EDB

Full Text Search Nativo no PostgreSQL

Juliano Atanazio

FTS: O que é? Para que serve?

Full Text Search / **Text Search** / **Busca Textual**: é um mecanismo que possibilita identificar documentos em linguagem natural que casem (combinem) com uma consulta e opcionalmente ordená-los por relevância.

O uso mais comum de busca é encontrar todos documentos que contém termos de uma consulta e retorná-los conforme sua semelhança com essa consulta.

Operadores ~, ~*, LIKE, and ILIKE

Por que NÃO?

- Não há suporte linguístico (mesmo para inglês);
- Expressões regulares não são suficientes, pois é difícil lidar com derivações de palavras;
- Não tem ordem (relevância) de resultados de busca, o que os faz ineficazes quando milhares de documentos que casem são encontrados;
- Tendência a lentidão, por não haver suporte nativo a índice, então eles têm que processar todos documentos por cada busca.

Termos: Documento

É a unidade de busca em um sistema de busca textual, por exemplo: um artigo de revista ou uma mensagem de e-mail.

Para buscas no PostgreSQL, um documento é normalmente um campo de texto em uma linha de uma tabela, ou possivelmente uma combinação (concatenação) de campos.

Resumindo: **documento** em buscas textuais é um **texto**.

Base de dados de exemplo db_fts

Foi criado exclusivamente para esta apresentação e possui apenas uma tabela, a `tb_post`, cuja estrutura é:

Coluna	**Tipo**
id	integer
title	character varying(100)
text_	text
tags	character varying(200)
Índice:	"pk_post" PRIMARY KEY, btree (id)

A tabela tem como função armazenar textos (artigos), cujos campos são `id`, `title`, `text_` e `tags`.
Um índice, `pk_post`, para o campo `id`.

Termos: Documento

```
SELECT
    title,
    substring(text_, 1, 70) AS text_
FROM tb_post
WHERE id = 1;
```

```
   title    |                                   text_
------------+-------------------------------------------------------------------
 Laranjeira | A laranja é o fruto produzido pela laranjeira (Citrus × sinensis), uma
```

A partir da consulta extraiu-se o documento.

Termos: Token

Tokens são as palavras em um documento, que podem variar seus tipos como números, palavras, palavras complexas e endereços de e-mail.

```
SELECT
    'O rato foi pego pela gata preta, pelo gato branco e pelo '
    ||'cachorro de pêlos lisos'::tsvector;
```

```
                                                                                                  ?column?
-------------------------------------------------------------------------------------------------------------------------------------------
'O' 'branco' 'cachorro' 'de' 'e' 'foi' 'gata' 'gato' 'lisos' 'pego' 'pela' 'pelo' 'preta,' 'pêlos' 'rato'
```

Termos: Lexema

Palavra ou parte da palavra que serve de base ao sentido por ela expresso, ou seja, é o radical que pode ser modificado por afixos, como prefixos e sufixos, para formar novas palavras.

Exemplos de palavras em que é possível identificar o lexema:

Palavra: “amoroso”

Lexema: “amor”

Afixo: “-oso”

Palavra: “infeliz”

Lexema: “feliz”

Prefixo: “in-”

Termos: Stem

Stem é uma representação simplificada de uma palavra após um *stemming*.

Esse processo consiste em reduzir uma palavra ao seu radical.

Exemplos:

A palavra “meninas” se reduziria a “menin”, assim como “meninos” e “menininhos”.

As palavras “gato”, “gata”, “gatos” e “gatas” reduziriam-se para “gat”.

Termos: Stem

```
SELECT to_tsvector('Portuguese', 'gatas');

 to_tsvector
-------------
 'gat':1
```

```
SELECT to_tsvector('pg_catalog.portuguese', 'gato');

 to_tsvector
-------------
 'gat':1
```

Termos: Stop words

Em português, palavras de parada, são palavras que podem ser consideradas irrelevantes para o conjunto de resultados a ser exibido em uma busca realizada em um *search engine*.

Artigos: o, a, os, as, um, uma, uns, umas

Conjunções: e, nem, mas, já, mas, ou, que, se

Crase: à

Preposições: a, com, como, de, em, para, por, sem

Pronomes: eu, nós, tu, vós, ele, eles, ela, elas, seu, teu, meu, no, na, nos, nas, quem, que, qual

Termos: Stop words

O postgres lê o arquivo de palavras de parada que está em no diretório *`$SHAREDIR/tsearch_data/`*.

A extensão desse arquivo é `.stop`.

Seu conteúdo é de apenas *stop words*.

`$SHAREDIR` pode variar dependendo da instalação do PostgreSQL. Por padrão é o diretório `share` da instalação.

Termos: Stop words

Para o idioma português o arquivo é

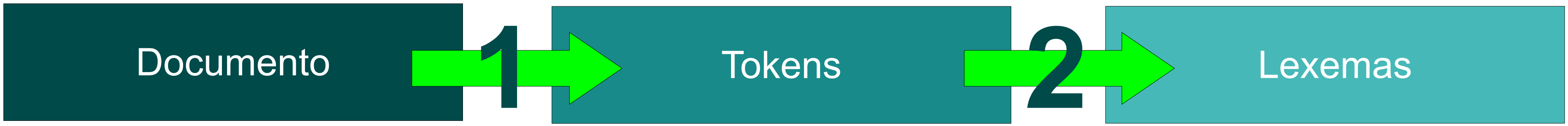
`$SHAREDIR/tsearch_data/portuguese.stop`

Para obter o valor de `$SHAREDIR`, dê o comando no *shell*:

```
$ pg_config --sharedir
```

/usr/share/postgresql/16

Pré processamento




```
SELECT
    'O rato foi pego pela gata preta, pelo gato branco e pelo '
    ||'cachorro de pêlos lisos'::tsvector;

                                  ?column?
-------------------------------------------------------------------------
 'O' 'branco' 'cachorro' 'de' 'e' 'foi' 'gata' 'gato' 'lisos' 'pego' 'pela' 'pelo' 'preta,' 'pêlos' 'rato'
```

```
SELECT
    to_tsvector(
        'Portuguese',
        'O rato foi pego pela gata preta, pelo gato branco e '
        ||'pelo cachorro de pêlos liso');

                         to_tsvector
---------------------------------------------------------------
 'branc':10 'cachorr':13 'gat':6,9 'lis':16 'peg':4 'pret':7 'pêl':15 'rat':2
```

Nota-se no segundo exemplo que não há *stop words*.

