

tsvector – хранилище для документов, оптимизированное для поиска

- отсортированный массив лексем
- позиционная информация
- структурная информация (важность)

tsquery – текстовый тип для запроса с логическими операторами & | ! ()

Полнотекстовый оператор: tsvector @@ tsquery

Операторы @>, <@ для tsquery

Функции: to_tsvector, to_tsquery, plainto_tsquery, ts_lexize, ts_debug, ts_stat, ts_rewrite,ts_headline, ts_rank, ts_rank_cd, setweight

Индексы: GiST, GIN

tsvector ???

Тип данных в PostgreSQL, оптимизированный для полнотекстового поиска.

```
=# select to_tsvector('russian',
     'и в грудь себе вонзает шешнадцать столовых НоЖей');

                      to_tsvector

-----------------------------------------------------

 'вонза':5 'груд':3 'нож':8 'столов':7 'шешнадца':6
```

Демо

```
# select to_tsvector('a fat cat sat on a mat and at a fat rat');
                  to_tsvector
-----------------------------------------------
 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4


# select plainto_tsquery('a fat cat rat');
    plainto_tsquery
-----------------------
 'fat' & 'cat' & 'rat'
```

FTS in PostgreSQL

Где выигрыш ?

документ обрабатывается при индексировании – не тратится время на обработку при поиске.

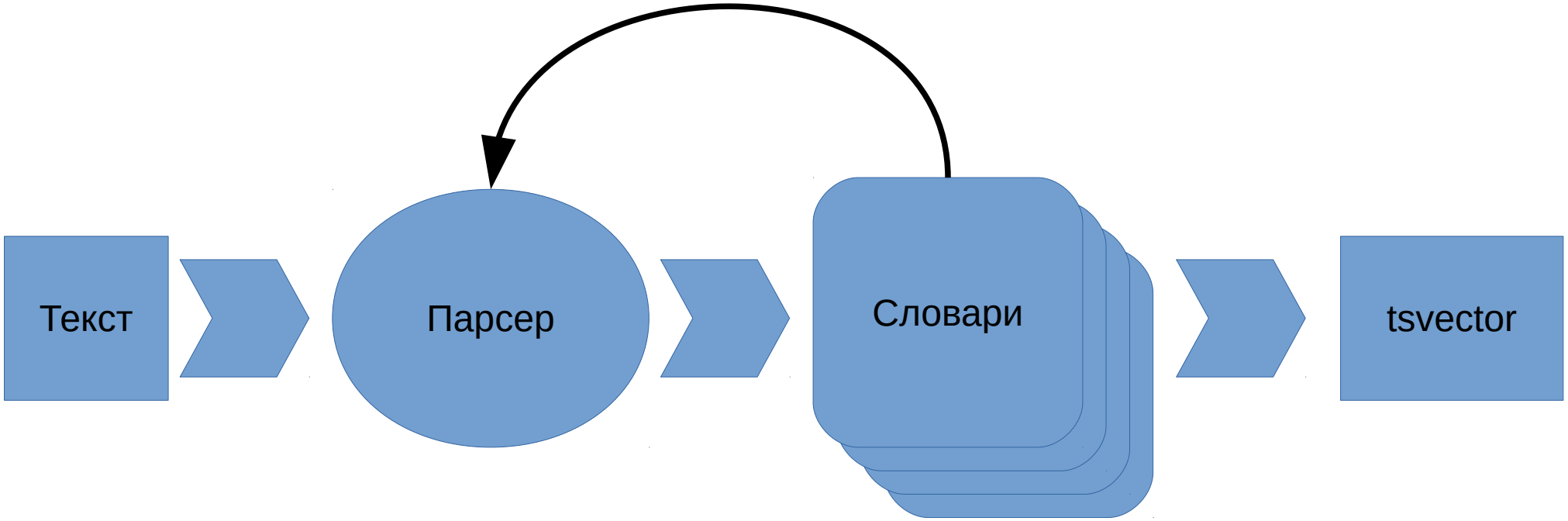
- документ разбивается на токены с помощью подключаемого парсера
- токены превращаются в лексемы с помощью подключаемых словарей
- запоминаются позиционная информация и важность лексемы, используется для ранжирования результатов
- стоп-слова игнорируются

Postgres PROFESSIONAL

Поговорим о

- Workflow
- Конфигурация
- Парсер
- Словари

Крупные мазки



Но языки, задачи, etc

Конфигурация!

- Парсер
- Словари
- Карта словарей (?)

Конфигурация

FTS конфигурация определяет

- какой парсер используется для разбивания текста на токены
- какие токены, какими словарями и в каком порядке обрабатываются

Конфигурация задается с помощью SQL команд

```
{CREATE | ALTER | DROP} TEXT SEARCH {CONFIGURATION | DICTIONARY | PARSER}
```

FTS конфигураций может быть много, поддерживаются схемы

Информация о конфигурации доступна в psql

\dF{,d,p}[+] [PATTERN]

Конфигурация

```
# \?
...
  \dF[+]  [PATTERN]      list text search configurations
  \dFd[+] [PATTERN]      list text search dictionaries
  \dFp[+] [PATTERN]      list text search parsers
  \dFt[+] [PATTERN]      list text search templates
...
```

Postgres PROFESSIONAL

Упс. Темплейты?!

Ispell — шаблон для словаря.

Набор методов, но без «данных». Если их укажем — будет словарь.

