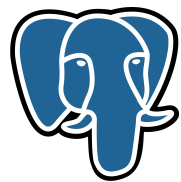


全文检索、中文分词 模糊查询、相似文本查询



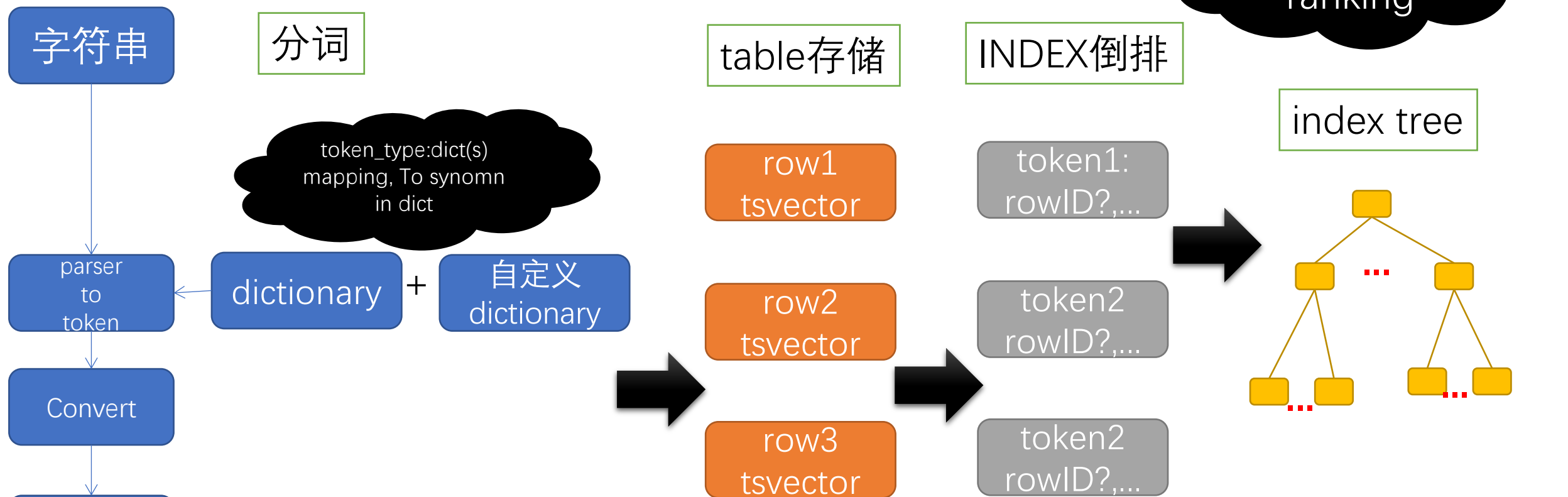
阿里云
digoal

目录

- 全文检索原理
- 自定义分词
- rank排序
- 关键词分析
- gin索引
- rum索引
- 全文检索+附加过滤
- 模糊查询
- 单字模糊查询
- 双字模糊查询
- wchar模糊查询注意事项
- 相似文本查询

- 文本特征向量：tsvector
- 检索条件：tsquery
- 中文分词插件
 - pg_scws
 - pg_jieba
 - zhparser

全文检索技术



```
select * from tbl where ts @@ to_tsquery('pg & alibaba')
order by ts_rank(ts, to_tsquery('pg & alibaba') );
-- order by ts_rank_cd(ts, to_tsquery('pg & alibaba') )
```

<https://www.postgresql.org/docs/10/static/textsearch-controls.html#TEXTSEARCH-RANKING>

全文检索技术 - 词距离条件

```
select * from tbl where ts @@ '速度 <距离值> 激情'::tsquery;
```

如

```
select * from tbl where ts @@ '速度 <1> 激情'::tsquery;
```

	ts
	'优质服务':9 '公司':8 '出租汽车':7 '创业':6 '创新':2 '北京':5 '坚持':3 '小花':10 '激情':1 '絮':11 '速度':4
	'激情':3 '电影':1 '破':5 '票房':4 '速度':2

按距离范围搜索

自定义UDF，RANGE相交操作判断。

https://github.com/digoal/blog/blob/master/201801/20180123_01.md

-- 测试，取距离是1到2（不含2）的

```
postgres=# select get_lexeme_pos_range(ts, '速度', '激情'), * from ts_test where ts @@ tsquery '速度 & 激情' and get_lexeme_pos_range(ts, '速度', '激情') && int4range(1,2) limit 1;
```

get_lexeme_pos_range	id	info	ts
[1,2)	1	电影速度与激情8的票房破亿	'激情':3 '电影':1 '破':5 '票房':4 '速度':2

(1 row)

Time: 0.713 ms

-- 测试，取距离是2到5（不含5）的

```
postgres=# select get_lexeme_pos_range(ts, '速度', '激情'), * from ts_test where ts @@ tsquery '速度 & 激情' and get_lexeme_pos_range(ts, '速度', '激情') && int4range(2,5) limit 1;
```

get_lexeme_pos_range	id	info	ts
[3,4)	1	激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮	'优质服务':9 '公司':8 '出租汽车':7 '创业':6 '创新':2 '北京':5 '坚持':3 '小花':10 '激情':1 '絮':11 '速度':4

(1 row)

Time: 0.682 ms

全文检索技术 - 词距离条件

带距离

```
postgres=# select * from ts_test where ts @@ '速度 <1> 激情'::tsquery;
```

id	info	ts
1	电影速度与激情8的票房破亿	'激情':3 '电影':1 '破':5 '票房':4 '速度':2

(1 row)