Configurações de FTS

Funcionalidades de busca textual são controladas por configurações próprias.

O PostgreSQL vem com configurações pré definidas para muitos idiomas.

Especifica todas opções necessárias para transformar um documento em um *tsvector*: o *parser* para quebrar texto em *tokens* e os dicionários para transformar cada *token* em um lexema.

Toda chamada a funções como `to_tsvector` e `to_tsquery` necessita de uma configuração para fazer seu processamento.

O parâmetro de configuração `default_text_search_config` especifca o nome da configuração padrão, que será é utilizada por funções de buscas textuais se o padrão relativo à configuração de busca textual for omitido.

Configurações de FTS

As configurações disponíveis podem ser listadas no `psql` da seguinte forma:

```
\dF
```

```
                  List of text search configurations
   Schema   |    Name    |                  Description
------------+------------+---------------------------------------
 pg_catalog | arabic     | configuration for arabic language
 pg_catalog | armenian   | configuration for armenian language
 pg_catalog | basque     | configuration for basque language
 . . .
```

Ou via SQL:

```
SELECT cfgname FROM pg_ts_config;
```

```
  cfgname
------------
 simple
 arabic
 armenian
. . .
```

Configurações de FTS

O parâmetro `default_text_search_config` é o que controla como FTS irá funcionar no PostgreSQL.

Exibindo:

```
SHOW default_text_search_config;
```

```
 default_text_search_config
----------------------------
 pg_catalog.english
```

Alterando para a sessão atual:

```
SET default_text_search_config = 'pg_catalog.portuguese';
```

ou

```
SET default_text_search_config = 'Portuguese';
```

Configurações de FTS

Essa configuração pode ser configurada no `postgresql.conf` ou em uma sessão individual utilizando o comando `SET`.

Muitas configurações de busca textual pré definidas estão disponíveis, e é possível criar uma customizada facilmente.

Para facilitar o gerenciamento de objetos de busca textual, um conjunto de comandos SQL está disponível e e há vários comandos do `psql` que exibem informações sobre objetos de busca textual.

Configurações de FTS

Criação de uma configuração chamada tsc_pg a partir da configuração português *built-in*:

```
CREATE TEXT SEARCH CONFIGURATION public.tsc_pg
     (COPY = pg_catalog.portuguese);
```

Criação de um arquivo de lista de sinônimos (`$SHAREDIR/tsearch_data/pg_dict.syn`):

```
# cat << EOF > `pg_config --sharedir`/tsearch_data/pg_dict.syn
postgres    pg
pgsql       pg
postgresql  pg
EOF
```

Configurações de FTS

Instalação de pacotes que proverão os arquivos necessários para o português brasileiro:

```
# apt install -y myspell-pt-br hunspell-pt-br
```

Localizando os arquivos .aff e .dic, que são necessários para gerar os arquivos `.affix` e `.dict`, respectivamente:

```
$ dpkg -S pt_BR.aff pt_BR.dic

hunspell-pt-br: /usr/share/hunspell/pt_BR.aff
hunspell-pt-br: /usr/share/hunspell/pt_BR.dic
```

Criando os arquivos `.dict` e `.affix`, com codificação UTF-8:

```
# cp /usr/share/hunspell/pt_BR.dic $SHAREDIR/tsearch_data/pt_br.dict

# sed 's/FLAG UTF-8/FLAG default/g' \
    /usr/share/hunspell/pt_BR.aff > $SHAREDIR/tsearch_data/pt_br.affix
```

Configurações de FTS

Definição do dicionário de sinônimos:

```
CREATE TEXT SEARCH DICTIONARY dic_pg (
    TEMPLATE = synonym,
    SYNONYMS = pg_dict
);
```

Agora registraremos o dicionário Ispell `portuguese_ispell`, que tem seus próprios arquivos de configuração:

```
CREATE TEXT SEARCH DICTIONARY dic_portuguese_ispell (
    TEMPLATE = ispell,
    DictFile = pt_br,
    AffFile = pt_br,
    StopWords = portuguese
);
```

Configurações de FTS

Configuração dos mapeamentos de palavras na configuração `tsc_pg`:

```
ALTER TEXT SEARCH CONFIGURATION tsc_pg
    ALTER MAPPING FOR asciiword,
                      asciihword,
                      hword_asciipart,
                      word,
                      hword,
                      hword_part
    WITH dic_pg,
       dic_portuguese_ispell,
       portuguese_stem;
```

Configurações de FTS

```
SELECT ts_debug ('public.tsc_pg','postgresql');
```

```
                                                                                                                ts_debug
-------------------------------------------------------------------------------------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",postgresql,"{dic_pg,dic_portuguese_ispell,portuguese_stem}",dic_pg,{pg})
```

```
SELECT ts_debug ('public.tsc_pg','pgsql');
```

```
                                                                                                                ts_debug
-------------------------------------------------------------------------------------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",pgsql,"{dic_pg,dic_portuguese_ispell,portuguese_stem}",dic_pg,{pg})
```

```
SELECT ts_debug ('public.tsc_pg','postgres');
```

```
                                                                                                                ts_debug
-------------------------------------------------------------------------------------------------------------------------------------------------------
 (asciiword,"Word, all ASCII",postgres,"{dic_pg,dic_portuguese_ispell,portuguese_stem}",dic_pg,{pg})
```

Configurações de FTS

Configurando a sessão para usar a nova configuração, que foi criada no schema public:

```
SET default_text_search_config = 'public.tsc_pg';
```

Verificando o atual valor:

```
SHOW default_text_search_config;
```

```
 default_text_search_config
----------------------------
 public.tsc_pg
```