Конфигурации

```
# \dF
               List of text search configurations
   Schema   |    Name    |              Description
------------+------------+---------------------------------------
 pg_catalog | danish     | configuration for danish language
 pg_catalog | dutch      | configuration for dutch language
 pg_catalog | english    | configuration for english language
 pg_catalog | finnish    | configuration for finnish language
 pg_catalog | french     | configuration for french language
 pg_catalog | german     | configuration for german language
 pg_catalog | hungarian  | configuration for hungarian language
 pg_catalog | italian    | configuration for italian language
 pg_catalog | norwegian  | configuration for norwegian language
 pg_catalog | portuguese | configuration for portuguese language
 pg_catalog | romanian   | configuration for romanian language
 pg_catalog | russian    | configuration for russian language
 pg_catalog | simple     | simple configuration
 pg_catalog | spanish    | configuration for spanish language
 pg_catalog | swedish    | configuration for swedish language
 pg_catalog | turkish    | configuration for turkish language
(16 rows)
```

Конфигурация

```
# \dF+ russian
Text search configuration "pg_catalog.russian"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | russian_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | russian_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | russian_stem
```

psql -E

```
# \dF+ russian
********** QUERY **********
SELECT c.oid, c.cfgname,
   n.nspname,
   p.prsname,
   np.nspname as pnspname
FROM pg_catalog.pg_ts_config c
   LEFT JOIN pg_catalog.pg_namespace n ON n.oid =
c.cfgnamespace,
 pg_catalog.pg_ts_parser p
   LEFT JOIN pg_catalog.pg_namespace np ON np.oid =
p.prsnamespace
WHERE  p.oid = c.cfgparser
  AND c.cfgname ~ '^(russian)$'
  AND pg_catalog.pg_ts_config_is_visible(c.oid)
ORDER BY 3, 2;
**************************

...
```

Парсер

```
# \dFp+
    Text search parser "pg_catalog.default"
     Method      |    Function    | Description
-----------------+----------------+-------------
 Start parse     | prsd_start     | (internal)
 Get next token  | prsd_nexttoken | (internal)
 End parse       | prsd_end       | (internal)
 Get headline    | prsd_headline  | (internal)
 Get token types | prsd_lextype   | (internal)

        Token types for parser "pg_catalog.default"
   Token name    |               Description
-----------------+------------------------------------------
 asciihword      | Hyphenated word, all ASCII
 asciiword       | Word, all ASCII
 blank           | Space symbols
 email           | Email address
 entity          | XML entity
 file            | File or path name
 float           | Decimal notation
 host            | Host
 hword           | Hyphenated word, all letters
 hword_asciipart | Hyphenated word part, all ASCII
 hword_numpart   | Hyphenated word part, letters and digits
 hword_part      | Hyphenated word part, all letters
 int             | Signed integer
 numhword        | Hyphenated word, letters and digits
 numword         | Word, letters and digits
 protocol        | Protocol head
 sfloat          | Scientific notation
 tag             | XML tag
 uint            | Unsigned integer
 url             | URL
 url_path        | URL path
 version         | Version number
 word            | Word, all letters
(23 rows)
```

Словарь

```
# \x
Expanded display is on.
# \dFd+  russian_stem
List of text search dictionaries
-[ RECORD 1 ]+-----------------------------------
Schema       | pg_catalog
Name         | russian_stem
Template     | pg_catalog.snowball
Init options | language = 'russian', stopwords = 'russian'
Description  | snowball stemmer for russian language
```

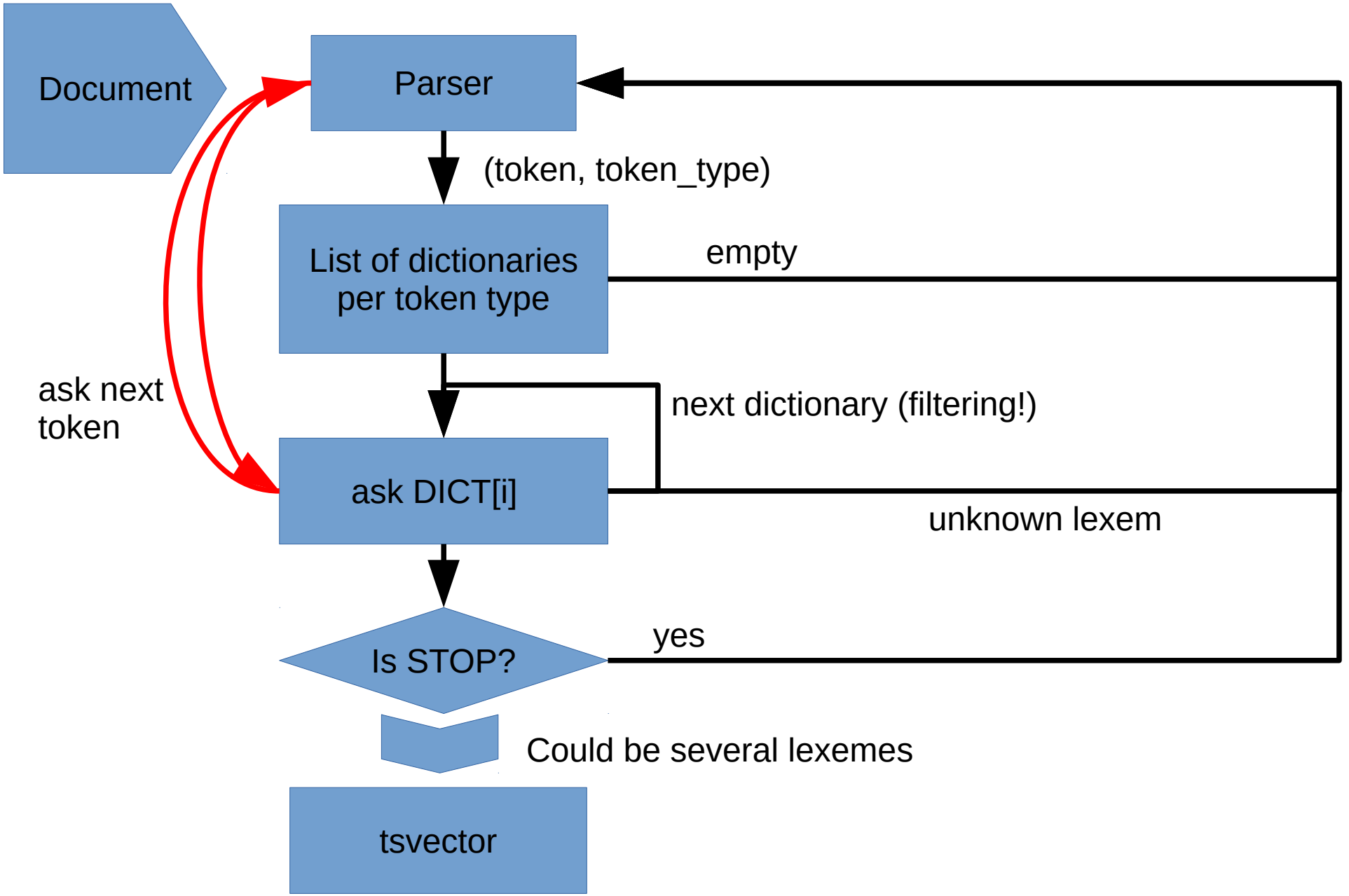
Template

```
# \dFt+ snowball
List of text search templates
-[ RECORD 1 ]-----------------
Schema      | pg_catalog
Name        | snowball
Init        | dsnowball_init
Lexize      | dsnowball_lexize
Description | snowball stemmer
```