不带距离

```
postgres=# select * from ts_test where ts @@ '速度 & 激情'::tsquery limit 5;
```

id	info	ts
1	激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮	'优质服务':9 '公司':8 '出租汽车':7 '创业':6
1	电影速度与激情8的票房破亿	'激情':3 '电影':1 '破':5 '票房':4 '速度':2
2	激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮	'优质服务':9 '公司':8 '出租汽车':7 '创业':6
2	激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮	'优质服务':9 '公司':8 '出租汽车':7 '创业':6
2	激情，创新，坚持，速度-- 北京北方创业出租汽车公司优质服务小花絮	'优质服务':9 '公司':8 '出租汽车':7 '创业':6

(5 rows)

全文检索技术 - 内置ranking

```
UPDATE tt SET ti =  
  setweight(to_tsvector(coalesce(title,'')), 'A') ||  
  setweight(to_tsvector(coalesce(keyword,'')), 'B') ||  
  setweight(to_tsvector(coalesce(abstract,'')), 'C') ||  
  setweight(to_tsvector(coalesce(body,'')), 'D');
```

支持4种weight：
标题、作者、摘要、
内容

内置ranking
算法

```
SELECT title, ts_rank_cd(textsearch, query) AS rank  
FROM apod, to_tsquery('neutrino|(dark & matter)') query  
WHERE query @@ textsearch  
ORDER BY rank DESC  
LIMIT 10;
```

title	rank
Neutrinos in the Sun	3.1
The Sudbury Neutrino Detector	2.4
A MACHO View of Galactic Dark Matter	2.01317
Hot Gas and Dark Matter	1.91171
The Virgo Cluster: Hot Plasma and Dark Matter	1.90953
Rafting for Solar Neutrinos	1.9
NGC 4650A: Strange Galaxy and Dark Matter	1.85774
Hot Gas and Dark Matter	1.6123
Ice Fishing for Cosmic Neutrinos	1.6
Weak Lensing Distorts the Universe	0.818218

全文检索技术 - ranking掩码

Both ranking functions take an integer *normalization* option that specifies whether and how a document's length should impact its rank.

0 (the default) ignores the document length

1 divides the rank by 1 + the logarithm of the document length

2 divides the rank by the document length

4 divides the rank by the mean harmonic distance between extents
(this is implemented only by ts_rank_cd)

8 divides the rank by the number of unique words in document

16 divides the rank by 1 + the logarithm of the number of unique words in document

32 divides the rank by itself + 1

内置ranking
算法

```
SELECT title, ts_rank_cd(textsearch, query, 32) /* rank/(rank+1) */ AS rank
FROM apod, to_tsquery('neutrino|(dark & matter)') query
WHERE query @@ textsearch
ORDER BY rank DESC
LIMIT 10;
```

title	rank
Neutrinos in the Sun	0.756097569485493
The Sudbury Neutrino Detector	0.705882361190954
A MACHO View of Galactic Dark Matter	0.668123210574724
Hot Gas and Dark Matter	0.65655958650282
The Virgo Cluster: Hot Plasma and Dark Matter	0.656301290640973
Rafting for Solar Neutrinos	0.655172410958162
NGC 4650A: Strange Galaxy and Dark Matter	0.650072921219637
Hot Gas and Dark Matter	0.617195790024749
Ice Fishing for Cosmic Neutrinos	0.615384618911517
Weak Lensing Distorts the Universe	0.450010798361481

全文检索技术 - 内置ranking

The two ranking functions currently available are:

`ts_rank([weights float4[],] vector tsvector, query tsquery [, normalization integer]) returns float4`

Ranks vectors based on the frequency of their matching lexemes.

`ts_rank_cd([weights float4[],] vector tsvector, query tsquery [, normalization integer]) returns float4`

This function computes the *cover density* ranking for the given document vector and query, as described in Clarke, Cormack, "Processing and Management", 1999. Cover density is similar to `ts_rank` ranking except that the proximity of matching lexeme

This function requires lexeme positional information to perform its calculation. Therefore, it ignores any "stripped" lexemes in [Section 12.4.1](#) for more information about the `strip` function and positional information in tsvectors.)

For both these functions, the optional *weights* argument offers the ability to weigh word instances more or less heavily depending on word, in the order:

```
{D-weight, C-weight, B-weight, A-weight}
```

If no *weights* are provided, then these defaults are used:

```
{0.1, 0.2, 0.4, 1.0}
```

全文检索技术 - 自定义ranking

1、店铺标签表：

```
create table tbl (  
  shop_id int8 primary key, -- 店铺 ID  
  tags text[], -- 数组, 标签1,标签2,.....  
  scores float8[] -- 数组, 评分1,评分2,.....  
);  
  
create index idx_tbl_1 on tbl using gin(tags);
```