Tipos e funções: tipo tsvector

- Vetor de lexemas e posições;
- Representa um documento em uma forma otimizada para buscas de texto;
- Seu valor é uma lista ordenada de lexemas distintos;
- Ordena, elimina duplicações automaticamente;
- Representa um documento de forma compacta;

```
SELECT
    'O carro correu, correu e venceu a corrida facilmente, '
    ||'ultrapassando o último carro a 15 min do final'::tsvector;

                                                      ?column?
------------------------------------------------------------------------------------------------------------------------------------
 '15' 'O' 'a' 'carro' 'correu' 'correu,' 'corrida' 'do' 'e' 'facilmente,' 'final' 'min' 'o' 'ultrapassando' 'venceu' 'último'
```

Tipos e funções: tipo tsvector

Opcionalmente, posições (em números inteiros) podem ser anexadas aos lexemas:

```
SELECT
     'o:1 carro:2 verde:3 correu:4 mais:5 do:6 que:7 o:8 carro:9'
     ||'vermelho:10'::tsvector;
```

```
                                    ?column?
-------------------------------------------------------------------------------
 'carro':2,9 'correu':4 'do':6 'mais':5 'o':1,8 'que':7 'verde':3 'vermelho':19
```

Em caso de duplicações as posições são delimitadas por vírgulas.

Tipos e funções: função to_tsvector

A função `to_tsvector` converte um documento para *tokens*, reduz esses *tokens* para lexemas e retorna um tsvector que lista os lexemas junto com suas posições no documento sem *stop words*.

Faz também stemming dos lexemas.

```
SELECT
    to_tsvector(
        'pg_catalog.portuguese',
        'O carro correu, correu e venceu a corrida facilmente, '
        ||'ultrapassando o último carro a 15 min do final');
```

```
                                                to_tsvector
-------------------------------------------------------------------------------------------------------
 '15':15 'carr':2,13 'corr':3,4,8 'facil':9 'final':18 'min':16 'ultrapass':10 'venc':6 'últim':12
```

Tipos e funções: função to_tsvector

Não confundir a função `to_tsvector` com um valor de uma *string* convertida para `tsvector`.

```
SELECT 'rato gato pato'::tsvector;
```

```
       tsvector
----------------------
 'gato' 'pato' 'rato'
```

```
SELECT to_tsvector('Portuguese', 'rato gato pato');
```

```
      to_tsvector
------------------------
 'gat':2 'pat':3 'rat':1
```

Tipos e funções: tipo tsquery

- Consulta de busca textual;
- Representa um texto de consulta;
- Armazena lexemas que serão base de busca;
- Combinação de lexemas utilizando operadores booleanos:

  & (AND), | (OR) e ! (NOT);

- Parênteses podem ser usados para reforçar um grupamento de operadores.

Tipos e funções: tipo tsquery

```
SELECT 'gato & rato'::tsquery;

     tsquery
-----------------
 'gato' & 'rato'


SELECT 'pato & (gato|rato)'::tsquery;

           tsquery
-----------------------------
 'pato' & ( 'gato' | 'rato' )


SELECT '! 1'::tsquery;

 tsquery
---------
 !'1'
```

Tipos e funções: função to_tsquery

Cria um valor `tsquery` de um *querytext*.

Normaliza cada *token* em um lexema usando a configuração especificada ou por omissão.

Similar à função `to_tsvector`, em `to_tsquery`, podemos determinar o idioma e normalizar os termos passados para seus radicais.

Reduz lexemas a *stems*.

```
SELECT to_tsquery('pg_catalog.portuguese', '!(correr & vencer)');
```

```
      to_tsquery
----------------------
 !( 'corr' & 'venc' )
```

```
SELECT to_tsquery('pg_catalog.english', '!(running & winning)');
```

```
     to_tsquery
--------------------
 !( 'run' & 'win' )
```

Tipos e funções: função plainto_tsquery

Transforma texto sem formação *querytext* para *tsquery*;

O campo `text_` é analisado e normalizado como se fosse um `tsvector` e então o operador booleano **& (AND)** é inserido entre as palavras sobreviventes.

```
SELECT
    plainto_tsquery(
        'english',
        'The Fat Rats are sleeping with the cats');

         plainto_tsquery
--------------------------------
 'fat' & 'rat' & 'sleep' & 'cat'
```

Operadores; combina?: @@ ou @@@

Operações possíveis:

- `tsvector @@ tsquery`
- `tsquery @@ tsvector`
- `text @@ tsquery` → equivalente a to_tsvector(x) @@ y
- `text @@ text` → equivalente a `to_tsvector(x) @@ plainto_tsquery(y)`

Operadores; combina?: @@ ou @@@

`tsvector` casa (combina) com `tsquery`? (ou vice-versa)

```
SELECT
    to_tsquery('portuguese', '!(gato & rato)') @@
    to_tsvector('portuguese', 'O gato correu atrás do rato');
```

```
 ?column?
----------
 f
```

```
SELECT
   to_tsquery('portuguese', '(gato & rato)')
   @@ to_tsvector('portuguese', 'O gato correu atrás do rato');
```

```
 ?column?
----------
 t
```

```
SELECT
    to_tsquery('portuguese', '(gato & navio)') @@
    to_tsvector('portuguese', 'O gato correu atrás do rato');
```

```
 ?column?
----------
 f
```

Operadores; combina?: @@ ou @@@

```
SELECT
    to_tsquery('portuguese', '(gato | navio)') @@
    to_tsvector('portuguese', 'O gato correu atrás do rato');

 ?column?
----------
 t
```

```
SELECT
    to_tsquery('portuguese', 'gato | rato') @@
    to_tsvector('portuguese', 'rata');

 ?column?
----------
 t
```

Operadores; negação de uma tsquery: !!

```
SELECT
    !! to_tsquery('portuguese', 'gato | rato')
    @@ to_tsvector('portuguese', 'gatas');

 ?column?
----------
 f
```

```
SELECT
    !! to_tsquery('portuguese', 'gato | rato')
    @@ to_tsvector('portuguese', 'pato');

 ?column?
----------
 t
```

Operadores; contém: @>

```
SELECT
    to_tsquery('portuguese', 'gato | rato | cão')
    @> to_tsquery('portuguese', 'cão & rata');

 ?column?
----------
 t
```

```
SELECT
    to_tsquery('portuguese', 'gato | roda')
    @> to_tsquery('portuguese', 'navio & macaco');

 ?column?
----------
 f
```