Altering

```
# \h alter text search
Command:     ALTER TEXT SEARCH CONFIGURATION
Description: change the definition of a text search
configuration
Syntax:
ALTER TEXT SEARCH CONFIGURATION name
    ADD MAPPING FOR token_type [, ... ] WITH
dictionary_name [, ... ]
...
Command:     ALTER TEXT SEARCH DICTIONARY
Description: change the definition of a text search
dictionary
Syntax:
ALTER TEXT SEARCH DICTIONARY name (
    option [ = value ] [, ... ]
)
```

Более подробно



Парсер

Парсер разбивает текст на токены

```
=# select * from ts_token_type('default');
 tokid |      alias      |                description
-------+-----------------+-------------------------------------------
     1 | asciiword       | Word, all ASCII
     2 | word            | Word, all letters
     3 | numword         | Word, letters and digits
     4 | email           | Email address
     5 | url             | URL
     6 | host            | Host
     7 | sfloat          | Scientific notation
     8 | version         | Version number
     9 | hword_numpart   | Hyphenated word part, letters and digits
    10 | hword_part      | Hyphenated word part, all letters
    11 | hword_asciipart | Hyphenated word part, all ASCII
    12 | blank           | Space symbols
    13 | tag             | XML tag
    14 | protocol        | Protocol head
    15 | numhword        | Hyphenated word, letters and digits
    16 | asciihword      | Hyphenated word, all ASCII
    17 | hword           | Hyphenated word, all letters
    18 | url_path        | URL path
    19 | file            | File or path name
    20 | float           | Decimal notation
    21 | int             | Signed integer
    22 | uint            | Unsigned integer
    23 | entity          | XML entity
(23 rows)
```

Парсер, интерфейс.

Псевдокод! Псевдотипы!

- `void* parserInit(char* str, int length)`
- `TokenType parserGetNextToken(void* parser, char** token, int* tokenLength)`
- `void parserEnd(void* parser)`
- `LexDescr* parserGetLexemsTypes()`
- `HeadlineParsedText* parserHeadline( HeadlineParsedText* hpt, List* options, TSQuery *query)`

Словари

Словарь – это **программа**, которая принимает на вход токен и выдает массив лексем или метку, если распознанно стоп-слово
API позволяет писать словари под разные задачи

- Укорачивать длинные цифры
- Приводить все обозначения цветов в один вид
- Приводить URL-и к каноническому виду
- Фильтрация

Встроенные словари-заготовки (templates) для

- словарей ispell, myspell, hunspell
- snowball stemmer
- thesaurus
- synonym
- simple

Словари

Словарь — это программа !

```
=# select ts_lexize('intdict', 11234567890);
  ts_lexize
-----------
 {112345}
=# select ts_lexize('roman', 'XIX');
 ts_lexize
------------
 {19}

=# select ts_lexize('colours','#FFFFFF');
ts_lexize
------------
{white}
```

Словарь, интерфейс

```
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
  );
```

Пример: simple

```
# select ts_lexize('simple','ЛопаТАми');
 ts_lexize
-----------
 {лопатами}
(1 row)
# select ts_lexize('russian_stem','ЛопаТАми');
 ts_lexize
-----------
 {лопат}
(1 row)
```

Словарь, интерфейс

```
typedef struct {
      uint16      nvariant;
      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

Flags: TSL_ADDPOS, TSL_PREFIX, TSL_FILTER

Словарь, интерфейс

```
SELECT ts_lexize('en_ispell','bookings');

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

Агглютинативные языки

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) '

Агглютинативные языки

Agglutinative languages have several variants of word's splitting:

Word 'foobarcom' (imaginary)

Lexeme	nvariant	Comment
foo	1	
bar	1	
com	1	
foob	2	-a- is an affix (interfix)
rcom	2	

```
tsvector: 'bar:1 com:1 foo:1 foob:1 rcom:1'
tsquery: '(foob & rcom) | (foo & bar & com)'
```

TSLexeme->flags

Each TSLexeme describes one normalized lexeme
TSLexeme->flags is an OR-ed:

- TSL_PREFIX indicates to use prefix search for this lexeme
- TSL_ADDPOS points to parser to increase position's counter
  Note: currently only thesaurus dictionary uses it
- TSL_FILTER pass modified lexeme to subsequent dictionaries