国民_足浴,国民_餐饮,娱乐_KTV

0.99,0.1,0.45

2、标签权值表：

```
create table tbl_weight (  
  tagid int primary key, -- 标签ID  
  tagname name, -- 标签名  
  desc text, -- 标签描述  
  weight float8 -- 标签权值  
);  
  
create index idx_tbl_weight_1 on tbl_weight (tagname);
```

```
create or replace function cat_ranking(tsquery) returns float8 as $$  
declare  
  
begin  
  for each x in array (contains_element) loop  
    search hit element's score.  
    search hit element's weight.  
    cat ranking and increment  
  end loop;  
  return res;  
end;  
$$ language plpgsql strict;
```

ranking sort by index

- rum index am

```
CREATE EXTENSION rum;
```

```
CREATE INDEX rumidx ON test_rum USING rum (a rum_tsvector_ops);
```

```
SELECT t, a <=> to_tsquery('english', 'beautiful | place') AS rank
FROM test_rum
WHERE a @@ to_tsquery('english', 'beautiful | place')
ORDER BY a <=> to_tsquery('english', 'beautiful | place');
           t                      | rank
```

```
-----+-----
It looks like a beautiful place | 8.22467
The situation is most beautiful | 16.4493
It is a beautiful                | 16.4493
(3 rows)
```

rds pg zhparser 自定义分词

https://help.aliyun.com/knowledge_detail/44451.html

```
create extension zhparser ;
```

```
CREATE TEXT SEARCH CONFIGURATION testzhcfg (PARSER = zhparser);
```

```
ALTER TEXT SEARCH CONFIGURATION testzhcfg ADD MAPPING FOR n,v,a,i,e,l WITH simple;
```

```
db1=> SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

```
NOTICE: zhparser add dict : "/data/tsearch_data/dict_extra.xdb" failed!
```

```
to_tsquery
```

```
-----  
'保障' & '房' & '资金' & '压力'
```

```
(1 row)
```

rds pg zhparser自定义分词

```
db1=> insert into pg_ts_custom_word values ('保障房资');
```

```
INSERT 0 1
```

```
db1=> select zhprs_sync_dict_xdb();
```

```
db1=> SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

```
to_tsquery
```

```
-----  
'保障房资' & '压力'
```

```
(1 row)
```

```
db1=> alter role all set zhparser.multi_short=on;
```

```
ALTER ROLE
```

```
db1=> SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

```
to_tsquery
```

```
-----  
'保障房资' & '保障' & '压力'
```

```
(1 row)
```

提示：内核小版本20160801及之后的版本才支持自定义中文分词词典，您可以通过show rds_release_date ;SQL语句进行查询内核版本。使用自定义分词的注意事项如下。

- 最多支持一百万条自定义分词，超出部分不做处理，用户必须保证分词数量在这个范围之内。自定义分词与缺省的分词词典将共同产生作用。
- 每个词的最大长度为128字节，超出部分将会截取。
- 通过增删改分词之后必须执行select zhprs_sync_dict_xdb();SQL语句，并且重新建立连接才会生效。

关键词统计

- ts_stat

The function `ts_stat` is useful for checking your configuration and for finding stop-word candidates.