Operadores; contido: <@

```
SELECT
    to_tsquery('portuguese', 'moto & carro')
    <@ to_tsquery('portuguese', 'moto | carro | ônibus');

 ?column?
----------
 t
```

```
SELECT
    to_tsquery('portuguese', 'moto | carro | ônibus')
    <@ to_tsquery('portuguese', 'moto & carro');

 ?column?
----------
 f
```

Consultas de busca textual

Exemplo de consulta envolvendo concatenação de campos de texto:

```
SELECT
    title
FROM tb_post
WHERE to_tsvector(
        'portuguese',
        title||' '||text_||' '||tags)
    @@ to_tsquery(
        'portuguese',
        'fruta');
```

```
   title
------------
 Laranjeira
 Ananás
 Limão
```

A concatenação dos campos `title`, `text_` e `tags` foi convertida para `tsvector`.

Só foram exibidos os títulos dos registros, em que a concatenação convertida combinasse com a conversão da string “fruta” para texto de consulta de busca textual.

Consultas de busca textual

E pesquisar uma frase?

```
SELECT
    title
FROM tb_post
WHERE to_tsvector(
        'portuguese',
        title||' '||text_||' '||tags)
    @@ to_tsquery(
        'portuguese',
        'presença humana');
```

```
ERROR:  syntax error in tsquery: "presença humana"
```

A string "`presença humana`" não é uma `tsquery`, *tokens* soltos, sem os operadores "`|`" (*OR*) ou "`&`" (*AND*) não são permitidos.

Consultas de busca textual

Como se faz?

```
SELECT
    title
FROM tb_post
WHERE to_tsvector(
        'portuguese',
        title||' '||text_||' '||tags)
    @@ to_tsquery(
        'portuguese',
        'presença & humana');
```

```
 title
--------
 Música
 Lobo
```

Foi necessário explicitar o texto *tsquery* de forma que tenha as palavras (normalizadas) “presença” e “humana”.

Consultas de busca textual

Pode-se também usar a função `plainto_tsquery` que converte o texto plano passado para o formato *tsquery*:

```
SELECT plainto_tsquery('portuguese', 'presença humana');

   plainto_tsquery
----------------------
 'presenc' & 'human'
```

Vale lembrar que a função `plainto_tsquery` faz a conversão utilizando a lógica *AND* (E).

Ela converte para o formato `tsquery` e normaliza para lexemas, e elimina *stop words*.

Tabela com campo tsvector

- Até aqui os exemplos foram feitos utilizando a conversão em tempo real de campos de texto concatenados para `tsvector`;
- Serve muito bem para exemplificar, porém, é mais custoso para o servidor de banco de dados;
- Consequentemente mais demorada será a *query*:

```
EXPLAIN ANALYZE
SELECT
    title
FROM tb_post
WHERE to_tsvector(
    'portuguese',
    title||' '||text_||' '||tags)
        @@ plainto_tsquery('portuguese', 'presença humana');

. . .

 Planning Time: 0.086 ms
 Execution Time: 12.364 ms
```

Tabela com campo tsvector

Adição de um campo *tsvector* na tabela:

```
ALTER TABLE tb_post
    ADD COLUMN text_vector tsvector
    GENERATED ALWAYS AS (
        to_tsvector(
            'portuguese',
             title||' '||text_||' '||tags))
        STORED;
```

O novo campo é auto gerado, em sua criação já sendo preenchido com seus devidos valores e para futuros `INSERT`s também será preenchido automaticamente.

Até a versão 11 do Postgres isso tinha que ser feito por um *trigger*.

Tabela com campo tsvector

Nova análise dos resultados, agora, com um campo tsvector:

```
EXPLAIN ANALYZE
SELECT
    title
FROM tb_post
WHERE text_vector
    @@ plainto_tsquery(
        'portuguese',
        'presença humana');

. . .

 Planning Time: 0.084 ms
 Execution Time: 0.216 ms
```

Uma significativa no desempenho, mas há como melhorar.

Índices GiST e GIN

Há dois tipos de índices que podem ser usados para acelerar buscas textuais: **GiST** e **GIN**;

Não é obrigatório, mas seu uso beneficia o desempenho de buscas textuais;

Esses tipos de índices são aplicados a colunas dos tipos *tsvector* ou *tsquery*;

Há diferenças significantes entre os dois tipos de índices, então é importante entender suas características.

Índices GiST e GIN: Sintaxe

```
CREATE INDEX nome_indice ON nome_tabela USING gist(nome_coluna);
```

Cria um índice GiST (*Generalized Search Tree* - Árvore de Busca Generalizada)

```
CREATE INDEX nome_indice ON nome_tabela USING gin(nome_coluna);
```

Cria um índice GIN (*Generalized Inverted Index* - Índice Invertido Generalizado).

Índices GiST

Um índice `GiST` tem perdas, o que significa que o índice pode produzir falsos positivos e é necessário checar a linha atual da tabela para eliminar tais falsos positivos (O PostgreSQL faz isso automaticamente quando necessário).

Índices `GiST` têm perdas porque cada documento é representado no índice por uma assinatura de largura fixa.

Perdas fazem com que haja uma degradação de performance devido a buscas desnecessárias nos registros de uma tabela que se tornam falsos positivos.

Como o acesso aleatório a registros da tabela é lento, isso limita a utilidade de índices `GiST`.

A probabilidade de falsos positivos depende de vários fatores, em particular o número de palavras únicas, então o **uso de dicionários** para reduzir esse número é recomendado.

Índices GIN

Um índice `GIN` não tem perdas para consultas padrão, mas sua performance depende logaritmicamente do número de palavras únicas.

No entanto, índices `GIN` armazenam apenas as palavras (lexemas) de valores *tsvector*, e não o peso de suas *labels*.

Assim uma rechecagem de uma linha de uma tabela é necessária quando usa uma consulta que envolve pesos.

Criação do índice:

```
CREATE INDEX idx_text_vector
    ON tb_post
    USING GIN (text_vector);
```

GiST vs GIN: Qual escolher?

