Atitit 全文检索 pgsql 艾提拉总结

# **分词**

全文索引的实现要靠 PgSQL 的 gin 索引。分词功能 PgSQL 内置了英文、西班牙文等，但中文分词需要借助开源插件 zhparser；

### **SQL**

这样，再进行查询时就可以直接使用 SELECT \* FROM table WHERE tsv\_column @@ 'keyword' 了

查询中我们可以使用最简单的 SELECT \* FROM table WHERE to\_tsvector('parser\_name', field) @@ 'word' 来查询 field 字段分词中带有 word 一词的数据；

使用 to\_tsquery() 方法将句子解析成各个词的组合向量，如 国家大剧院 的返回结果为 '国家' & '大剧院' & '大剧' & '剧院' ，当然我们也可以使用 & | 符号拼接自己需要的向量；在查询 长句 时，可以使用 SELECT \* FROM table WHERE to\_tsvector('parser\_name', field) @@ to\_tsquery('parser\_name','words')；

有时候我们想像 MySQL 的 SQL\_CALC\_FOUND\_ROWS 语句一样同步返回结果条数，则可以使用 SELECT COUNT(\*) OVER() AS score FROM table WHERE ...，PgSQL 会在每一行数据添加 score 字段存储查询到的总结果条数；

到这里，普通的全文检索需求已经实现了。

# **总结**

简单的数据迁移并不是终点，后续要做的还有很多，如整个系统的数据同步、查询效率优化、查询功能优化（添加拼音搜索、模糊搜索）等。特别是查询效率，不知道是不是我配置有问题，完全达不到那种 E级毫秒 的速度，1kw 的数据效率在进行大结果返回时就大幅下降（200ms），只好老老实实地提前进行了分表，目前百万级查询速度在 20ms 以内，优化还有一段路要走。

不过这次倒是对 技术的“生态”有了个更深的体会，这方面 PgSQL 确实和 MySQL 差远了，使用 MySQL 时再奇葩的问题都能在网上快速找到答案，而 PgSQL 就尴尬了，入门级的问题搜索 stackoverflow 来来回回就那么几个对不上的回答。虽然也有阿里的“德哥”一样的大神在辛苦布道，但用户的数量才是根本。不过，随着 PgSQL 越来越完善，使用它的人一定会越来越多的，我这篇文章也算是为 PgSQL 加温了吧，哈哈~希望能帮到后来的使用者。

关于本文有什么问题可以在下面留言交流，如果您觉得本文对您有帮助，可

们为该表的的name字段创建一个全文索引，并将它的信息独立的存储在一个表中，表如下：

CREATE TABLE full\_text\_index    
(    
<pre name="code" class="sql"> id bigserial NOT NULL,    
CREATE TABLE full\_text\_index  
(  
<pre name="code" class="sql"> id bigserial NOT NULL, text\_fti tsvector NOT NULL,)WITH ( OIDS=FALSE);ALTER TABLE full\_text\_index OWNER TO postgres;  
）;

我们注意看字段text\_fti，它的类型是tsvector，说明该字段是存储全文索引的，这就是postgresql数据库中全文索引的字段类型。

那么如何将我们的信息自动的创建全文索引信息添加到我们的full\_text\_index 中呢？那么这里将要用到sql中的触发器，

如下创建触发器函数：

1. **CREATE** OR REPLACE **FUNCTION** node\_text\_fti()
2. **RETURNS** **trigger** **AS**
3. $BODY$**begin**
4. if TG\_OP = 'INSERT' **then**
5. **insert** **into** full\_text\_index (text\_fti) **values**(to\_tsvector(coalesce(NEW.**name**,'')));
6. **end** if;
7. **return** NEW;**end**;$BODY$
8. LANGUAGE plpgsql VOLATILE
9. COST 100;
10. **ALTER** **FUNCTION**

**select** \* **from** test **where** content\_fti @@ to\_tsquery('keyword');

# 第1章    全文检索语法

## 1.1 概述

查询引擎为文本数据类型提供~, ~\*, LIKE和ILIKE操作符，并提供全文检索以识别自然语言文档，并通过相关性查询进行排序。查询引擎提供两种数据类型用于支持全文检索，即tsvector类型与tsquery类型。

## 1.2 文档(tsvector)**类型**

对于tsvector类型，表示一个检索单元，通常是一个数据库表中一行的文本字段，或者这些字段的可能组合（级联），也可能存储在多个表中或者动态地获得，它的值是一个无重复值的lexemes排序列表，即一些同一个词的不同变种的标准化，在输入的同时会自动排序和消除重复。to\_tsvector函数通常用于解析和标准化文档字符串。