```
ts_stat(sqlquery text, [ weights text, ]  
      OUT word text, OUT ndoc integer,  
      OUT nentry integer) returns setof record
```

sqlquery is a text value containing an SQL query which must return a single `tsvector` column. `ts_stat` contains the `tsvector` data. The columns returned are

- **word** text — the value of a lexeme
- **ndoc** integer — number of documents (`tsvectors`) the word occurred in
- **nentry** integer — total number of occurrences of the word

If **weights** is supplied, only occurrences having one of those weights are counted.

For example, to find the ten most frequent words in a document collection:

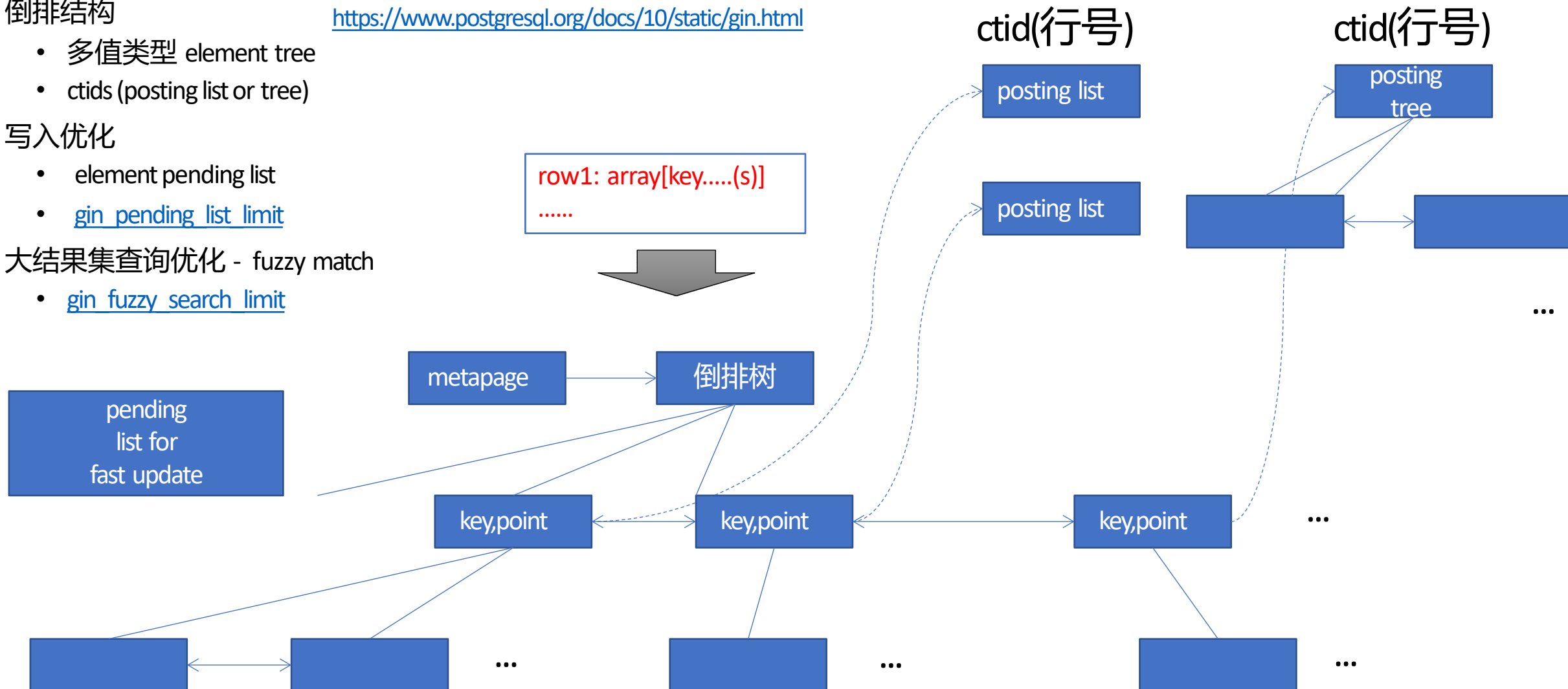
```
SELECT * FROM ts_stat('SELECT vector FROM apod')  
ORDER BY nentry DESC, ndoc DESC, word  
LIMIT 10;
```

目录

- 全文检索原理
- 自定义分词
- rank排序
- 关键词分析
- gin索引
- rum索引
- 全文检索+附加过滤
- 模糊查询
- 单字模糊查询
- 双字模糊查询
- wchar模糊查询注意事项
- 相似文本查询

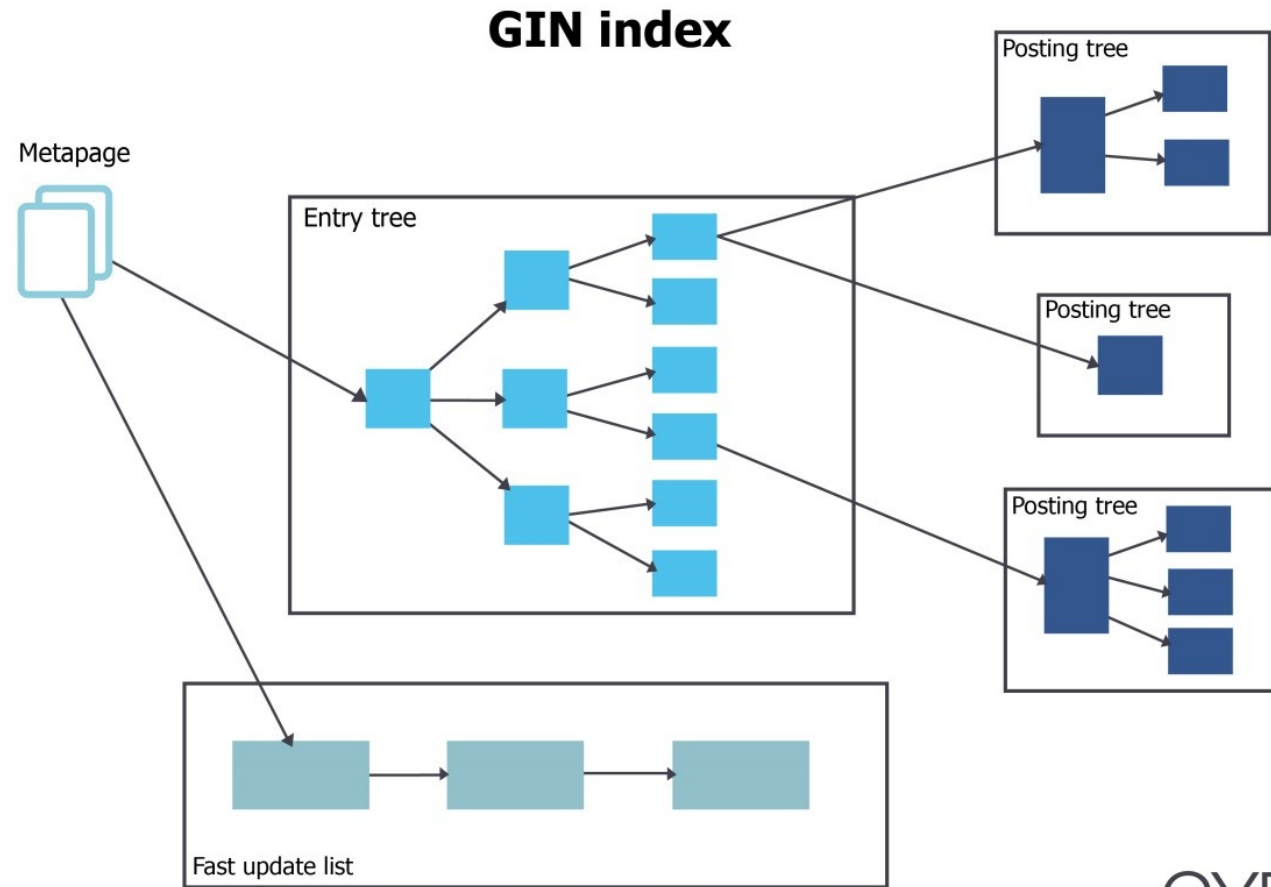
gin索引结构

- [src/backend/access/gin/README](#)
- 倒排结构
 - 多值类型 element tree
 - ctids (posting list or tree)
- 写入优化
 - element pending list
 - [gin_pending_list_limit](#)
- 大结果集查询优化 - fuzzy match
 - [gin_fuzzy_search_limit](#)



gin索引结构

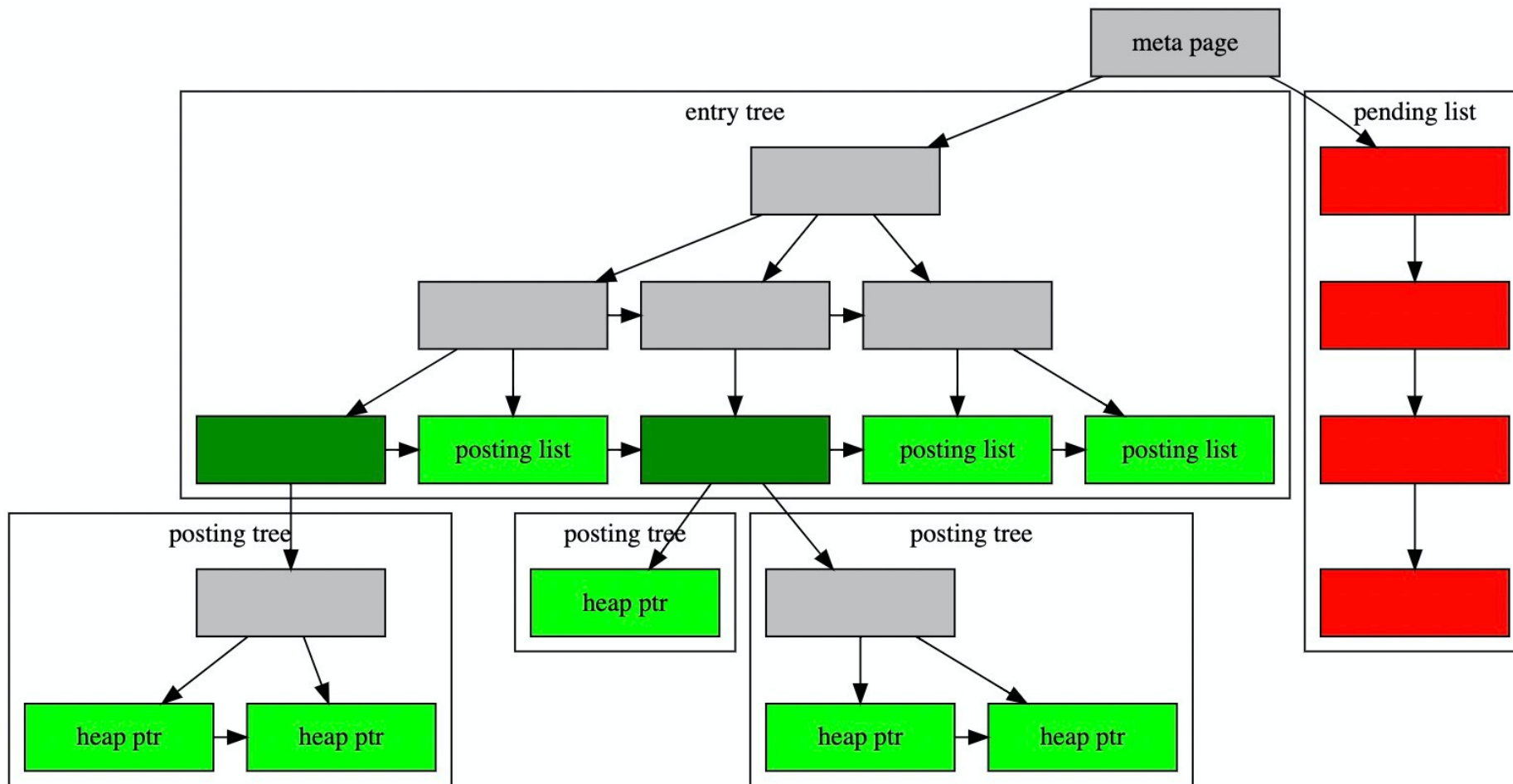
- <https://www.cybertec-postgres.com/>
- <https://www.postgresql.org/docs/>



gin索引结构

Figure 66.1. GIN Internals

- <https://www.cybertec-po>
- <https://www.postgresql.or>



`gin_page_opaque_info` returns information about a GIN index opaque area, like t