Buscas `GIN` são cerca de 3 (três) vezes mais rápidas do que `GiST`;

Índices `GIN` levam um tempo 3 (três) vezes maior para serem construídos do que `GiST`;

Índices `GIN` são moderadamente mais lentos para atualizar do que os índices `GiST`, mas cerca de 10 (dez) vezes mais lentos se o suporte a *fast-update* for desabilitado;

`GIN` são de 2 (duas) a 3 (três) vezes maiores do que índices `GiST`;

GiST vs GIN: Qual escolher? / Resumo

Índices `GIN` são melhores para dados estáticos porque as buscas são mais rápidas.

Para dados dinâmicos, os índices `GiST` são mais rápidos para serem atualizados. Especificamente, índices `GiST` são muito bons para dados dinâmicos e rápidos se palavras únicas (lexemas) forem abaixo de 100.000 (cem mil), enquanto os índices `GIN` lidarão melhor quando for acima disso, porém mais lentos para se atualizarem.

Vale lembrar que o tempo de construção de um índice `GIN` pode frequentemente ser melhorado aumentando o parâmetro `maintenance_work_mem`, porém isso não tem efeito para um índice `GiST`;

Particionamento de grandes coleções e o uso próprio de índices `GiST` e `GIN` permite a implementação de buscas muito mais rápidas com atualização *online*;

Particionamento pode ser feito no nível da base de dados usando herança de tabelas, ou pela distribuição de documentos sobre servidores e coletando resultados de buscas usando o módulo `dblink`. Sendo que esse último é possível por uso de funções de *ranking* em informações locais apenas.

Classificação de resultados de busca

Classificar resultados é tentar medir o quanto um documento é relevante para uma consulta em particular, de modo que quando houver muitos "casamentos" (*matches*), os mais relevantes são exibidos primeiro.

Há duas funções pré definidas de classificação, que leva em conta léxicos, proximidade e estrutura da informação, que é, considerar o quão frequentes os termos de uma consulta aparecem em um documento, como estão próximos os termos, o quão importante é a parte do documento que ocorre.

Porém, o conceito de relevância é vago e varia especificamente para uma aplicação. Aplicações diferentes podem pedir informações adicionais para classificação, e.g.; data de modificação do documento.

As funções *bult-in* de classificação são apenas exemplos. Pode-se escrever funções próprias e / ou combinar seus resultados com fatores adicionais para adequar às suas necessidades específicas.

Classificação de resultados de busca

As duas funções de classificação disponíveis atualmente são:

- **`ts_rank`**

  Classifica os vetores baseados na frequência da combinação de seus lexemas.

- **`ts_rank_cd`**

  Computa a classificação para densidade de correlacionamento* (entre palavras) para um dado vetor de documento ou consulta.

  Esta função requer informação posicional em sua entrada, portanto não funcionará valores `tsvector` "*stripped*"**.

Cover density ranking.

**Submetidos à função `strip()`

Classificação de resultados de busca

Para ambas funções, os argumentos de pesos (*weights*) oferecem a habilidade de fazer com que as buscas sejam feitas com pesos especificados.

A ordem crescente dos pesos é *D*, *C*, *B*, *A*.

Existe uma função que define o peso de entrada de lexemas: a função `setweight`.

Não é possível combinar `tsvector_update_trigger` com `setweight`.

Classificação de resultados de busca

Para processar colunas diferentemente, por exemplo, atribuir pesos diferentes para `título`, `tags` e `text_`, é necessário escrever uma função customizada:

```
CREATE OR REPLACE FUNCTION fc_setweight_tb_post(
    title text,
    text_ text,
    tags text
)
RETURNS tsvector
IMMUTABLE AS
$body$
BEGIN

RETURN
    setweight(to_tsvector('portuguese', coalesce(title,'')), 'A') ||
    setweight(to_tsvector('portuguese', coalesce(tags,'')), 'B') ||
    setweight(to_tsvector('portuguese', coalesce(text_,'')), 'C');

END;
$body$
LANGUAGE PLPGSQL;
```

Classificação de resultados de busca

É necessário recriar o campo *tsvector*:

```
-- Dropping the column
ALTER TABLE tb_post DROP COLUMN text_vector;


-- Recreating the column as auto generated
ALTER TABLE tb_post
    ADD COLUMN text_vector tsvector
    GENERATED ALWAYS AS (
        fc_setweight_tb_post(title, text_, tags))
        STORED;
```

Classificação de resultados de busca

Inserir um novo registro:

```
INSERT INTO tb_post (title, text_, tags) VALUES (
    'Cidade de São Paulo',
    'A cidade de São Paulo é a capital do Estado de mesmo nome e também a'
    ||'mais populosa do Brasil.',
    'metrópole SP caos trânsito violência');
```

Classificação de resultados de busca

Junto aos lexemas, além das posições, há os pesos:

```
SELECT
    text_vector
FROM tb_post
WHERE text_vector @@ to_tsquery('portuguese', 'sp');
```

```
  text_vector

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------
 'brasil':29C 'caos':7B 'capital':17C 'cidad':1A,11C 'estad':19C
'metrópol':5B 'nom':22C 'paul':4A,14C 'popul':27C 'sp':6B 'trânsit':8B
'violênc':9B 'é':15C
```

Destacando resultados na busca textual

Função ts_headline

Apresenta o(s) resultado (s) da busca, de modo a exibir a parte de cada documento como está relacionada com a consulta.

É muito comum, por exemplo, search engines mostrarem fragmentos do documento com os termos de consulta marcados.

Sintaxe:

```
ts_headline([ config regconfig, ] document text, query tsquery [, options text ]) returns text
```

Destacando resultados na busca textual

```
SELECT
    ts_headline(
        'portuguese',
        text_,
        to_tsquery('portuguese', 'fruta'),
        'StartSel = <b>, StopSel =</b>')
FROM tb_post
WHERE text_vector @@ to_tsquery('portuguese', 'laranja');
```

```
                                                ts_headline
-----------------------------------------------------------------------------------------------------------------
 <b>fruto</b> produzido pela laranjeira (Citrus × sinensis), uma árvore da família Rutaceae. A laranja é um
<b>fruto</b>
```

No exemplo é demonstrado como destacar palavras relativas ao termo pesquisado com HTML.