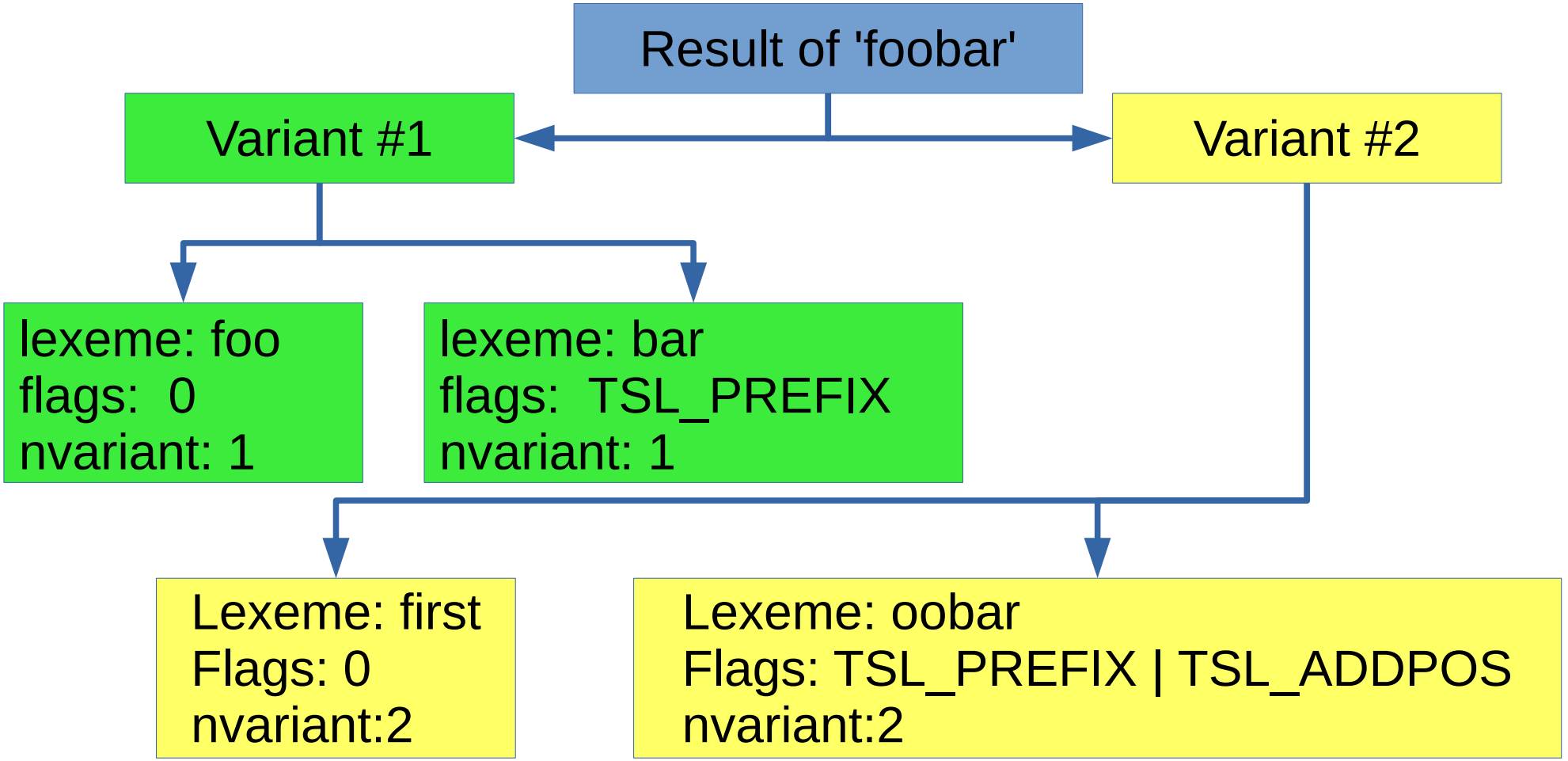
Synonyms

```
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');
 ts_lexize
-----------
 {index}
```

Synonyms

```
=# 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
```

TSLexeme




```
tsvector: 'foo:1 bar:1 first:1 oobar:2'
tsquery: '(foo & bar:*) | (first & oobar:*)'
```

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.

Пример: contrib/unaccent

contrib/unaccent - 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)

```
=# select ts_lexize('unaccent','Hotels') is NULL;
 ?column?
----------
 t
```

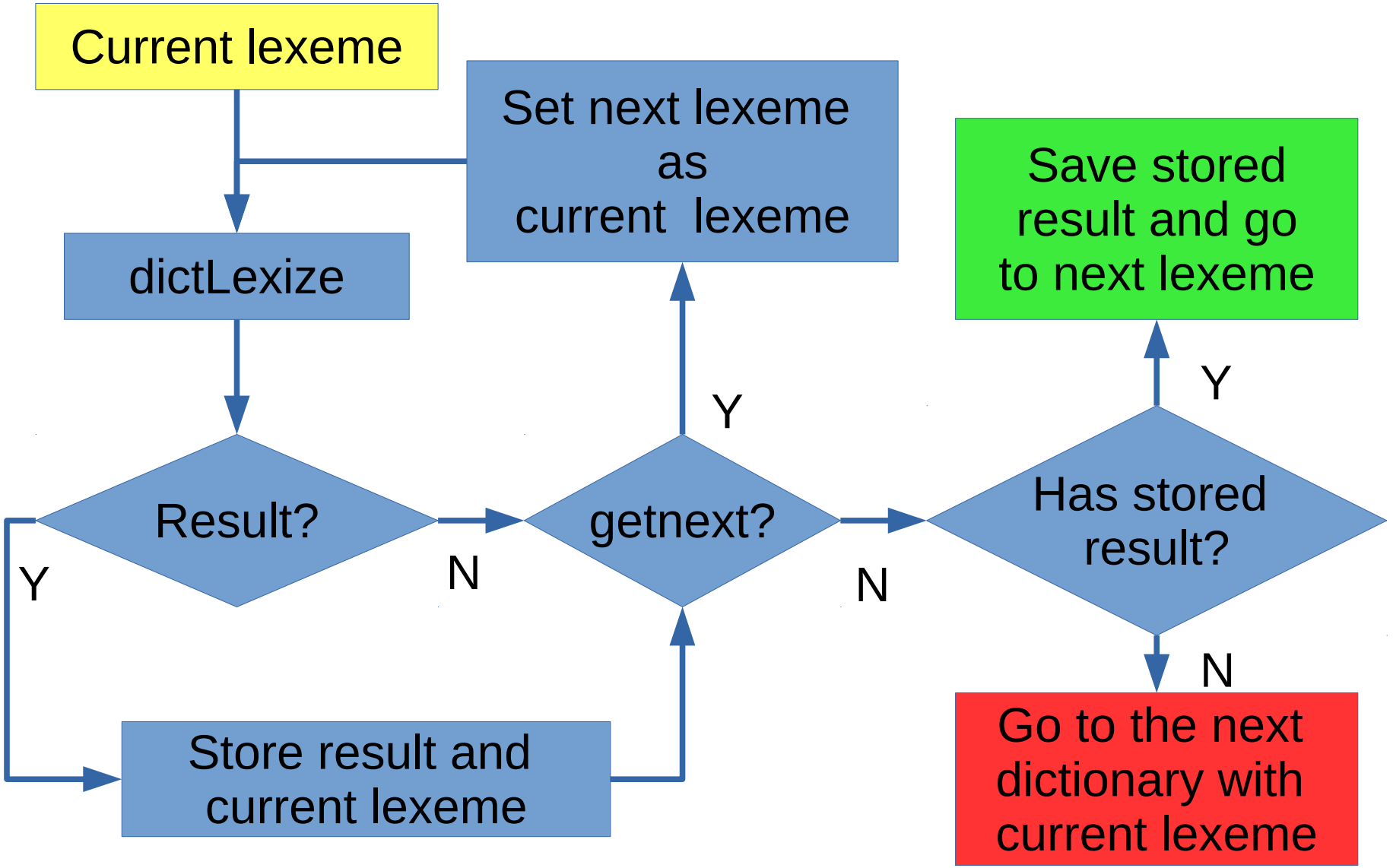
2. Unaccent dictionary removes accent and returns 'Hotel'.
(lexeme 'Hotel' will be passed to the next dictionary if any)