```
test=# SELECT * FROM gin_page_opaque_info(get_raw_page('gin_index', 2));
 rightlink | maxoff |          flags
-----+-----+-----
          5 |         0 | {data, leaf, compressed}
(1 row)
```

`gin_leafpage_items` returns information about the data stored in a GIN leaf page. For example:

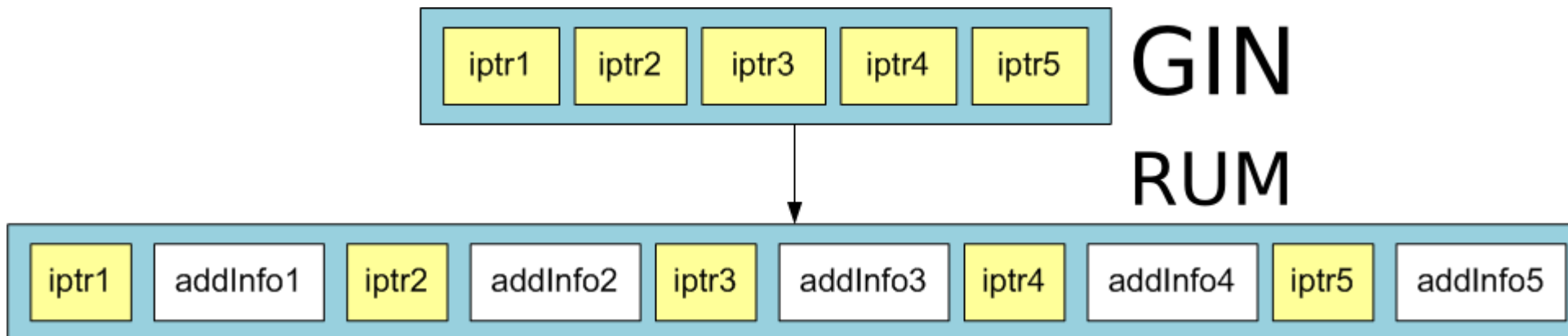
内窥GIN内容 [https://www.postgres](https://www.postgres.cn/)
[1.11.7.31.7](#)