Dicionários

Dicionários são usados para eliminar palavras que não devem ser consideradas em uma busca (palavras de parada / *stop words*) e normalizar palavras em suas formas derivadas para uma base (lexema).

Apesar de melhorar a qualidade de pesquisa, normalização e remoção de *stop words*, reduzem o tamanho da representação do *tsvector* de um documento, de forma a melhorar a performance.

Nem sempre normalização tem um propósito linguístico e geralmente depende da semântica da aplicação.

Um dicionário aceita um *token* como entrada e retorna:

- Um vetor de lexemas se a entrada é conhecida para o dicionário;
- Um vetor vazio se o dicionário conhece o *token*, mas ele é uma *stop word*;
- `NULL` se o dicionário não reconhece o *token* de entrada.

Dicionários

Alguns exemplos de normalização:

- Linguística: dicionários Ispell tentam reduzir palavras de entrada para uma forma normalizada, dicionários stemmer removem o fim de palavras;

- URLs podem ser normalizadas para fazer com que diferentes combinem:

  https://www.python.org/about/

  https://docs.python.org/3/library/index.html

  https://docs.python.org/3/library/stdtypes.html#boolean-type-bool

Dicionários

Exemplos de normalização

- Nomes de cores podem ser substituídas por seus valores hexadecimais, por exemplo: *red*, *green*, *blue*, *magenta* -> `FF0000`, `00FF00`, `0000FF`, `FF00FF`.
- Se indexar números, podemos remover alguns dígitos fracionais para reduzir a faixa de possíveis números, por exemplo:

  `3.14159265359`, `3.1415926`, `3.14` serão os mesmos após normalização se apenas dois dígitos forem mantidos após o ponto decimal.

Dicionários

O PostgreSQL fornece dicionários predefinidos para muitas linguagens.

Há também *templates* predefinidos que podem ser usados para criar novos dicionários com parâmetros customizados.

A regra geral para configurar uma lista de dicionários é alocar primeiramente o mais restrito, mais específico, então os mais gerais depois, fechando com o dicionário mais geral, como um Snowball stemmer ou simples, que reconhecem tudo.

Cabe ao dicionário específico como tratar *stop words*.

Dicionários

Por exemplo, os dicionários Ispell primeiro normalizam palavras e depois buscam na lista de palavras de parada, enquanto stemmers Snowball primeiro verificam a lista de palavras de parada.

A razão para o comportamento diferente é uma tentativa para reduzir o "ruído".

Para listar os dicionários de uma base de dados, utlizamos o comando do psql:

```
\dFd
```

Ou o comando SQL:

```
SELECT * FROM pg_ts_dict;
```

Dicionários

A maioria dos tipos de dicionários dependem de arquivos de configuração, como os arquivos de *stop words*.

Esses arquivos devem ser armazenados com o encoding UTF-8.

Normalmente uma sessão lerá um arquivo de configuração de dicionário uma única vez, quando for usado pela primeira vez dentro da sessão.

Se o arquivo de configuração for modificado e quer forçar as sessões existentes a utilizar seu novo conteúdo, faça:

```
ALTER TEXT SEARCH DICTIONARY dicionário ...
```

Uma solução "*dummy*" que não muda nenhum valor :)

Ou então:

```
ALTER TEXT SEARCH DICTIONARY dicionário (RELOAD);
```

Dicionários simples

O *template* de dicionário simples converte cada *token* de entrada em caixa baixa (letras minúsculas) e faz a checagem em um arquivo de palavras de parada.

Se esse *token* for achado no arquivo, será retornado um vetor vazio, causando o descarte do *token*.

Se o *token* não estiver nessa "lista negra", a forma em letras minúsculas é retornada como um lexema normalizado.

```
CREATE TEXT SEARCH DICTIONARY dic_simple(
    TEMPLATE = pg_catalog.simple,
    STOPWORDS = portuguese
);
```

Dicionários simples

Diretório de buscas textuais:

```
$ cd $SHAREDIR/tsearch_data
```

Busca das palavras “foi” e “sim” no arquivo de palavras de parada em português:

```
$ grep -E 'foi|sim' portuguese.stop
```

```
foi
```

Testes de lexização:

```
SELECT ts_lexize('public.dic_simple', 'foi');
```

```
 ts_lexize
-----------
 {}
```

```
SELECT ts_lexize('public.dic_simple', 'SiM');
```

```
  ts_lexize
-----------
 {sim}
```

Dicionário de sinônimos

A função ts_debug exibe o lexema da palavra “Paris”:

```
SELECT * FROM ts_debug('portuguese', 'Paris');
```

```
   alias   |   description   | token |   dictionaries   |   dictionary   | lexemes
-----------+-----------------+-------+------------------+----------------+---------
 asciiword | Word, all ASCII | Paris | {portuguese_stem} | portuguese_stem | {par}
```

Criação da linha no arquivo de configuração do dicionário:

```
# echo 'Paris paris' > $SHAREDIR/tsearch_data/syn.syn
```

Dicionário de sinônimos

Criação do dicionário de sinônimos:

```
CREATE TEXT SEARCH DICTIONARY dic_syn(
    TEMPLATE = synonym,
    SYNONYMS = syn
);
```

Adicionando o dicionário criado à configuração de busca textual em português:

```
ALTER TEXT SEARCH CONFIGURATION portuguese
    ALTER MAPPING FOR asciiword
    WITH dic_syn, portuguese_stem;
```

Dicionário de sinônimos

Novamente com a função `ts_debug`, mas agora o teste já conta com o dicionário de sinônimos criado:

```
SELECT * FROM ts_debug('portuguese', 'Paris');
```

```
   alias   |   description   | token |         dictionaries          |  dictionary   | lexemes
-----------+-----------------+-------+-------------------------------+---------------+---------
 asciiword | Word, all ASCII | Paris | {dic_sinonimos,portuguese_stem} | dic_sinonimos | {paris}
```

Dicionário thesaurus

Um dicionário Thesaurus (às vezes abreviado como TZ) é uma coleção de palavras que incluem informação sobre a relação entre palavras e frases, por exemplo: termos mais amplos (*broader terms*: BT), termos mais restritos (*narrower terms*: NT), termos preferidos, termos não preferidos, termos relacionados, etc.

