Full Text Search

FONCTIONNEMENT DANS POSTGRESQL

DALIBO
L'expertise PostgreSQL

AUTEUR

- Adrien Nayrat
- Consultant PostgreSQL chez Dalibo
  - email : adrien.nayrat@dalibo.com
  - twitter : @Adrien_nayrat
  - blog : http://blog.anayrat.info/

- Hashtag de la journée : #pgday_fr
- Licence : Creative Common BY-NC-SA

AU MENU

- Comment rechercher de l'information dans un document?
  - La Recherche d'Information
  - Full Text Search dans Postgres
  - Recherche dans un jeu de données : Stackoverflow

COMMENT FAIRE UNE RECHERCHE?

- Notre cerveau sait :
  - Identifier les synonymes
  - Ignorer les mots non importants "le, la,..."
  - Sans forcément prêter attention à la casse, ponctuation
    - But retenir les mots porteurs de sens
    - Traiter de manière automatisée un langage naturel

DÉFINITION

La recherche d'information (RI) est le domaine qui étudie la manière de retrouver des informations dans un corpus...

[...]représenter des documents dans le but d'en récupérer des informations, au moyen de la construction d’index. - Wikipedia

RI (COMPOSANTS) 1/2

- Pré-traitement :
  - Extraire les descripteurs :
    - Suppression des mots outils ou mots vides
    - Lemmatisation *stemming* : Obtenir la racine des mots
    - Remplacer des synonymes
    - Utiliser un thésaurus

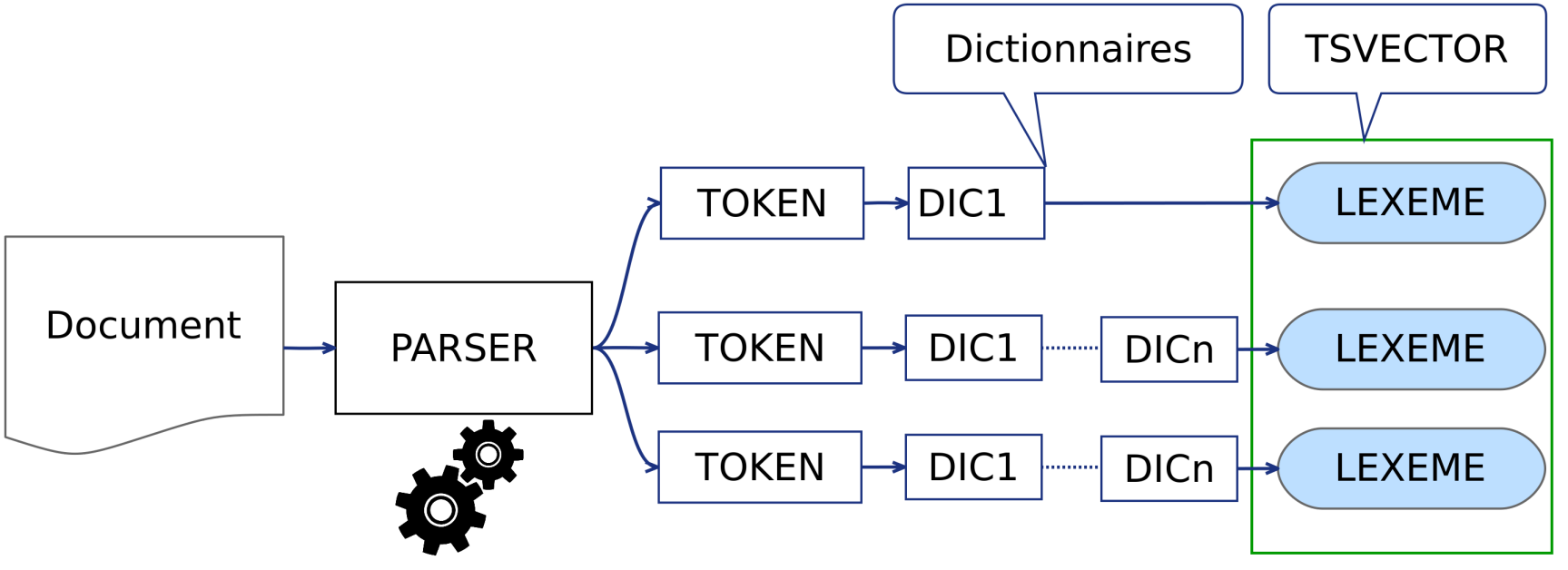
RI (COMPOSANTS) 2/2

- But : obtenir une représentation vectorielle d'un document
- Indexation des vecteurs
- Interrogation des index à l'aide d'un langage "informatique"

DANS POSTGRESQL?