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

Словарь, интерфейс

```
/*
 * Struct for supporting complex dictionaries like thesaurus.
 * 4th argument for dictlexize method is a pointer to this
 */
typedef struct
{
    bool        isend;          /* in: marks for lexize_info about
                                 * text end is reached */
    bool        getnext;        /* out: dict wants next token */
    void       *private_state;  /* internal dict state between calls with
                                 * getnext == true */
} DictSubState;
```

Несколько лексем



Примеры словарей

- Simple
- Стеммеры
- Ispell/Myspell/Hunspell
- Synonym
- contrib/dict_int
- contrib/dict_xsyn
- contrib/unaccent

Астрономический словарь

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)([abcd]?) GRB$1$2
```

Postgres PROFESSIONAL

Индексы

Совсем чуть-чуть. Просто для понимания.

Индексы

- Индекс — это поисковое дерево, в листьях которого содержатся указатели на записи в таблице
- Индекс не содержит информации о видимости записи (MVCC !)
- Индексы только ускоряют выполнение запроса (операторы,операнды)
- Результаты выборки с использованием индекса должны совпадать с последовательным сканом и фильтрацией
- Индексы могут быть **partial** (where price > 0.0), **functional** (to_tsvector(text)), **multicolumn** (timestamp, tsvector)

GiST (RD-Tree)

Сигнатура слова — слово хэшируется в позицию '1'

w1 -> S1: 01000000 Document: w1 w2 w3
w2 -> S2: 00010000
w3 -> S3: 10000000

Сигнатура документа (запроса) — суперпозиция (bit-wise OR) индивидуальных сигнатур

S: 11010000

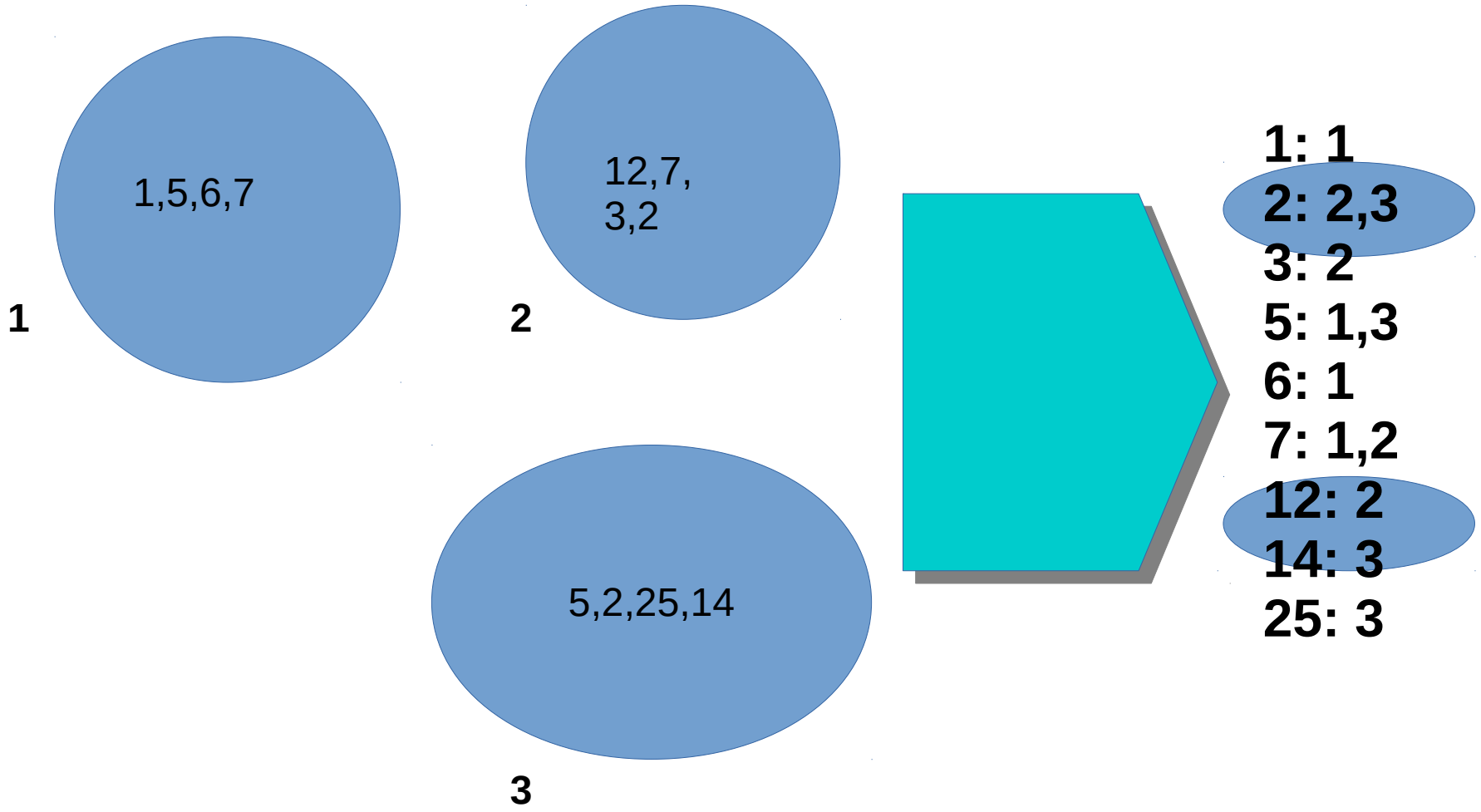
Фильтр Блюма (Bloom filter)

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

Сигнатура — неточное (lossy) представление док-та

- + fixed length, compact, + fast bit operations
- - lossy (false drops), - saturation with #words grows

GIN



Query: 2 & 12
Result: 2

Индексы

- GiST индекс для изменяющихся данных
  - быстро обновляется
  - не очень хорошо шкалируется
  - зависит от количества уникальных слова
- GiN индекс для архивных таблиц
  - дольше обновляется (при вставке документа из 1000 слов требуется сделать 1000 updates). Gin Fast Update проблему сильно ослабил !
  - хорошо шкалируется
  - очень слабо зависит от числа уникальных слов
- RUM — новый индекс, наследник GIN
  - Сильно быстрее в поиске с ранжированием
  - Умеет дополнительные условия

Пишем расширение для полнотекстового поиска

- Парсер
- Template (шаблон словаря)
- Словарь
- Конфигурация

https://github.com/postgrespro/tsexample

Простой парсер

Различаем 2 типа токенов:

- Слово
- Число

Токен – это непрерывный участок букв, цифр и знаков «_». Число – это токен состоящий только из цифр, остальное – слово.

Парсер, интерфейс.

Псевдокод! Псевдотипы!

- `void* parserInit(char* str, int length)`
- `TokenType parserGetNextToken(void* parser, char** token, int* tokenLength)`
- `void parserEnd(void* parser)`
- `LexDescr* parserGetLexemsTypes()`
- `HeadlineParsedText* parserHeadline( HeadlineParsedText* hpt, List* options, TSQuery *query)`

Парсер: список типов лексем