一个tsvector的值是唯一分词的分类列表，把一话一句词格式化为不同的词条，在进行分词处理的时候tsvector会自动去掉分词中重复的词条，按照一定的

## 1.3查询(tsquery)**类型**

对于tsquery类型，表示一个检索条件，存储用于检索的词汇，并且使用布尔操作符&(AND)，|(OR)和!(NOT) 来组合它们，括号用来强调操作符的分组。与tsvector一样，任何单词必须在转换为tsquery类型前规范化。to\_tsquery函数及plainto\_tsquery函数可以方便的用来执行规范化。

SELECT 'fat & rat'::tsquery;

 tsquery

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

 'fat' & 'rat'

SELECT 'fat &

## 1.4检索表

查询引擎的全文检索基于匹配算子@@，如果一个tsvector与一个tsquery匹配，则返回true。在不使用索引的情况下也是可以进行全文检索的,一个简单查询,显示出title从所有body字段中包含friend的每一行：

SELECT title

FROM web

WHERE to\_tsvector('english', body) @@ to\_tsquery('english','friend');

其中to\_tsvector和to\_tsquery中第一个参数用于指定全文检索的分词语言设置，一般可省略，语句如下：

SELECT title

FROM web

WHERE to\_tsvector(body) @@ to\_tsquery('friend');

## 1.5创建索引

gist 和 gin的索引类型，这两种索引都能用在提高全文检索的速度，注意全文检索不一定非要使用索引，但是当一个字段被固定规律搜索时，使用索引将会有很好的效果。创建gist 索引字段的类型可以是 tsvector 或者 tsquery。创建gin 索引字段的类型必须是tsvector

在PostgreSQL中使用全文索引介绍 第2页\_Linux教程\_Linux公社-Linux系统门户网站.html

postgresql全文检索语法 - CSDN博客.html

让我们尝试把一个字符串转换为一个tsvector。

[?](http://www.jb51.net/article/64544.htm)

|  |  |
| --- | --- |
| 1 | SELECT to\_tsvector('Try not to become a man of success, but rather try to |

发生了怪事。首先比原文的词少了，一些词也变了（try变成了tri），而且后面还有数字。怎么回事？

一个tsvector是一个标准词位的有序列表（sorted list），标准词位（distinct lexeme）就是说把同一单词的各种变型体都被标准化相同的。

标准化过程几乎总是把大写字母换成小写的，也经常移除后缀（比如英语中的s,es和ing等）。这样可以搜索同一个字的各种变体，而不是乏味地输入所有可能的变体。

数字表示词位在原始字符串中的位置，比如“man"出现在第6和15的位置上。你可以自己数数看。

Postgres中to\_tesvetor的默认配置的文本搜索是“英语“。它会忽略掉英语中的停用词（stopword，译注：也就是am is are a an等单词)。

这解释了为什么tsvetor的结果比原句子中的单词少。后面我们会看到更多的语言和文本搜索配置。

**查询**

我们知道了如何构建一个文档，但我们的目标是搜索文档。我们对tsvector搜索时可以使用@@操作符，使用说明见此处。看几个查询文档的例子。

[?](http://www.jb51.net/article/64544.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | > select to\_tsvector('If you can dream it, you can do it') @@ 'dream'; |

select \* FROM datadic WHERE to\_tsvector(val)@@to\_tsquery('字段')='t'

第二个查询返回了假，因为我们需要构建一个tsquery，使用@@操作符时，把字符串转型(cast)成了tsquery。下面显示了这种l转型和使用to\_tsquery()之间的差别。

[?](http://www.jb51.net/article/64544.htm)

|  |  |
| --- | --- |
| 1  2  3  4 | SELECT 'impossible'::tsquery, to\_tsquery('impossible'); |
| SELECT 'impossible'::tsquery, to\_tsquery('impossible');   tsquery | to\_tsquery  --------------+------------   'impossible' | 'imposs'(1 row) | |

但"dream"的词位与它本身相同。

中文的全文检索 不需要tsquery ，那个是构造词根的。。

select to\_tsvector( '排序，请 注意，如果 一个 字段 没有 被 索引，那么 它是 无法 排序 的') @@ '字段' as col

返回col》》t

select \* FROM datadic WHERE to\_tsvector(val)@@'排序'='t'