```
test=# SELECT first_tid, nbytes, tids[0:5] AS some_tids
       FROM gin_leafpage_items(get_raw_page('gin_test_idx', 2));
 first_tid | nbytes |          some_tids
-----+-----+-----
(8, 41)    |    244 | {"(8, 41)", "(8, 43)", "(8, 44)", "(8, 45)", "(8, 46)"}
(10, 45)   |    248 | {"(10, 45)", "(10, 46)", "(10, 47)", "(10, 48)", "(10, 49)"}
(12, 52)   |    248 | {"(12, 52)", "(12, 53)", "(12, 54)", "(12, 55)", "(12, 56)"}
(14, 59)   |    320 | {"(14, 59)", "(14, 60)", "(14, 61)", "(14, 62)", "(14, 63)"}
(167, 16)  |    376 | {"(167, 16)", "(167, 17)", "(167, 18)", "(167, 19)", "(167, 20)"}
(170, 30)  |    376 | {"(170, 30)", "(170, 31)", "(170, 32)", "(170, 33)", "(170, 34)"}
(173, 44)  |    197 | {"(173, 44)", "(173, 45)", "(173, 46)", "(173, 47)", "(173, 48)"}
(7 rows)
```

rum索引结构

- <https://github.com/postgrespro/rum>
- https://github.com/digoal/blog/blob/master/201907/20190706_01.md

额外信息 (例如attach column's value, tsvector's 出现次数等)



```
SELECT t, a <=> to_tsquery('english', 'beautiful | place') AS rank
FROM test_run
WHERE a @@ to_tsquery('english', 'beautiful | place')
ORDER BY a <=> to_tsquery('english', 'beautiful | place');
```

t	rank
-----+-----	
It looks like a beautiful place	8.22467
The situation is most beautiful	16.4493
It is a beautiful	16.4493

(3 rows)

```
CREATE INDEX tst_idx ON tst USING rum (t rum_tsvector_addon_ops, d)
WITH (attach = 'd', to = 't');
```

Now we can execute the following queries:

```
EXPLAIN (costs off)
SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tst WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
QUERY PLAN
```

Limit

```
-> Index Scan using tst_idx on tst
    Index Cond: (t @@ ''wr'' & ''qh''::tsquery)
    Order By: (d <=> 'Mon May 16 14:21:25 2016'::timestamp without time zone)
```

(4 rows)

```
SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tst WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
```

id	d	?column?
355	Mon May 16 14:21:22.326724 2016	2.673276
354	Mon May 16 13:21:22.326724 2016	3602.673276
371	Tue May 17 06:21:22.326724 2016	57597.326724
406	Wed May 18 17:21:22.326724 2016	183597.326724
415	Thu May 19 02:21:22.326724 2016	215997.326724

(5 rows)

rum-全文检索+附加属性查询

```
CREATE TABLE tsts (id int, t tsvector, d timestamp);
```

```
\copy tsts from 'rum/data/tsts.data'
```

```
CREATE INDEX tsts_idx ON tsts USING rum (t rum_tsvector_addon_ops, d)  
WITH (attach = 'd', to = 't');
```

```
XPLAIN (costs off)
```

```
SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
```

```
QUERY PLAN
```

```
-----  
Limit
```

```
-> Index Scan using tsts_idx on tsts
```

```
Index Cond: (t @@ "'wr" & "qh"'::tsquery)
```

```
Order By: (d <=> 'Mon May 16 14:21:25 2016'::timestamp without time zone)
```

```
(4 rows)
```


rum-全文检索+附加属性查询

```
SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
```

```
id |          d          | ?column?
```

```
-----+-----+-----
```

```
355 | Mon May 16 14:21:22.326724 2016 |    2.673276
```

```
354 | Mon May 16 13:21:22.326724 2016 | 3602.673276
```

```
371 | Tue May 17 06:21:22.326724 2016 | 57597.326724
```

```
406 | Wed May 18 17:21:22.326724 2016 | 183597.326724
```

```
415 | Thu May 19 02:21:22.326724 2016 | 215997.326724
```

```
(5 rows)
```

目录

- 全文检索原理
- 自定义分词
- rank排序
- 关键词分析
- gin索引
- rum索引
- 全文检索+附加过滤
- 模糊查询
- 单字模糊查询
- 双字模糊查询
- wchar模糊查询注意事项
- 相似文本查询

前缀模糊查询背景技术

```
create extension pg_trgm;  
create index idx on tbl (col text_patten_ops);  
select * from tbl where {col ~ '^前缀' | like '前缀%'};
```

自动

