Engineer

Volltextsuche

Dokumente nach Wörtern suchen und nach Relevanz ordnen

Was es ist

Volltextsuche findet Dokumente, die die Wörter einer Anfrage in beliebiger Wortform enthalten, und kann die besten Treffer zuerst anzeigen. Im Englischen wird full-text search auch ohne Bindestrich als full text search geschrieben. Ein Dokument ist hier das, was die Anwendung als eine durchsuchbare Einheit behandelt: eine E-Mail, ein Artikel, ein Supportticket oder Titel und Text einer Tabellenzeile zusammen.

Der Unterschied zum Mustervergleich liegt in der Arbeit vor dem Vergleich. Zuerst wird der Text in Tokens zerlegt: Wörter, Zahlen, E-Mail-Adressen und andere Klassen. Jedes Token wird anschließend zu einem Lexem normalisiert, einer Form, die Varianten desselben Wortes zusammenführt. Großbuchstaben werden zu Kleinbuchstaben, ein Stemmer entfernt Endungen wie den englischen Plural, und Stoppwörter entfallen: so häufige Wörter, dass sie kein Signal tragen. Das Ergebnis wird samt Position jedes Lexems gespeichert. Eine spätere Bewertungsfunktion kann dadurch Dokumente bevorzugen, in denen die Suchwörter nahe beieinanderstehen. Die Anfrage durchläuft denselben Prozess. Eine Suche nach „payments“ und ein Dokument mit „payment“ treffen sich so in der Mitte.

PostgreSQL baut das in die Datenbank ein. Der Typ `tsvector` hält ein Dokument in normalisierter Form, `tsquery` eine Anfrage, und der Operator `@@` sagt, ob beide zusammenpassen. Eine Textsuchkonfiguration bestimmt die Normalisierung: welcher Parser den Text zerlegt und welche Wörterbücher Tokens in Lexeme verwandeln. Die Standarddistribution enthält Konfigurationen für viele Sprachen, darunter `english`, `danish`, `norwegian` und `swedish`. Der psql-Befehl `\dF` listet die installierten auf. SQLite bietet dieselbe Idee über die Erweiterung FTS5: Eine virtuelle Tabelle hält den Index, der Operator `MATCH` fragt ihn ab.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist Volltextsuche?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Die Abfragen und fehlenden Indizes hinter einem von Jahr zu Jahr schwerfälligeren System finden und die entscheidenden Probleme beheben.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 30 Tage Stromdaten in 180 ms — 180 ms für die Aggregation eines Monats an Zählerständen

30 Tage Stromdaten in 180 ms Kontakt aufnehmen →

Warum es wichtig ist

Die PostgreSQL-Dokumentation erklärt, was den älteren Operatoren fehlt. `LIKE` , `ILIKE` und reguläre Ausdrücke haben keine sprachliche Unterstützung. Eine Suche nach „satisfy“ verfehlt deshalb ein Dokument mit „satisfies“, sofern nicht jemand jede abgeleitete Form ausdrücklich angibt. Sie haben keine Relevanzbewertung: Tausend Treffer kommen ohne nützliche Reihenfolge zurück. Für diese Suchart können sie außerdem keinen gewöhnlichen Index nutzen, sodass jede Abfrage jedes Dokument liest. Ein Muster wie `%cat%` trifft auch „category“ und „concatenate“. Das ist Rauschen statt besserer Trefferabdeckung.

Menschen geben Wörter in ein Suchfeld ein und erwarten die Dokumente dazu. In einem Vertragsarchiv, einem Helpdesk oder einem Produktkatalog macht Volltextsuche den Unterschied zwischen einem vertrauenswürdigen Suchfeld und einem, das nach dem dritten Fehlschlag nicht mehr benutzt wird. Die Bewertung ist so wichtig wie der Abgleich: Wer „late payment“ sucht, möchte das Mahnschreiben oben sehen, nicht hundert Rechnungen, die „payment“ einmal erwähnen.

Die Suche in einer bereits betriebenen Datenbank hat praktische Vorteile. Ihr Index lebt in derselben Transaktion wie die Daten. Eine Zeile wird dadurch mit ihrem Commit durchsuchbar, eine gelöschte Zeile verschwindet im selben Moment aus den Ergebnissen. Filter für Datumswerte, Kunden oder Berechtigungen sind gewöhnliche `WHERE` -Klauseln neben der Textbedingung. Ein separater Suchserver kann mehr, ist aber ein zweites zu befüllendes und mit dem ersten synchron zu haltendes System. Jede Lücke zeigt sich als fehlender Treffer oder als einer, der längst verschwunden sein sollte.

Volltextsuche hat auch eine Grenze, die von Anfang an bekannt sein sollte: Sie vergleicht Wörter, keine Bedeutung. Eine Anfrage nach „car“ findet „automobile“ nur, wenn ein Synonymwörterbuch beide zuordnet. Vektorsuche, die numerische Darstellungen von Bedeutung vergleicht, deckt diesen Fall ab. Beide werden oft kombiniert.

