Engineer

Recherche plein texte

recherche de documents par mots et classement par pertinence

Ce que c’est

La recherche plein texte trouve les documents contenant les mots d’une requête, quelles que soient les formes de ces mots, et peut présenter les meilleures correspondances en premier. En anglais, le terme full-text search s’écrit aussi full text search, sans trait d’union. Un document désigne ici ce que l’application considère comme une unité de recherche : un courriel, un article, un ticket d’assistance ou le titre et le corps d’une ligne réunis.

Ce qui la distingue d’une recherche par motif, c’est le travail effectué avant toute comparaison. Le texte est d’abord découpé en unités lexicales : mots, nombres, adresses électroniques et autres catégories. Chaque unité est ensuite normalisée en lexème, une forme choisie pour ramener les variantes d’un mot à une même représentation. Les majuscules deviennent des minuscules, un algorithme de racinisation retire des terminaisons comme celle du pluriel anglais, et les mots vides, trop fréquents pour être significatifs, sont supprimés. Le résultat est stocké avec la position de chaque lexème. Une fonction de classement peut ainsi favoriser les documents où les mots recherchés sont proches. La requête subit le même traitement : la recherche de « payments » et un document contenant « payment », respectivement « paiements » et « paiement », se retrouvent sur une forme commune.

PostgreSQL intègre ce mécanisme à la base de données. Le type `tsvector` contient la forme normalisée d’un document, `tsquery` contient une requête, et l’opérateur `@@` indique si l’un correspond à l’autre. Une configuration de recherche textuelle détermine la normalisation : quel analyseur découpe le texte et quels dictionnaires transforment les unités lexicales en lexèmes. La distribution standard fournit des configurations pour de nombreuses langues, notamment `english`, `danish`, `norwegian` et `swedish`. La commande

psql `\dF` liste celles qui sont installées. SQLite propose le même principe dans son extension FTS5 : une table virtuelle contient l'index et l'opérateur `MATCH` l'interroge.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la recherche plein texte ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Identifier les requêtes et les index manquants dans un système alourdi au fil des ans, puis corriger ce qui compte.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 30 jours de données électriques traités en 180 ms — 180 ms pour agréger un mois de relevés de compteurs

30 jours de données électriques traités en 180 ms Contactez-nous →

Pourquoi c'est important

La documentation PostgreSQL explique ce qui manque aux anciens opérateurs. `LIKE`, `ILIKE` et les expressions régulières n'offrent pas de traitement linguistique : rechercher « satisfy » manque un document contenant « satisfies », à moins d'énumérer toutes les formes dérivées. Ils ne classent pas les résultats par pertinence : mille correspondances reviennent sans ordre utile. Ils ne peuvent pas non plus utiliser un index ordinaire pour ce type de recherche : chaque requête lit donc chaque document. Un motif comme `%cat%` trouve aussi « category » et « concatenate » : c'est du bruit plutôt qu'un gain de rappel.

Les personnes qui saisissent des mots dans un champ de recherche s'attendent à trouver des documents sur ces sujets. Dans des archives de contrats, un service d'assistance ou un catalogue de produits, la recherche plein texte fait la différence entre un champ de recherche auquel on se fie et un autre que l'on abandonne au troisième échec. Le classement compte autant que la correspondance : une personne recherchant un retard de paiement veut voir d'abord la lettre de relance, pas les cent factures qui mentionnent une fois le paiement.

L'intégrer à la base déjà exploitée présente un avantage pratique. L'index de recherche participe à la même transaction que les données : une ligne devient recherchable dès la validation de la transaction, et une ligne supprimée disparaît des résultats au même moment. Les filtres sur les dates, les clients ou les autorisations sont de simples clauses `WHERE` placées à côté de la condition

textuelle. Un serveur de recherche distinct peut faire davantage, mais c'est un deuxième système à alimenter et à synchroniser avec le premier. Tout décalage produit un résultat absent ou qui aurait dû disparaître.

Il faut aussi connaître dès le départ une limite de la recherche plein texte : elle rapproche les mots, pas le sens. Une recherche de « car » ne trouve pas « automobile », deux mots anglais désignant une voiture, sans dictionnaire de synonymes les reliant. La recherche vectorielle, qui compare des représentations numériques du sens, couvre ce cas ; les deux approches sont souvent combinées.