```
postgres=# explain select * from pre where c1 like '你%';  
               QUERY PLAN
```

```
-----  
Index Scan using idx_pre on pre (cost=0.29..2.71 rows=1 width=21)  
  Index Cond: ((c1 ~>= ~ '你'::text) AND (c1 ~< ~ '你'::text))  
  Filter: (c1 ~~ '你%'::text)  
(3 rows)
```

```
postgres=# select chr(ascii('你')+1);  
chr  
-----  
  你  
(1 row)
```

后缀模糊查询背景技术

```
postgres=# create index idx_pre1 on pre (reverse(c1) text_pattern_ops);
```

```
CREATE INDEX
```

```
postgres=# select reverse('你好abcd ');
```

```
reverse
```

```
-----
```

```
dcba你好
```

```
(1 row)
```

```
postgres=# explain select * from pre where reverse(c1) like reverse('结尾')||'%';
```

```
QUERY PLAN
```

```
-----
```

```
Index Scan using idx_pre1 on pre (cost=0.29..45.93 rows=50 width=21)
```

```
Index Cond: ((reverse(c1) ~>= ~ '结尾'::text) AND (reverse(c1) ~<~ '尾'::text))
```

```
Filter: (reverse(c1) ~~ '结尾%'::text)
```

```
(3 rows)
```

前后模糊查询背景技术

```
postgres=# create index idx_pre2 on pre using gin (c1 gin_trgm_ops);
```

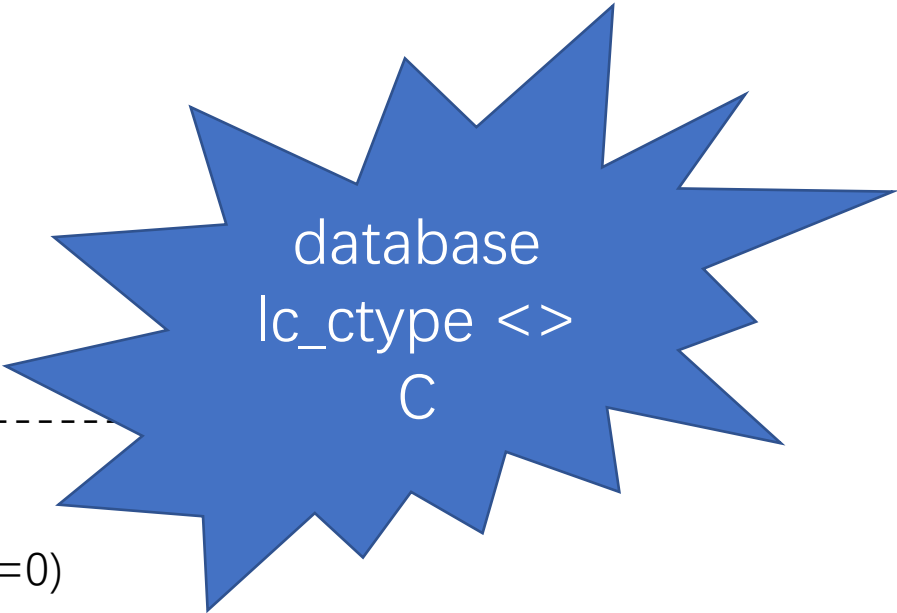
```
CREATE INDEX
```

```
postgres=# select show_trgm('abcde');
          show_trgm
```

```
-----
{" a"," ab",abc,bcd,cde,"de "}
(1 row)
```

```
postgres=# explain select * from pre where c1 like '%abc%';
          QUERY PLAN
```

```
-----
Bitmap Heap Scan on pre (cost=3.61..4.82 rows=1 width=21)
  Recheck Cond: (c1 ~~ '%abc% '::text)
  -> Bitmap Index Scan on idx_pre2 (cost=0.00..3.61 rows=1 width=0)
       Index Cond: (c1 ~~ '%abc% '::text)
(4 rows)
```



database
lc_ctype <>
C

前后模糊查询背景技术

- 当前后模糊查询低于3个字符时，需要使用表达式索引, split
 - https://github.com/digoal/blog/blob/master/201704/20170426_01.md

支持中文的前后模糊查询背景技术

- https://github.com/digoal/blog/blob/master/201605/20160506_02.md

Command: CREATE DATABASE

Description: create a new database

Syntax:

CREATE DATABASE name

[[WITH] [OWNER [=] user_name]

[TEMPLATE [=] template]

[ENCODING [=] encoding]

[LC_COLLATE [=] lc_collate]

[LC_CTYPE [=] lc_ctype] **不能=C (=c时,pg_trgm无法切分wchar)**

[TABLESPACE [=] tablespace_name]

[ALLOW_CONNECTIONS [=] allowconn]

[CONNECTION LIMIT [=] conlimit]

[IS_TEMPLATE [=] istemplate]]

目录

- 全文检索原理
- 自定义分词
- rank排序
- 关键词分析
- gin索引
- rum索引
- 全文检索+附加过滤
- 模糊查询
- 单字模糊查询
- 双字模糊查询
- wchar模糊查询注意事项
- 相似文本查询

相似算法

- pg_trgm , token 切词
- token 相交率
- https://github.com/digoal/blog/blob/master/201802/20180202_01.md

Key	Heap ID	Page	1	35	44	101	109	1000
1			1	0	0	:	1	0	0
3			0	0	1	:	0	1	1
101			0	0	1	:	1	0	0
198			0	1	0	:	1	1	0
792			1	0	0	:	1	0	1
1000			0	0	1	:	0	0	0
			2	1	3	...	4	2	2

性能指标：

CASE	单次相似搜索响应速度
10亿行，每行64个随机中文	40毫秒

相似度算法

4、通过 `similarity` 或 `word_similarity` 可以查看两个字符串的相似度值。