Basicamente substitui todos termos não preferidos por um termo preferido e opcionalmente preserva os termos originais para indexação, por exemplo.

Dicionário Thesaurus

Na implementação atual do PostgreSQL do dicionário Thesaurus é uma extensão do dicionário de sinônimos com suporte a frase adicionado.

Esse tipo de dicionário requer um arquivo de configuração no seguinte formato:

```
# comentário
palavra(s) de exemplo : palavra(s) indexada(s)
mais palavra(s) de exemplo : mais palavra(s) indexada(s)
…
```

O caractere ":" age como um delimitador entre uma frase e sua substituição.

Dicionário Thesaurus

Um dicionário Thesaurus usa um subdicionário (que é especificado na configuração de dicionário) para normalizar o texto de entrada antes de checar por frases que combinem.

É possível selecionar apenas um subdicionário.

Um erro é reportado se o subdicionário falha para reconhecer uma palavra. Nesse caso, deve-se remover o uso da palavra ou ensinar o dicionário sobre ela.

Pode-se colocar um "*" no começo de uma palavra indexada para pular a aplicação do subdicionário para ela, mas todas palavras de exemplo devem ser conhecidas para o subdicionário.

Dicionário Thesaurus

O dicionário Thesaurus escolhe a combinação mais longa se há múltiplas frases combinando com a entrada, e os laços são quebrados usando a última definição.

Stop words específicas reconhecidas pelo subdicionário devem ser precedidas com "?".

**** Atenção ****

Dicionários do tipo *thesaurus* são usados durante indexação, então qualquer mudança em seus parâmetros requer reindexação, diferentemente de outros tipos de dicionários.

Dicionário Thesaurus

Configuração Thesaurus

Para definir um novo dicionário Thesaurus, use o modelo (*template*) Thesaurus, por exemplo:

```
# cp $SHAREDIR/tsearch_data/thesaurus_sample.ths \
    $SHAREDIR/tsearch_data/pt_br_tz.ths
```

Adiconar linhas de regras ao arquivo:

```
# cat << EOF >> $SHAREDIR/tsearch_data/pt_br_tz.ths
cidade luz : *Paris cidade luz
jogar bola : futebol
pasta ? dente : *dentifrício
EOF
```

Dicionário Thesaurus

Criação do dicionário Thesaurus:

```
CREATE TEXT SEARCH DICTIONARY dic_pt_br_tz (
    /* template */
    TEMPLATE = thesaurus,

    /* configuration file (.ths). */
    DictFile = pt_br_tz,

    /* subdictionary (Portuguese Snowball stemmer) */
     Dictionary = pg_catalog.portuguese_stem);
```

Agora é possível vincular o dicionário `pt_br_tz` para os tipos de *tokens* desejados em uma configuração, por exemplo:

```
ALTER TEXT SEARCH CONFIGURATION portuguese
    ALTER MAPPING FOR
        asciiword,
        asciihword,
        hword_asciipart
    WITH dic_pt_br_tz, portuguese_stem;
```

Dicionário Thesaurus

```
SELECT to_tsvector('portuguese', 'cidade luz');
```

```
        to_tsvector
----------------------------
 'Paris':1 'cidad':2 'luz':3
```

```
SELECT to_tsvector('portuguese', 'jogar bola');
```

```
 to_tsvector
-------------
 'futebol':1
```

```
SELECT to_tsvector('portuguese', 'pasta de dente');
```

```
   to_tsvector
-----------------
 'dentifrício':1
```

```
SELECT
    to_tsvector(
        'portuguese',
        'Após usar a pasta de dente vou jogar bola na cidade luz');
```

```
                                to_tsvector
-----------------------------------------------------------------------------------
 'Paris':8 'após':1 'cidad':9 'dentifrício':4 'futebol':6 'luz':10 'usar':2 'vou':5
```

Dicionário Thesaurus

Se for necessário que uma palavra indexada não seja lexemizada, deve ser colocado o asterisco na frente dela:

```
# echo 'sp : *São *Paulo' >> $SHAREDIR/tsearch_data/pt_br_tz.ths
```

Um teste:

```
SELECT to_tsvector('portuguese','sp');

 to_tsvector
-------------
 'sp':1
```

O teste não deu certo pois é necessário que se recarregue as configurações do dicionário.

Dicionário Thesaurus

Dentro da sessão recarregar a configuração do dicionário:

```
ALTER TEXT SEARCH DICTIONARY dic_pt_br_tz (RELOAD);
```

Novo teste:

```
SELECT to_tsvector('portuguese','sp');
```

```
     to_tsvector
--------------------
 'Paulo':2 'São':1
```

Dicionário Ispell

O *template* de dicionário Ispell suporta dicionários morfológicos.

Pode normalizar muitas diferentes formas linguísticas de uma palavra em um mesmo lexema.

Exemplo (em português):

“correu”, “correndo”, “correria”, “corro” e “corrermos”.

Dicionário Ispell

Para criar um dicionário Ispell, use o *template built-in* e especifique outros parâmetros:

```
CREATE TEXT SEARCH DICTIONARY dic_pt_br_ispell (
    TEMPLATE = ispell,
    DictFile = pt_br,
    AffFile = pt_br,
    StopWords = portuguese
);
```

`DictFile`, `AffFile` e `StopWords` especificam os nomes, respectivamente dos arquivos de dicionário, afixos e palavras de parada.

Dicionários Ispell normalmente reconhecem um conjunto limitado de palavras, então eles devem ser seguidos por outro dicionário mais amplo, como um dicionário Snowball, que reconhece tudo.

Dicionário Ispell

Dicionários Ispell suportam divisão de palavras compostas, uma característica muito útil.

Observação:

MySpell não suporte palavras compostas.
Hunspell tem suporte sofisticado para palavras compostas.
Atualmente, o PostgreSQL implementa apenas operações básicas do Hunspell para palavras compostas.