- On parle de Recherche Plein Texte (**Full Text Search**)
- Plusieurs composants formant une chaîne :
  - Parser
  - Un ou des dictionnaires
- But : obtenir une liste triée de lexèmes formant un *tsvector*

MACHINE FULL TEXT SEARCH



LEXÈME?

Définition : Morphème lexical d'un lemme, c'est-à-dire une unité de sens et de son qui n'est pas fonctionnelle ou dérivationnelle.

- Les mots utiles et seulement leur racine.
- Exemple : un verbe sans terminaison :
  - Empêcher ⇒ empech

TYPE TSVECTOR

- Liste triée de lexèmes

```
SELECT  to_tsvector('french','Le meilleur SGBD Opensource');
              to_tsvector
-----------------------------------------
 'meilleur':2 'opensourc':4 'sgbd':3
```

PARSER

- But : identifier des *tokens*
- Exemple :
  - `word` : Mot comportant des lettres
  - `int` : Entier signé
  - `url` : Lien
  - `email` : adresse mail
  - `tag` : balise xml
  - `blank` : espace

EXEMPLE

PGday France	contact@pgday.fr	http://pgday.fr/contact.html
Word	email	url

```
SELECT alias,description,token
FROM ts_debug('PGDay France contact@pgday.fr http://pgday.fr/contact.h
   alias    |   description   |          token
------------+-----------------+-----------------------
 asciiword  | Word, all ASCII | PGDay
 blank      | Space symbols   |
 asciiword  | Word, all ASCII | France
 blank      | Space symbols   |
 email      | Email address   | contact@pgday.fr
 blank      | Space symbols   |
 protocol   | Protocol head   | http://
 url        | URL             | pgday.fr/contact.html
 host       | Host            | pgday.fr
 url_path   | URL path        | /contact.html
```

DICTIONNAIRES

- Succession de filtres permettant d'obtenir un lexème (lemmatisation)
  - Supprimer la casse
  - Retirer les *stopswords* (mots vides)
  - Remplacer des synonymes
  - ...

UNE CONFIGURATION FTS C'EST :

- Un parser
- Plusieurs dictionnaires
- Mapping : applique les dictionnaires en fonction des catégories de token.

MAPPING PAR DÉFAUT

```
\dF+ english
Text search configuration "pg_catalog.english"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciiword       | english_stem
 email           | simple
 float           | simple
...
 int             | simple
 url             | simple
 word            | english_stem
```

OPÉRATEURS

- Comment interroger les `tsvector` ?
  - Type `tsquery`
  - Opérateur `@@`
  - Fonctions `to_tsquery`, `plainto_tsquery` et `phraseto_tsquery`

TYPE TSQUERY

- Comprend les lexèmes recherchés qui peuvent être combinés avec les opérateurs suivants :
  - & (AND)
  - | (OR)
  - ! (NOT)
  - L'opérateur de recherche de phrase (depuis la 9.6) : <-> (FOLLOWED BY)

EXEMPLE :

- Une recherche dans google de type “chat AND chien” se traduirait en :

```
SELECT 'chat & chien'::tsquery;
     tsquery
------------------
 'chat' & 'chien'
```

OPÉRATEUR @@

- Permet d'interroger un tsvector

```
SELECT to_tsvector('chat chien') @@ 'chat'::tsquery;
----------
 t
```

```
SELECT to_tsvector('french','cheval poney') @@ 'cheval'::tsquery;
----------
 t

SELECT to_tsvector('french','cheval poney') @@ 'chevaux'::tsquery;
----------
 f

SELECT to_tsvector('french','chevaux');
 to_tsvector
-------------
 'cheval':1
```