Wie es in der Praxis aussieht

Eine Dokumenttabelle enthält Titel, Text und die Herkunft jedes Dokuments. Eine gespeicherte generierte Spalte hält bei jedem Einfügen und Aktualisieren die durchsuchbare Form aktuell. Ein GIN-Index beschleunigt die Suche:

```
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);
```

Drei Details sind entscheidend. Die Konfiguration ist als `'english'` benannt, statt dem sitzungsbezogenen `default_text_search_config` überlassen zu werden. Die Form von `to_tsvector` mit zwei Argumenten kann PostgreSQL speichern und indizieren. `coalesce` verwandelt einen fehlenden Titel oder Text in eine leere Zeichenfolge, weil ein einziges NULL sonst das ganze Dokument zu NULL machen würde. Und `setweight` kennzeichnet jedes Lexem nach seiner Herkunft: A für den Titel, D für den Text. Die Bewertungsfunktionen verwenden ohne andere Vorgabe die Gewichte {0.1, 0.2, 0.4, 1.0} für D, C, B und A. Ein Suchwort im Titel zählt deshalb zehnmal so viel wie dasselbe Wort im Text.

GIN ist der von der Dokumentation bevorzugte Indextyp für Textsuche: ein invertierter Index mit einem Eintrag je Lexem und einer komprimierten Liste der Zeilen, die es enthalten. GiST funktioniert ebenfalls, ist aber verlustbehaftet und muss die Tabellenzeile prüfen, um falsche Treffer auszuschließen.