À quoi cela ressemble en pratique

Prenons une table de documents contenant un titre, un corps et la provenance de chacun. Une colonne générée stockée tient leur forme recherchable à jour à chaque insertion et modification. Un index GIN accélère les recherches :

```
CREATE TABLE documents (
    id         bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    title      text,
    body       text,
    source     text NOT NULL,
    search_vec tsvector GENERATED ALWAYS AS (
                   setweight(to_tsvector('english', coalesce(title, '')), 'A')
                || setweight(to_tsvector('english', coalesce(body, '')), 'D')
               ) STORED
);

CREATE INDEX documents_search_idx ON documents USING GIN (search_vec);
```

Trois détails sont déterminants. La configuration est nommée, `'english'`, plutôt que laissée au `default_text_search_config` de la session : la forme de `to_tsvector` à deux arguments est celle que PostgreSQL peut stocker et indexer. `coalesce` transforme un titre ou un corps absent en chaîne vide, car un seul NULL rendrait sinon tout le document NULL. Enfin, `setweight` marque chaque lexème selon sa provenance : A pour le titre, D pour le corps. Sans indication contraire, les fonctions de classement utilisent les poids {0,1 ; 0,2 ; 0,4 ; 1,0} pour D, C, B et A : un mot recherché dans le titre compte donc dix fois plus que dans le corps.

GIN est le type d'index recommandé par la documentation pour la recherche textuelle : un index inversé, avec une entrée par lexème et une liste compressée des lignes qui le contiennent. GiST fonctionne aussi, mais sa représentation

approximative impose de vérifier la ligne de la table pour écarter les fausses correspondances.

La requête reçoit les mots tels que l'utilisateur les a saisis :

```
SELECT id, title, source,
       ts_rank_cd(search_vec, query) AS rank
FROM   documents,
       websearch_to_tsquery('english', 'unpaid invoice -draft') AS query
WHERE  search_vec @@ query
ORDER  BY rank DESC
LIMIT  10;
```

`websearch_to_tsquery` lit la saisie comme un moteur de recherche web. Tous les mots ordinaires doivent être présents, sous une forme que l'algorithme de racinisation anglais ramène à la même racine. Le texte entre guillemets doubles doit apparaître comme une séquence, le mot « or » propose une alternative et un signe moins initial exclut un mot. Cette requête écarte donc tout document contenant « draft », c'est-à-dire « brouillon ». La fonction ne lève jamais d'erreur de syntaxe, ce qui en fait le choix sûr pour une saisie brute. `ts_rank_cd` classe par densité de couverture, une méthode issue d'un article de Clarke, Cormack et Tudhope publié en 1999, qui favorise aussi la proximité des mots recherchés. Sa fonction sœur `ts_rank` compte leurs occurrences. Toutes deux acceptent un indicateur de normalisation : 1, par exemple, divise le score par la somme du logarithme de la longueur du document et d'une unité, pour qu'un texte long ne l'emporte pas du seul fait de sa longueur.

Le même besoin traverse le travail de l'entreprise sur les données pour les agents d'IA. La page décrit la conversion d'anciens PDF, de documents numérisés et d'archives de courriels en texte recherchable, avec leur source conservée à leurs côtés : les retrouver, plutôt que seulement les stocker. La colonne `source` ci-dessus joue ce rôle : une correspondance est plus utile si le lecteur peut ouvrir l'original.

SQLite atteint le même résultat plus simplement. Une table virtuelle déclarée avec `CREATE VIRTUAL TABLE notes USING fts5(title, body)` s'interroge avec `WHERE notes MATCH 'invoice NOT draft'` . `ORDER BY bm25(notes)` place les meilleures correspondances en premier : FTS5 leur attribue des valeurs numériques plus

faibles, donc le tri croissant implicite convient. Le segmenteur standard est `unicode61` ; `porter` ajoute la racinisation et, selon sa documentation, est conçu uniquement pour l'anglais.

Où les problèmes surviennent

Un index que la requête n'utilise jamais. Un index d'expression sur `to_tsvector('english', body)` ne sert qu'aux requêtes répétant exactement cette expression. Une requête écrite `to_tsvector(body)`, qui s'appuie sur la configuration active de la session, lit toute la table. Stockez le vecteur dans une colonne générée, comme ci-dessus, et interrogez cette colonne : la requête ne peut alors plus s'écarter de l'index.

Un NULL qui cache un document. `to_tsvector(NULL)` renvoie NULL ; concaténer un titre NULL avec un corps donne NULL. Une ligne sans titre disparaît donc silencieusement de toutes les recherches. Entourez chaque champ de `coalesce`.