```
postgres=# select similarity('abc','abcd');
similarity
```

```
-----
          0.5
(1 row)
```

```
postgres=# select word_similarity('abc','abcd');
word_similarity
```

```
-----
          0.75
(1 row)
```

```
postgres=# select word_similarity('abc','abc');
word_similarity
```

```
-----
          1
(1 row)
```

```
postgres=# select similarity('abc','abc');
similarity
```

```
-----
          1
(1 row)
```

相似算法详情请参考

<https://www.postgresql.org/docs/devel/static/pgtrgm.html>

相似查询

```
create or replace function get_res(  
    text,    -- 要按相似搜的文本  
    int8,    -- 限制返回多少条  
    float4 default 0.3, -- 相似度阈值, 低于这个值不再搜搜  
    float4 default 0.1  -- 相似度递减步长, 直至阈值  
) returns setof record as $$  
declare  
    lim float4 := 1;  
begin  
    -- 判定  
    if not ($3 <= 1 and $3 > 0) then  
        raise notice '$3 must >0 and <=1';  
        return;  
    end if;  
    if not ($4 > 0 and $4 < 1) then  
        raise notice '$4 must >0 and <=1';  
        return;  
    end if;
```

相似查询

loop

-- 设置相似度阈值

perform set_limit(lim);

return query select similarity(info, \$1) as sml, * from tbl where info % \$1 order by sml desc limit \$2;

-- 如果有，则退出loop

if found then

return;

end if;

-- 否则继续，降低阈值

-- 当阈值小于0.3时，不再降阈值搜索，认为没有相似。

if lim < \$3 then

return;

else

lim := lim - \$4;

end if;

end loop;

end;

\$\$ language plpgsql strict;

相似查询

```
select * from get_res(  
    '输入搜索文本',  
    输入限制条数,  
    输入阈值,  
    输入步长  
) as t(sml float4, id int, info text);
```

相似查询

```
postgres=# select * from get_res('拏掇贼展跃髻峪四餽麾鄣賃青乖湔鯨掄搵垓屹操犝淒銳約韞夊缝特鏽邕鯪垓縛墻軌禮倚亦猗厝醢恠  
嚙銭翺勑嘹雍岬擦寵沚蒸彼鴉縻嫫妝倨丿淝', 10, 0.4, 0.05) as t(sml float4, id int, info text);
```

-[RECORD 1]-----

sml	0.882353
id	1
info	佛拏掇贼展跃髻峪四餽麾鄣賃青乖湔鯨掄搵垓屹操犝淒銳約韞夊缝特鏽邕鯪垓縛墻軌禮倚亦猗厝醢恠嚙銭翺勑嘹雍岬擦寵沚蒸 彼鴉縻嫫妝倨丿淝鵒

Time: 52.852 ms

参考资料

- 案例
- MySQL手册
 - <https://www.mysqltutorial.org/>
 - <https://dev.mysql.com/doc/refman/8.0/en/>
- PG 管理、开发规范
 - https://github.com/digoal/blog/blob/master/201609/20160926_01.md
- PG手册
 - <https://www.postgresql.org/docs/current/index.html>
 - <https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-vs-mysql/>
- GIS手册
 - <http://postgis.net/docs/manual-3.0/>

一期开课计划(PG+MySQL联合方案)

- - 2019.12.30 19:30 RDS PG产品概览，如何与MySQL结合使用
- - 2019.12.31 19:30 如何连接PG， GUI， CLI的使用
- - 2020.1.3 19:30 如何压测PG数据库、如何瞬间构造海量测试数据
- - 2020.1.6 19:30 MySQL与PG对比学习(面向开发者)
- - 2020.1.7 19:30 如何将MySQL数据同步到PG (DTS)
- - 2020.1.8 19:30 PG外部表妙用 - mysql_fdw, oss_fdw (直接读写MySQL数据、冷热分离)
- - 2020.1.9 19:30 PG应用场景介绍 - 并行计算， 实时分析
- - 2020.1.10 19:30 PG应用场景介绍 - GIS
- - 2020.1.13 19:30 PG应用场景介绍 - 用户画像、实时营销系统
- - 2020.1.14 19:30 PG应用场景介绍 - 多维搜索
- - 2020.1.15 19:30 PG应用场景介绍 - 向量计算、图像搜索
- - 2020.1.16 19:30 PG应用场景介绍 - 全文检索、模糊查询
- - 2020.1.17 19:30 PG 数据分析语法介绍
- - 2020.1.18 19:30 PG 更多功能了解：扩展语法、索引、类型、存储过程与函数。如何加入PG技术社群

本课程习题

- 中文分词的常用插件有哪些？
- 全文检索使用什么索引？
- 全文检索支持自定义分词吗？
- 全文检索支持哪些搜索方法？
- 为什么有些数据库无法支持wchar模糊查询加速，lc_ctype应该如何设置？
- prefix%的like查询使用什么索引？
- %suffix的like查询使用什么索引？
- %xxx%前后模糊查询使用什么索引？
- rum索引和gin索引的重要差别有哪些？
- 少于3个字符的%xxx%前后模糊查询使用什么索引？
- 相似文本查询用什么方法？

技术社群



PG技术交流钉钉群(3600+人)

