Mise en œuvre du moteur de recherche de ce blog

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 mars 2009

https://www.bortzmeyer.org/moteur-recherche-wsgi.html

Le tout nouveau moteur de recherche de ce blog <https://www.bortzmeyer.org/moteur-recherche.html> repose sur plusieurs techniques que j'apprécie, notamment WSGI <https://www.bortzmeyer.org/wsgi.html>, TAL <https://www.bortzmeyer.org/generer-html-avec-tal.html> et la recherche texte de PostgreSQL.

Les données sont à l'origine dans des fichiers XML (voir l'article « Mise en œuvre de ce blog <https://www.bortzmeyer.org/blog-implementation.html> »). Ces données sont, chaque nuit, lues par un programme Python, `blog2db.py`, qui traduit le XML en texte et met ledit texte dans une base de données PostgreSQL. Cette base est configurée (script de création en `create.sql`) pour faire de la recherche plein texte <https://www.bortzmeyer.org/postgresql-recherche-texte.html>. Je peux y accéder depuis la ligne de commande Unix ainsi :

```
% search-blog    médiane
             filename              |                        title
-----------------------------------+-----------------------------------------------------
 mediane-et-moyenne.entry_xml      | Médiane et moyenne
 2681.rfc_xml                      | A Round-trip Delay Metric for IPPM
 le-plus-rapide-dns.entry_xml      | Quel est le plus rapide serveur DNS d'un groupe ?
 echoping-6-fini.entry_xml         | Version 6 d'echoping
...
```

où le script `search-blog` fait simplement une requête SQL

```
psql -c "SELECT filename,trim(replace(substr(title,1,63),E'\n','')) \
          AS title  \
          FROM Blog.search('$query') LIMIT 13;" \
   blog
```

Et pour le Web ? WSGI <https://www.bortzmeyer.org/wsgi.html> est utilisé comme plateforme de développement et d'exécution du script Python qui anime l'interface Web du moteur de recherche </search>. Le langage de gabarit utilisé est TAL <https://www.bortzmeyer.org/generer-html-ave
html> (alors que le reste du blog est en Cheetah, je sais, ce n'est pas très cohérent). Le gabarit est nommé `search.xhtml` et le programme WSGI est `search.py`.

Le programme se connecte à la base de données, la connexion reste ouverte tant que le démon WSGI tourne. On ne se connecte donc pas à chaque requête. Mais la base de donnés posait un autre problème. Une fois qu'un a configuré PostgreSQL pour faire de la recherche plein texte, on a le choix entre deux syntaxes pour exprimer les requêtes :

— La syntaxe normale à base de & pour ET, de — pour OU, par exemple 'postgresql&python' pour trouver les articles qui parlent du SGBD et du langage de programmation. C'est celle qu'on a avec `to_tsquery()`.
— La syntaxe sans opérateurs explicites par exemple 'postgresql python' pour la même requête. C'est celle qu'on a avec `plainto_tsquery()`.

Aucune des deux ne me convenait tout à fait. Je voulais une syntaxe sans opérateurs mais où l'opérateur implicite soit OU et pas ET comme avec `plainto_tsquery()`.

Le code du programme essaie donc `to_tsquery()` :

```
try:
    cursor.execute("SELECT to_tsquery('french', '%(query)s')" % \
                           {'query' : query})
except psycopg2.ProgrammingError:
    # Not PostgreSQL FTS syntax, convert it
    query = to_postgresql(query, default_is_or)
    cursor.execute("ROLLBACK;")
```

et, si cela échoue, il traduit la requête en PostgreSQL avec l'opérateur par défaut sélectionné (via un bouton radio) :

```
def to_postgresql(q, default_or = True):
    """ Converts query q (typically an human-entered string with funny
    characters to PostgreSQL FTS engine syntax. By default, the connector is OR.
    """
    q = separators.sub(" ", q)
    q = q.strip()
    if default_or:
        connector = "|"
    else:
        connector = "&" # AND
    return re.sub(" +", connector, q)
```

Le système n'est pas parfait : PostgreSQL accepte des requêtes non documentées (par exemple « foo ;bar » est accepté et équivalent à « foo&bar ») mais il permet de satisfaire l'utilisateur avancé (qui veut contrôler précisément la recherche grâce au langage de requête de PostgreSQL) et les autres, qui veulent simplement taper une liste de mots.

Tous les fichiers nécessaires sont distribués dans l'archive des programmes qui animent ce blog (en ligne sur `https://www.bortzmeyer.org/files/blog-support-files.tar.gz`).