- On compare un mot à un lexème
- On devrait comparer deux lexèmes

FONCTION TO_TSQUERY

- Transforme une chaîne de texte en `tsquery` composée de lexèmes

```
SELECT  to_tsquery('french','chevaux');
 to_tsquery
------------
 'cheval'
SELECT to_tsvector('french','cheval poney')
@@ to_tsquery('french','chevaux');
----------
 t
```

PLAINTO_TSQUERY

- Convertit une chaîne de texte en `tsquery`
- `phraseto_tsquery` permet la recherche de phrase

```
SELECT plainto_tsquery('french','chevaux poney');
  plainto_tsquery
--------------------
 'cheval' & 'poney'
```

PERFORMANCE?

- Tests sur la base *stackoverflow*

```
 \dt+ posts
                      List of relations
 Schema | Name  | Type  |  Owner   | Size  | Description
--------+-------+-------+----------+-------+-------------
 public | posts | table | postgres | 37 GB |
```

RECHERCHE DE QUELQUES MOTS

```
SELECT * FROM posts
WHERE to_tsvector('my_fts',body)
  @@ plainto_tsquery('my_fts','postgres full text search')
...
  ->  Seq Scan on posts  (cost=0.00..12755502.24 rows=5 width=84)
       Filter: (to_tsvector('my_fts'::regconfig, body)
        @@ '''pgsql'' & ''full'' & ''text'' & ''search'''::tsquery)
```

- Le moteur doit lire l'intégralité de la table

INDEXATION

Indexer un `tsvector` : **GIN** & **GiST**

```
SELECT amname from pg_am;
 amname
--------
 btree
 hash
 gist
 gin
 spgist
 brin
 bloom
```

GIN

Gin stands for Generalized Inverted Index and should be considered as a genie, not a drink. - src/backend/access/gin/README

- Index Inversé : Contient chaque élément d'un tsvector
- Compressé depuis la 9.4
- Très performant en lecture

GIST

- *Generalized Search Tree*
- Indexe la véracité d'un prédiat
- Rapide à construire
- Lent en lecture et volumineux

INDEXER UN TSVECTOR

- Indexer une colonne tsvector
  - Oblige à maintenir une colonne supplémentaire
  - Permet de concatener des champs et d'attribuer des poids
- Index fonctionnel
  - Simple à utiliser

ORDRE SQL

```
CREATE INDEX ON posts USING gin (to_tsvector('my_fts',body));
```

RECHERCHE AVEC PLUSIEURS MOTS

```
SELECT 'http://stackoverflow.com/questions/' || posts.id as url_post,
  posts.title
FROM posts
WHERE to_tsvector('my_fts',body)
  @@ plainto_tsquery('my_fts','postgres full text search')
LIMIT 5;
```

RECHERCHE D'UNE PHRASE

```
SELECT 'http://stackoverflow.com/questions/' || posts.id as url_post,
  posts.title
FROM posts
WHERE to_tsvector('my_fts',body)
  @@ phraseto_tsquery('my_fts','full text search')
LIMIT 5;
```

CLASSEMENT

```
\set terms 'postgres&full<->text<->search'
SELECT 'http://stackoverflow.com/questions/' || posts.id as url_post,
  posts.title,
  ts_rank_cd(vector_weight,to_tsquery('my_fts',:'terms'),4|8) as rang
FROM posts
WHERE vector_weight @@ to_tsquery('my_fts',:'terms')
ORDER BY rang DESC
LIMIT 5;
```

EXTENSIBILITÉ

Tout est extensible!

- `CREATE TEXT SEARCH PARSER`
- `CREATE TEXT SEARCH CONFIGURATION`
- `CREATE TEXT SEARCH TEMPLATE`
- `CREATE TEXT SEARCH DICTIONARY`

EVOLUTION DU FTS

- Dictionnaire en mémoire partagée
- Index RUM
- FTS sur objet JSON/JSONB (v10)
- ...

Questions?