Une saisie brute transmise à `to_tsquery`. `to_tsquery` attend sa propre syntaxe d'opérateurs et tolère moins d'écarts que les autres fonctions de conversion. Des parenthèses déséquilibrées ou une esperluette égarée conduisent l'utilisateur à une page d'erreur. Passez les saisies par `websearch_to_tsquery`, ou par `plainto_tsquery` si aucun opérateur n'est souhaité.

La mauvaise langue. Un texte danois indexé avec la configuration anglaise subit les règles de racinisation et le filtrage des mots vides de l'anglais : la moitié des variantes ne se rejoignent jamais. Choisissez la configuration correspondant à la langue du texte. Une collection multilingue peut stocker le nom de la configuration dans sa propre colonne et indexer `to_tsvector(config_name, body)`, à condition que les requêtes désignent la même colonne.

Classer toutes les correspondances. Le classement lit le `tsvector` de chaque ligne correspondante ; la documentation avertit qu'il peut être lent et limité par les entrées-sorties. Un seul mot correspondant à la moitié de la table oblige la base à classer cette moitié avant que `LIMIT` n'en retienne dix. Réduisez d'abord les candidats avec des conditions ordinaires, comme une plage de dates ou un client, et conservez le vecteur stocké pour éviter de le recalculer pour chaque ligne.

Un score présenté comme un pourcentage. Les fonctions de classement n'utilisent aucune information sur la collection entière : un score ne peut pas se transformer honnêtement en « pertinent à 87 % ». L'indicateur de normalisation 32 ramène chaque score à une valeur entre 0 et 1, mais la documentation qualifie ce changement de cosmétique. Utilisez le score uniquement pour ordonner les résultats.

Des extraits qui transportent du balisage. `ts_headline` renvoie un extrait avec les mots recherchés marqués, exactement ce qu'attend une page de résultats. PostgreSQL avertit qu'il n'est « pas garanti que sa sortie puisse être incluse directement dans des pages web en toute sécurité ». Un document contenant du HTML peut introduire un script dans la page. Supprimez le balisage de l'entrée ou passez la sortie par un outil d'assainissement HTML. La fonction travaille en outre sur le texte original, pas sur le vecteur : appelez-la uniquement pour les lignes de la page, après `LIMIT`, et non pour chaque correspondance.

Des documents trop grands pour un seul vecteur. Un `tsvector` doit rester sous 1 Mo et ne conserve pas plus de 256 positions par lexème. Un livre entier ou une année de comptes rendus dans une ligne se heurte aux deux limites. Découpez les longs documents en sections ou en pages : chaque résultat pourra aussi pointer vers un endroit plus précis.

Les fragments et les références de pièces. La recherche plein texte compare des lexèmes entiers : « inv » ne trouve rien sur les factures, « invoices » en anglais, et une référence comme `AB-1042-X` peut être découpée par l'analyseur. Conservez un index ordinaire sur les colonnes recherchées par valeur exacte, ou utilisez dans SQLite le segmenteur `trigram`, qui traite chaque séquence de trois caractères comme une unité et permet ainsi la recherche de sous-chaînes.

Un index SQLite qui diverge de sa table. Une table FTS5 déclarée avec `content=` lit son texte dans une autre table. La documentation laisse à l'utilisateur le soin de les maintenir cohérentes. Si elles divergent, les requêtes renvoient des lignes apparemment contradictoires. Créez les déclencheurs d'insertion, de modification et de suppression présentés dans la documentation, dans la même migration que la table virtuelle.

Pour aller plus loin

- Introduction à la recherche plein texte — les définitions de PostgreSQL pour les documents, unités lexicales, lexèmes et configurations, et ce qui manque à `LIKE` et aux expressions régulières.
- Tables et index — interroger une table, choisir entre index d'expression et colonne générée stockée, et pourquoi la configuration doit être nommée explicitement.
- Contrôle de la recherche textuelle — les poids, `websearch_to_tsquery`, `ts_rank` et `ts_rank_cd`, les indicateurs de normalisation et l'avertissement relatif à `ts_headline`.
- Types d'index recommandés pour la recherche textuelle — comparaison de GIN et GiST, et raisons pour lesquelles GiST doit revérifier les lignes.
- Limites — les limites de taille des lexèmes, vecteurs, positions et requêtes.
- Extension SQLite FTS5 — les tables virtuelles, l'opérateur `MATCH`, les segmenteurs intégrés, `bm25()` et les tables à contenu externe.

Glossaire

Choisissez selon vos contraintes réelles

Présentez un jeu de données représentatif, les questions auxquelles il doit répondre et le goulot d'étranglement actuel. Nous pouvons convenir de la manière de mesurer la qualité, le débit et la reprise.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que la recherche plein texte ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/full-text-search/