Die Anfrage übernimmt die vom Nutzer eingegebenen Wörter unverändert:

```
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` liest Eingaben wie ein Websuchfeld. Einfache Wörter müssen alle vorkommen, jeweils in einer der Formen, die der englische Stemmer zusammenführt. Text in doppelten Anführungszeichen muss als Phrase vorkommen, das Wort „or“ bietet eine Alternative, und ein führendes Minus schließt ein Wort aus. Diese Anfrage lässt daher jedes Dokument weg, das „draft“ erwähnt. Die Funktion erzeugt nie einen Syntaxfehler und ist deshalb die sichere Wahl für unverarbeitete Eingaben. `ts_rank_cd` bewertet nach Überdeckungsdichte, einer Methode aus einem Aufsatz von Clarke, Cormack und Tudhope von 1999, die auch nahe beieinanderstehende Suchwörter belohnt. Die verwandte Funktion `ts_rank` zählt, wie oft die Wörter vorkommen. Beide nehmen ein Normalisierungsflag an. 1 teilt die Bewertung beispielsweise durch eins plus den Logarithmus der Dokumentlänge. Ein langes Dokument gewinnt dann nicht allein wegen seiner Länge.

Derselbe Bedarf zieht sich durch die Arbeit des Unternehmens an Daten für KI-Agenten. Die Seite beschreibt alte PDFs, Scans und E-Mail-Archive, die mit ihrer Quelle in durchsuchbaren Text verwandelt werden, damit sie sich finden und nicht nur speichern lassen. Die Spalte `source` oben erfüllt diesen Zweck: Ein Treffer ist mehr wert, wenn der Leser das Original öffnen kann.

SQLite erreicht dasselbe Ergebnis mit weniger Aufwand. Eine mit `CREATE VIRTUAL TABLE notes USING fts5(title, body)` deklarierte virtuelle Tabelle wird mit `WHERE notes MATCH 'invoice NOT draft'` abgefragt. `ORDER BY bm25(notes)` setzt die besten Treffer zuerst: FTS5 gibt für bessere Treffer niedrigere Zahlenwerte zurück. Die standardmäßige aufsteigende Sortierung ist daher richtig. Der Standardtokenizer heißt `unicode61`. Der Tokenizer `porter` ergänzt Stemming, das laut Dokumentation ausschließlich für Englisch entworfen ist.

Wo es schiefgeht

Ein Index, den die Anfrage nie verwendet. Ein Ausdrucksindex auf `to_tsvector('english', body)` bedient nur Anfragen, die genau diesen Ausdruck wiederholen. Eine Anfrage mit `to_tsvector(body)`, die auf die

Standardkonfiguration vertraut, liest die ganze Tabelle. Speichern Sie den Vektor wie oben in einer generierten Spalte und fragen Sie diese ab. Anfrage und Index können dann nicht auseinanderlaufen.

Ein NULL verbirgt ein Dokument. `to_tsvector(NULL)` gibt NULL zurück. Die Verkettung eines NULL-Titels mit einem Text ergibt ebenfalls NULL. Eine Zeile ohne Titel verschwindet dadurch unbemerkt aus jeder Suche. Umschließen Sie jedes Feld mit `coalesce` .

Rohe Eingaben in `to_tsquery` . `to_tsquery` erwartet seine eigene Operatorsyntax und verzeiht bei der Eingabe weniger als die anderen Umwandlungsfunktionen. Wer eine ungeschlossene Klammer oder ein einzelnes Kaufmanns-Und eingibt, erhält daher eine Fehlerseite. Leiten Sie Nutzereingaben durch `websearch_to_tsquery` oder durch `plainto_tsquery` , wenn gar keine Operatoren gewünscht sind.

Die falsche Sprache. Dänischer Text in der englischen Konfiguration wird nach englischen Regeln gestemmt und nach englischen Stoppwörtern gefiltert. Die Hälfte der Varianten trifft deshalb nie zusammen. Wählen Sie die Konfiguration für die Textsprache. Eine mehrsprachige Sammlung kann den Konfigurationsnamen in einer eigenen Spalte halten und `to_tsvector(config_name, body)` indizieren, solange die Anfragen dieselbe Spalte nennen.

Jeder Treffer wird bewertet. Die Bewertung liest den `tsvector` jeder passenden Zeile. Laut Dokumentation kann das I/O-gebunden und langsam sein. Eine Einwortanfrage, die die halbe Tabelle trifft, lässt die Datenbank erst diese Hälfte bewerten, bevor `LIMIT` zehn Ergebnisse übrig lässt. Grenzen Sie die Kandidaten zuerst mit gewöhnlichen Bedingungen ein, etwa einem Zeitraum oder einem Kunden, und halten Sie den Vektor gespeichert, damit er nicht für jede Zeile neu berechnet wird.

Eine Bewertung wird als Prozentzahl gezeigt. Die Bewertungsfunktionen verwenden keine Informationen über die gesamte Sammlung. Eine Bewertung lässt sich daher nicht in eine faire Angabe wie „87 % relevant“ verwandeln. Das Normalisierungsflag 32 bildet jede Bewertung auf einen Wert zwischen 0 und 1 ab. Die Dokumentation nennt das jedoch eine kosmetische Änderung. Verwenden Sie die Bewertung allein zum Sortieren der Ergebnisse.

Hervorhebungen enthalten Markup. `ts_headline` gibt einen Ausschnitt mit markierten Suchwörtern zurück, genau das, was eine Ergebnisseite braucht. PostgreSQL warnt, die Ausgabe sei „nicht garantiert sicher für die direkte Einbindung in Webseiten“. Ein Dokument mit HTML kann ein Skript in die Seite tragen. Entfernen Sie Markup aus der Eingabe oder führen Sie die Ausgabe durch einen HTML-Sanitizer. Die Funktion arbeitet außerdem auf dem Originaltext, nicht dem Vektor. Rufen Sie sie daher erst nach `LIMIT` für die Zeilen der Ergebnisseite auf, nicht für jeden Treffer.

Dokumente sind für einen Vektor zu groß. Ein `tsvector` muss unter 1 MB bleiben und hält höchstens 256 Positionen pro Lexem. Ein ganzes Buch oder ein Jahr Sitzungsprotokolle in einer Zeile erreicht beide Grenzen. Teilen Sie lange Dokumente in Abschnitte oder Seiten. Jedes Ergebnis erhält dadurch auch ein genaueres Verweisziel.

Fragmente und Teilenummern. Volltextsuche vergleicht ganze Lexeme. „inv“ findet deshalb nichts zu „invoices“, und eine Teilenummer wie `AB-1042-X` kann vom Parser zerlegt werden. Halten Sie einen gewöhnlichen Index auf den Spalten vor, die nach exakten Werten durchsucht werden, oder verwenden Sie in SQLite den Tokenizer `trigram`. Er behandelt jede Folge von drei Zeichen als Token und unterstützt so die Suche nach Teilzeichenfolgen.

Ein SQLite-Index entfernt sich von seiner Tabelle. Eine mit `content=` deklarierte FTS5-Tabelle liest ihren Text aus einer anderen Tabelle. Laut Dokumentation müssen die Nutzer beide konsistent halten. Laufen sie auseinander, geben Anfragen scheinbar widersprüchliche Zeilen zurück. Erzeugen Sie die in der Dokumentation gezeigten Trigger zum Einfügen, Aktualisieren und Löschen in derselben Migration wie die virtuelle Tabelle.

Weiterführende Lektüre

- Einführung in die Volltextsuche — PostgreSQLs Definitionen von Dokumenten, Tokens, Lexemen und Konfigurationen sowie die Lücken von `LIKE` und regulären Ausdrücken.
- Tabellen und Indizes — Tabellensuche, Ausdrucksindizes gegenüber einer gespeicherten generierten Spalte und die erforderliche ausdrückliche Konfigurationsangabe.
- Steuerung der Textsuche — Gewichte, `websearch_to_tsquery`, `ts_rank` und `ts_rank_cd`, Normalisierungsflags und die Warnung zu `ts_headline`.

- Bevorzugte Indextypen für Textsuche — GIN und GiST im Vergleich, einschließlich der nötigen Zeilen-Nachprüfung bei GiST.
- Grenzen — Größenlimits für Lexeme, Vektoren, Positionen und Anfragen.
- SQLite-FTS5-Erweiterung — virtuelle Tabellen, der Operator `MATCH`, eingebaute Tokenizer, `bm25()` und Tabellen mit externen Inhalten.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist Volltextsuche?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/full-text-search/