默认varcha类型

使用like和@@语法比较，，@@更慢 0.6s ,like只要0.2s 可能是默认没有索引导致的

select \* from vaj1 where vaj57 like '%玻璃%' limit 100000

select \* from vaj1 where to\_tsvector(vaj57)@@'玻璃'='t' limit 100000

## **to\_tsquery**

to\_tsquery(text)

这个函数作用是用来把文本转化为可查询的语句，例如：

**SELECT** to\_tsquery('我爱&文因互联')

这样就是寻找文本既包含“我爱”，也包含“文因互联”的语句，&是AND与操作，|是OR或操作

如果想知道一个tsvector是否让tsquery成立，可以使用

## **to\_tsvector**

to\_tsvector(text， text)

这个函数可以把一个文本转换为一个向量，其中保存单词和其出现的顺序，下面是官方的示例，可以看出其中保存的是英语单词及其位置，例如fat出现了两次，分别在2位置和11位置

**SELECT** to\_tsvector('english', 'a fat cat sat on a mat - it ate a fat rats');

to\_tsvector*-----------------------------------------------------*

'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4

其中第一个参数是语言，第二个参数是需要转换的文本，当然了，至少官方是不支持中文的。第一个参数也可以省略，默认使用english。

如果想知道一个tsvector是否让tsquery成立，可以使用

**@@**

操作符，例如：

**SELECT**to\_tsvector('simple', '我爱 文因互联') **@@** to\_tsquery('我爱&文因互联')

## **索引**

最后为了提高查询速度， 还需要建索引，PotsgreSQL提供了一个叫做pg\_trgm的扩展：

**CREATE** **EXTENSION** **IF** **NOT** **EXISTS** pg\_trgm;

这个扩展引入了两个索引类型，一个是gin，一个是gist

这两个选择的不同从使用上主要是gist构建速度比较快，搜索速度比gin慢，而gin相反，搜索速度很快，构建速度非常慢，具体可以参考[3]

数据源是text类型或者varchar类型，索引是tsvector类型。

然后采用是独立索引表或者增加索引字段方式，不管哪种方式，都需要创建触发器。

对于大批量的数据入库的话，可能对写操作速度有些影响

索引表(独立的方式)记录如图：

GiST就是通用搜索树。

谈谈一个初步的体验

数据源是text类型或者varchar类型，索引是tsvector类型。

然后采用是独立索引表或者增加索引字段方式，不管哪种方式，都需要创建触发器。

对于大批量的数据入库的话，可能对写操作速度有些影响

索引表(独立的方式)记录如图：

另外一种是GIN，叫通用倒排索引，是一个存储对(key,posting list)集合的索引结构，Key是一个键值，而posting list是一组出现过"key“的位置。  
据说在全文检索时，采用这种方式比GIST更好，参考内容：  
GIN index lookups are about three times faster than GiST.  
GIN indexes take about three times longer to build than GiST.  
GIN indexes are about ten times slower to update than GiST.  
GIN indexes are two-to-three times larger than GiST.  
  
后面是会对GIN的使用再谈下感受。

给 varchar 创建gist所哟ishiy

CREATE INDEX "vaj1\_vaj57\_index" ON "public"."vaj1" USING gist ("vaj57" gist\_trgm\_ops );

关于PostgreSQL的GiST索引之五-skykiker-ChinaUnix博客.html

CREATE INDEX "vaj1\_vaj57\_index" ON "public"."vaj1" USING gist ("vaj57" gist\_trgm\_ops );

CREATE EXTENSION IF NOT EXISTS pg\_trgm;

建立了gist所哟ishiy貌似么有效果

select \* from vaj1 where to\_tsvector(vaj57)@@'玻璃'='t' limit 100000

单独建立 索引表vaj1\_ori limit 200

select \* from vaj1 where to\_tsvector(vaj57)@@'玻璃'='t' limit 100000

select vaj57 from vaj1

insert into vaj1\_ori select to\_tsvector(vaj57) from vaj1

select \* from vaj1\_ori limit 200

select \* from vaj1\_ori where vaj57@@'玻璃'='t' limit 100000

默认查询已经是0.12了，建立gist索引，就到0.05s了。。落哦 direcgt build gist index on col type tsvecto...

If varchat type ,build gist index ,need gist\_trgm\_ops

和mysql全文索引的区别

Mysql 索引全自动，，pgsql的需要单独建立缩影表以及触发器维护缩影。。

Mysql使用更加简单。。