```
#define  WORD_TOKEN          1
#define  NUMBER_TOKEN        2
#define  LAST_TOKEN_NUM      2

static const char *const tok_alias[] = {"", "word", "number"};

static const char *const lex_descr[] = {"", "Word, all
  alphanumeric characters", "Number, all digits"};

Datum sparser_lextype(PG_FUNCTION_ARGS)
{
      LexDescr   *descr = (LexDescr *)
         palloc(sizeof(LexDescr) * (LAST_TOKEN_NUM + 1));
      int        i;
      for (i = 1; i <= LAST_TOKEN_NUM; i++)
      {
         descr[i - 1].lexid = i;
         descr[i - 1].alias = pstrdup(tok_alias[i]);
         descr[i - 1].descr = pstrdup(lex_descr[i]);
      }
      descr[LAST_TOKEN_NUM].lexid = 0;
      PG_RETURN_POINTER(descr);
}
```

Postgres PROFESSIONAL

Datum

- Datum – это unsigned integer достаточной длины, чтобы хранить в себе pointer.
- Любое значение PostgreSQL может быть приведено к типу Datum и наоборот. Для этого используются макросы DatumGet*(x) и *GetDatum(x).
- Значения, которые помещаются в Datum, могут быть переданы по значению, остальные передаются по указателю.
- Аргументы функции и возвращаемые значения передаются как Datum.

PostgreSQL calling convention version 1

- Параметры передаются через специальную структуру FunctionCallInfo.

  /* Standard parameter list for fmgr-compatible functions */

  #define PG_FUNCTION_ARGSFunctionCallInfo fcinfo

  typedef struct FunctionCallInfoData *FunctionCallInfo;

- Для доступа к параметрам заведены макросы PG_GET_ARG_*(n)

  #define PG_GETARG_DATUM(n) (fcinfo->arg[n])

  #define PG_GETARG_INT32(n) DatumGetInt32(PG_GETARG_DATUM(n))

- Для возвращения результата используются макросы PG_RETURN_*(x)

  #define PG_RETURN_DATUM(x) return (x)

  #define PG_RETURN_INT32(x) return Int32GetDatum(x)

- PG_FUNCTION_INFO_V1(funcname) указывает на использование calling convention version 1, был ещё version 0...

Парсер: инициализация

```
typedef struct
{
	char	*begin;
	char	*end;
	char	*p;
} SParserStatus;

Datum
sparser_start(PG_FUNCTION_ARGS)
{
	SParserStatus  *status = (SParserStatus *)
	   palloc0(sizeof(SParserStatus));

	status->begin = (char *) PG_GETARG_POINTER(0);
	status->end = status->begin + PG_GETARG_INT32(1);
	status->p = status->begin;

	PG_RETURN_POINTER(status);
}
```

Парсер: следующий токен

```
Datum sparser_nexttoken(PG_FUNCTION_ARGS)
{
        SParserStatus   *status = (SParserStatus *) PG_GETARG_POINTER(0);
        char        **t = (char **) PG_GETARG_POINTER(1);
        int             *tlen = (int *) PG_GETARG_POINTER(2);
        bool         found = false, has_nondigit = false;
        while (status->p < status->end)
        {
            int p_len = pg_mblen(status->p);
            if (t_isalpha(status->p) || t_isdigit(status->p) ||
                (p_len == 1 && *status->p == '_'))
            {
                if (!t_isdigit(status->p)) has_nondigit = true;
                if (!found)
                {
                    *t = status->p;
                    found = true;
                }
            }
            else if (found) break;
            status->p += p_len;
        }
        if (found)
        {
            *tlen = status->p - *t;
            if (has_nondigit) PG_RETURN_INT32(WORD_TOKEN);
            else PG_RETURN_INT32(NUMBER_TOKEN);
        }
        else
            PG_RETURN_INT32(0);
}
```

Парсер: завершение

```
Datum
sparser_end(PG_FUNCTION_ARGS)
{
	SParserStatus  *status = (SParserStatus *)
	   PG_GETARG_POINTER(0);

	pfree(status);
	PG_RETURN_VOID();
}
```

Парсер: SQL-определение

```
CREATE OR REPLACE FUNCTION sparser_start(internal, integer)
RETURNS internal AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE;

CREATE OR REPLACE FUNCTION sparser_nexttoken(internal, internal,
  internal)
RETURNS internal AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE;

CREATE OR REPLACE FUNCTION sparser_end(internal)
RETURNS void AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE;

CREATE OR REPLACE FUNCTION sparser_lextype(internal)
RETURNS internal AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE;

CREATE TEXT SEARCH PARSER sample_parser (
      START = sparser_start,
      GETTOKEN = sparser_nexttoken,
      END = sparser_end,
      LEXTYPES = sparser_lextype
);
COMMENT ON TEXT SEARCH PARSER sample_parser
IS 'sample word parser';
```

Проверяем парсер