Dicionário Ispell

Criação de uma nova configuração de busca textual, usando uma pré existente como modelo:

```
CREATE TEXT SEARCH CONFIGURATION tsc_pt_br
     (COPY = pg_catalog.portuguese);
```

Adicionando o dicionário Ispell criado à configuração:

```
ALTER TEXT SEARCH CONFIGURATION tsc_pt_br
     ALTER MAPPING FOR
          asciiword,
          asciihword,
          hword_asciipart,
          word,
          hword,
          hword_part
    WITH dic_pt_br_ispell, portuguese_stem, simple;
```

Dicionário Ispell

Testando (verbos irregulares):

```
SELECT
    to_tsvector('tsc_pt_br', 'caberia') @@
    to_tsquery('tsc_pt_br', 'caibo');

 ?column?
----------
 t

SELECT to_tsvector('tsc_pt_br', 'eu trouxe!!!');

 to_tsvector
-------------
 'trazer':2
```

Dicionário Snowball

Este *template* de dicionário é baseado em um projeto de Martin Porter, inventor do popular algoritmo de *stemming* para o inglês.

Snowball hoje fornece algoritmo de *stemming* para muitos idiomas.

Cada algoritmo entende como reduzir formas variantes comum de palavras para uma base, ou *stem*, na ortografia de seu idioma.

Um dicionário Snowball requer um parâmetro de língua para identificar que *stemmer* usar, e opcionalmente poder especificar

Um arquivo de palavras de parada que é uma lista de palavras para eliminar.

As listas de palavras de parada padrões do PostgreSQL também são fornecidas pelo projeto Snownball.

http://snowball.tartarus.org/

Dicionário Snowball

Criação de um dicionário Snowball:

```
CREATE TEXT SEARCH DICTIONARY portuguese_stem (
    TEMPLATE = snowball,
    Language = portuguese,
    StopWords = portuguese
);
```

Um dicionário Snowball reconhece tudo, simplificando uma palavra ou não, então ele deve ser alocado ao final da lista de dicionários.

É inútil tê-lo antes que qualquer outro dicionário, pois um *token* nunca passará por ele ao dicionário seguinte.

Módulo unaccent

É um módulo *contrib* que implementa um dicionário de busca textual que remove acentos de lexemas.

É um dicionário de filtragem, o que significa que sua saída é sempre passada para o próximo dicionário (se for o caso), ao contrário do comportamento normal de dicionários. Isso permite processamento de busca textual sem considerar acentos.

A implementação atual do unaccent não pode ser usada como um dicionário de normalização para o dicionário Thesaurus.

Módulo unaccent: configuração

`RULES` é o nome do arquivo que contém a lista de regras de tradução.

Esse arquivo deve estar armazenado em `$SHAREDIR/tsearch_data/`, cujo nome deve ter a extensão `.rules` (tal extensão não deve ser incluída no parâmetro `RULES`).

O arquivo de regras deve ter o seguinte formato:

```
À       A
Á       A
Â       A
Ã       A
Ä       A
Å       A
Æ       A
```

Módulo unaccent

Cada linha representa um par, consistindo de um caractere com acento seguido que será traduzido pelo seguinte que é sem acento.

Um exemplo mais completo, que é diretamente útil para a maioria dos idiomas europeus, pode ser encontrado no arquivo `unaccent.rules`, que é instalado do diretório `$SHAREDIR/tsearch_data/` quando o módulo é instalado.

Para instalar o módulo acesse a base de dados que deseja nela trabalhar com o dicionário *unaccent* e dê o comando:

```
CREATE EXTENSION unaccent;
```

Módulo unaccent: uso

Ao instalar a extensão `unaccent` cria-se um *template* de busca textual `unaccent` e um dicionário `unaccent` baseados nela.

O dicionário unaccent tem o parâmetro padrão de configuração `RULES='unaccent'`, faz com que seja imediatamente utilizável pelo arquivo `unaccent.rules`.

Módulo unaccent

Se preferir, pode alterar o parâmetro, por exemplo:

```
ALTER TEXT SEARCH DICTIONARY unaccent (RULES='minhas_regras');
```

...ou criar novos dicionários baseados no *template*.

```
SELECT ts_lexize('unaccent', 'Armação');

 ts_lexize
-----------
 {Armacao}
```

Módulo unaccent

Um exemplo de como inserir um dicionário unaccent em uma configuração de busca textual:

```
CREATE TEXT SEARCH CONFIGURATION tsc_pt_unaccent
     (COPY = portuguese);
```

Módulo unaccent

```
ALTER TEXT SEARCH CONFIGURATION tsc_pt_unaccent
    ALTER MAPPING FOR asciiword,
                      Asciihword,
                      hword_asciipart,
                      word,
                      hword,
                      hword_part
   WITH
      dic_pt_br_ispell,
      unaccent,
      portuguese_stem,
      simple;
```

Observação:

Na função `ts_lexize`, seu primeiro parâmetro é um dicionário de busca textual e não uma configuração de busca textual.

Módulo unaccent

```
SELECT
    to_tsvector('tsc_pt_unaccent', 'Hotéis do Mar');

    to_tsvector
--------------------
 'hotel':1 'mar':3


SELECT
    to_tsvector('tsc_pt_unaccent', 'Hotéis do Mar')
    @@ to_tsquery('tsc_pt_unaccent','Hotel');

 ?column?
----------
 t
```

Módulo unaccent

Destacar o que combinar com “Hotel”:

```
SELECT
    ts_headline(
        'tsc_pt_unaccent',
        'Hotéis do Mar',
        to_tsquery(
            'tsc_pt_unaccent',
            'Hotel'));
```

```
     ts_headline
-----------------------
 <b>Hotéis</b> do Mar
```

Módulo unaccent: função unaccent

- Remove acentos de uma dada *string*.
- Basicamente, é um invólucro (wrapper) que envolve o dicionário `unaccent`, mas pode ser usada fora do contexto de busca textual.

  unaccent([dicionário,] string) returns text

```
SELECT unaccent('unaccent', 'Hotéis');

 unaccent
----------
 Hoteis
```