```
# SELECT * FROM ts_parse('sample_parser', 'abc def 123 1xx yy3
   pg_config');
 tokid |   token
-------+-----------
     1 | abc
     1 | def
     2 | 123
     1 | 1xx
     1 | yy3
     1 | pg_config
(6 rows)
```

Шаблон словаря

Тестовый словарь: откусываем серединку у длинных слов.

Параметры:

- nbegin – сколько символов оставляем в начале
- nend – сколько символов оставляем в конце

Словарь, интерфейс

```
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
  );
```

Шаблон словаря: инициализация

```
Datum cutdict_init(PG_FUNCTION_ARGS)
{
	List		*dictoptions = (List *) PG_GETARG_POINTER(0);
	CutDict	*d = (CutDict *) palloc0(sizeof(CutDict));
	bool		nbegin_loaded = false,
			nend_loaded = false;
	ListCell	*l;
	foreach(l, dictoptions)
	{
	    DefElem	*defel = (DefElem *) lfirst(l);
	    if (pg_strcasecmp("nbegin", defel->defname) == 0)
	    {
	        if (nbegin_loaded) ereport(ERROR, ...);
	        d->nbegin = atoi(defGetString(defel));
	        nbegin_loaded = true;
	    }
	    else if (pg_strcasecmp("nend", defel->defname) == 0)
	    {
	        if (nend_loaded) ereport(ERROR, ...);
	        d->nend = atoi(defGetString(defel));
	        nend_loaded = true;
	    }
	    else ereport(ERROR, ...);
	}
	if (!nbegin_loaded || !nend_loaded) ereport(ERROR, ...);
	PG_RETURN_POINTER(d);
}
```

Словарь, интерфейс

```
typedef struct {
      uint16      nvariant;
      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

Flags: TSL_ADDPOS, TSL_PREFIX, TSL_FILTER

Шаблон словаря: обработка токена

```
Datum cutdict_lexize(PG_FUNCTION_ARGS)
{
      CutDict      *d = (CutDict *) PG_GETARG_POINTER(0);
      char      *in = (char *) PG_GETARG_POINTER(1);
      int32     len = PG_GETARG_INT32(2);
      char      *txt;
      int       strlen, residx = 0;
      TSLexeme   *res;
      uint16    nvariant = 1;

      res = palloc0(sizeof(TSLexeme) * 4);
      txt = lowerstr_with_len(in, len);
      strlen = pg_mbstrlen(txt);
      if (strlen <= d->nbegin + d->nend)
      {
         res[residx].nvariant = nvariant;
         res[residx++].lexeme = txt;
         nvariant++;
      }
…..................................................
```

Шаблон словаря: обработка токена

```
…..........................................................
      if (strlen > d->nbegin)
      {
         int    i;
         char   *p = txt;

         for (i = 0; i < d->nbegin; i++)
            p += pg_mblen(p);
         res[residx].nvariant = nvariant;
         res[residx++].lexeme = pnstrdup(txt, p - txt);
      }
      if (strlen > d->nend)
      {
         int    i;
         char   *p = txt;

         for (i = 0; i < strlen - d->nend; i++)
            p += pg_mblen(p);
         res[residx].nvariant = nvariant;
         res[residx++].lexeme = pstrdup(p);
      }
      PG_RETURN_POINTER(res);
}
```

Шаблон словаря: SQL-определение

```
CREATE OR REPLACE FUNCTION cutdict_init(internal)
RETURNS internal AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE;

CREATE OR REPLACE FUNCTION cutdict_lexize(internal,
internal, internal, internal)
RETURNS internal AS 'MODULE_PATHNAME' LANGUAGE C STRICT IMMUTABLE;

CREATE TEXT SEARCH TEMPLATE cutdict (
      INIT = cutdict_init,
      LEXIZE = cutdict_lexize
);

COMMENT ON TEXT SEARCH TEMPLATE cutdict IS 'cut dictionary:
  lowercase and keep only beginning and ending of long words';
```

Словарь и конфигурация

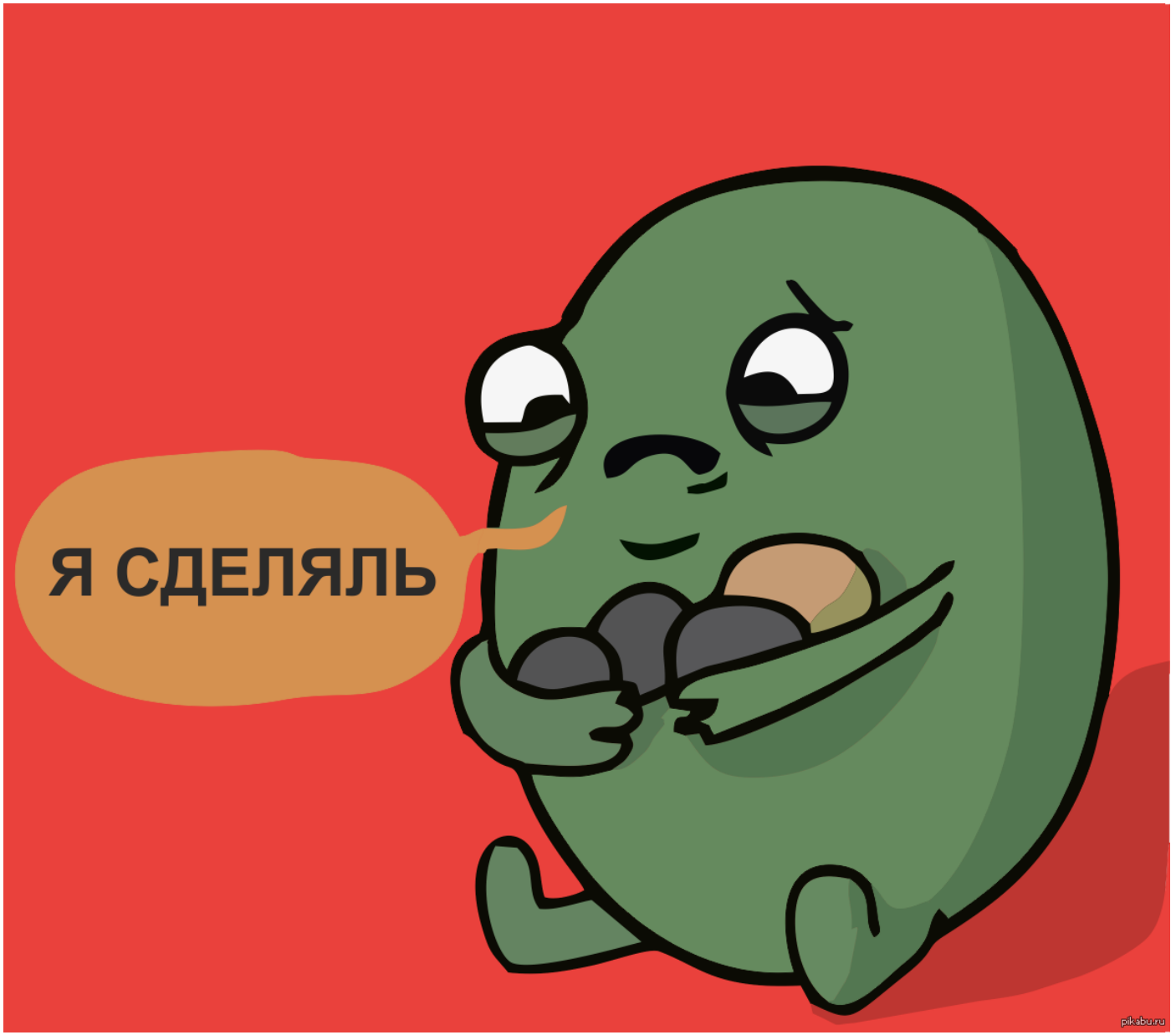
```
CREATE TEXT SEARCH DICTIONARY cut3 (
      TEMPLATE = cutdict,
      nbegin = 3,
      nend = 3
);
COMMENT ON TEXT SEARCH DICTIONARY cut3 IS
'cut dictionary with nbegin = nend = 3';

CREATE TEXT SEARCH CONFIGURATION sample (
      PARSER = "sample_parser"
);
ALTER TEXT SEARCH CONFIGURATION sample
ADD MAPPING FOR word WITH cut3;
ALTER TEXT SEARCH CONFIGURATION sample
ADD MAPPING FOR number WITH simple;

COMMENT ON TEXT SEARCH CONFIGURATION sample IS 'sample
  configuration';
```

Проверяем

```
# SELECT * FROM to_tsvector('sample', 'longlonglongword');
   to_tsvector
-----------------
 'lon':1 'ord':1

# SELECT * FROM to_tsquery('sample', 'longword');
  to_tsquery
---------------
 'lon' & 'ord'

# SELECT to_tsvector('sample', 'longlonglongword') @@
  to_tsquery('sample', 'longword');
 ?column?
----------
 t
(1 row)
```

Postgres PROFESSIONAL

Наш экстеншен готов!



Tips and Tricks 1

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

Little function helps:

```
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
configuraton to have really good cache key !

Tips and Tricks 2

How to find documents, which contain emails ?

```
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;
```

Tips and Tricks 2

```
=# SELECT document_token_types(title) FROM papers
        LIMIT 10;
                        document_token_types
-----------------------------------------------------------
 {asciihword,asciiword,blank,hword_asciipart}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank,float,host}
 {asciiword,blank}
 {asciihword,asciiword,blank,hword_asciipart,int,numword,uint}
 {asciiword,blank}
 {asciiword,blank}
(10 rows)

CREATE INDEX fts_types_idx ON papers USING
                        gin( document_token_types (title) );
```

Tips and Tricks 2

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');
```

Tips and Tricks 3

```
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;

SELECT id, (ts_stat(fts)).* FROM apod WHERE id=1;
 id |    word     | ndoc | nentry
----+-------------+------+--------
  1 | 1           |    1 |      1
  1 | 2           |    1 |      2
  1 | io          |    1 |      2
  1 | may         |    1 |      1
  1 | new         |    1 |      1
  1 | red         |    1 |      1
  1 | two         |    1 |      1
```

Tips and Tricks 4

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

```
=# select to_tsquery('ob_1','inferences') @@
                to_tsvector('ob_1','inference');
 ?column?
----------
 f
```

Use ts_debug() to understand the problem

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

'inference':
{french_ispell,french_stem} | french_ispell | {inference}
```

Tips and Tricks 4

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

Tips and Tricks 5

Use functional index (GiST or GiN)

- no ranking, use other ordering

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


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

Tips and Tricks 5

ts_headline() функция медленная – используйте **subselect**

790 times

```
select id,ts_headline(body,q),ts_rank(fts,q) as rank
from apod, to_tsquery('stars') q
where fts @@ q order by rank desc limit 10;

Time: 723.634 ms
```

10 times !

```
select id,ts_headline(body,q),ts_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;

Time: 21.846 ms

=#select count(*)from apod where fts @@ to_tsquery('stars');
 count
-------
   790
```

Tips and Tricks 6

Изменение запроса **online**

- расширение запроса
  - синонимы (new york => Gottham, Big Apple, ...)
- Сужение запроса
  - Курск => подводная лодка Курск

Похоже на словарь тезаурус (синонимов), но не требует переиндексации

Tips and Tricks 7

Если не нужна релевантность, то позиционная информация не нужна — можно иметь табличку сильно меньше !

```
postgres=# select to_tsvector('w1 w3 w1 w3');
    to_tsvector
--------------------
 'w1':1,3 'w3':2,4
(1 row)

Time: 0.268 ms
postgres=# select strip(to_tsvector('w1 w3 w1 w3'));
   strip
-----------
 'w1' 'w3'
(1 row)
```

Tips and Tricks 8

gin_stat() vs ts_stat()

```
=# 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;

  word  | exact  | estimation | error
--------+--------+------------+-------
 page   | 340430 |     340858 | 0.13%
 figur  | 240104 |     240366 | 0.11%
 use    | 147132 |     148022 | 0.60%
 model  | 133444 |     134442 | 0.74%
 result | 128977 |     129010 | 0.03%
(5 rows)
Time: 550.562 ms
```

Tips and Tricks 8

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

Tips and Tricks 9

Не только полнотекст, но и

- Индексирование shared libraries
- Гены/геном
- Химические соединения

Т.е. универсальный механизм расшибания на подпоследовательности и их поиск

PROFESSIONAL Postgres

www.postgrespro.ru