Engineer

Fuldtekstsøgning

søgning på ord i dokumenter, sorteret efter relevans

Hvad det er

Fuldtekstsøgning finder de dokumenter, der indeholder ordene i en forespørgsel, uanset hvilken bøjningsform ordene står i, og kan sætte de bedste træf øverst. Mange kalder det også fritekstsøgning, og på engelsk hedder det full-text search. Et dokument er her det, programmet behandler som én enhed at søge i: en mail, en artikel, en supportsag eller titel og brødtekst fra én række lagt sammen.

Forskellen fra et simpelt mønstermatch ligger i det arbejde, der sker, før noget bliver sammenlignet. Først deles teksten op i tokens: ord, tal, mailadresser og andre klasser. Derefter gøres hvert token til et leksem, en normaliseret form, der får varianter af det samme ord til at falde sammen. Store bogstaver bliver til små, endelser skæres af, så ordet føres tilbage til sin stamme, og stopord, de ord der er så almindelige, at de intet siger om indholdet, fjernes. Resultatet gemmes sammen med hvert leksems position i teksten, så rangeringen senere kan foretrække dokumenter, hvor søgeordene står tæt. Forespørgslen gennemgår samme behandling, så en søgning på »indtægter« og et dokument, der skriver »indtægt«, mødes på halvvejen.

PostgreSQL har det indbygget. Datatypen `tsvector` rummer et dokument i den normaliserede form, `tsquery` rummer en forespørgsel, og operatoren `@@` afgør, om den ene matcher den anden. En tekstsøgningskonfiguration bestemmer, hvordan normaliseringen foregår: hvilken parser der deler teksten op, og hvilke ordbøger der gør tokens til leksemer. Standardinstallationen har konfigurationer til en lang række sprog, blandt andre `danish`, `norwegian`, `swedish` og `english`, og kommandoen `\dF` i psql viser dem, der er installeret. SQLite tilbyder den samme idé gennem udvidelsen FTS5, hvor en virtuel tabel rummer indekset, og operatoren `MATCH` søger i det.

Brug begrebet i en beslutning

Brug »Hvad er fuldtekstsøgning?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. At finde de forespørgsler og manglende indeks bag et system, der er blevet tungere hvert år, og rette dem, der betyder noget.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. 30 dages eldata på 180 ms — 180 ms til at aggregere en måneds måleraflæsninger

30 dages eldata på 180 ms Kontakt os →

Hvorfor det betyder noget

PostgreSQL-dokumentationen gør rede for, hvad de ældre operatorer mangler. `LIKE`, `ILIKE` og regulære udtryk har ingen sproglig forståelse, så en søgning på »undersøgelse« overser et dokument med »undersøgte«, medmindre nogen skriver alle afledte former ud. De rangerer ikke, så tusind træf kommer tilbage i en tilfældig rækkefølge. Og de kan ikke bruge et almindeligt indeks til den slags søgning, så hver forespørgsel må læse hvert eneste dokument. Et mønster som `%bil%` rammer desuden både »bilag« og »stabil«, og det er støj, ikke bedre dækning.

Folk skriver ord i et søgefelt og forventer at finde de dokumenter, der handler om dem. I et kontraktarkiv, en helpdesk eller et produktkatalog er fuldtekstsøgning forskellen på et søgefelt, man stoler på, og et, man opgiver efter tredje forgæves forsøg. Rangeringen tæller lige så meget som selve matchet: den, der søger efter »sen betaling«, vil have rykkerbrevet øverst og ikke de hundrede fakturaer, der nævner betaling én gang.

Når søgningen bor i den database, man allerede driver, har det en praktisk side. Søgeindekset ændres i samme transaktion som dataene, så en række er søgbar i det øjeblik, den bliver committet, og en slettet række forsvinder fra resultaterne i samme øjeblik. Filtre på dato, kunde eller rettigheder er almindelige `WHERE`-betingelser ved siden af tekstbetingelsen. En separat søgeserver kan mere, men den er et ekstra system, der skal fodres og holdes i takt med det første, og hvert hul imellem dem viser sig som et resultat, der mangler, eller et, der burde være væk.

Fuldtekstsøgning har også en grænse, som er værd at kende fra starten: den matcher ord, ikke betydning. En søgning på »bil« finder ikke »personvogn«, medmindre en synonymordbog kobler dem sammen. Vektorsøgning, der

sammenligner talmæssige repræsentationer af betydning, dækker det tilfælde, og de to bliver ofte kombineret.

Sådan ser det ud i praksis

Tag en tabel med dokumenter, der har en titel, en brødtekst og en angivelse af, hvor hvert dokument kom fra. En gemt genereret kolonne holder den søgbare form ajour ved hver indsættelse og opdatering, og et GIN-indeks gør den hurtig at søge i:

```
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('danish', coalesce(title, '')), 'A')
                || setweight(to_tsvector('danish', coalesce(body, '')), 'D')
               ) STORED
);

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

Tre detaljer bærer det hele. Konfigurationen er nævnt ved navn, `'danish'`, i stedet for at blive overladt til sessionens `default_text_search_config`; det er den form af `to_tsvector` med to argumenter, PostgreSQL kan gemme og indeksere. `coalesce` gør en manglende titel eller brødtekst til en tom streng, for ellers ville én NULL gøre hele dokumentet til NULL. Og `setweight` mærker hvert leksem med, hvor det kom fra: A for titlen, D for brødteksten. Rangeringsfunktionerne bruger vægtene `{0.1, 0.2, 0.4, 1.0}` for D, C, B og A, medmindre man angiver andre, så et søgeord i titlen tæller ti gange så meget som det samme ord i brødteksten.

GIN er den indekstype, dokumentationen anbefaler til tekstsøgning: et inverteret indeks med én post pr. leksem og en komprimeret liste over, hvor det forekommer. GiST kan også bruges, men det er tabsgivende og må slå op i selve tabelrækken for at luge falske træf ud.

Forespørgslen tager imod de ord, brugeren skrev, præcis som brugeren skrev dem:

```
SELECT id, title, source,
       ts_rank_cd(search_vec, query) AS rank
FROM   documents,
       websearch_to_tsquery('danish', 'ubetalt faktura -kladde') AS query
WHERE  search_vec @@ query
ORDER  BY rank DESC
LIMIT  10;
```

`websearch_to_tsquery` læser input, som et søgefelt på nettet gør. Almindelige ord skal alle forekomme, i en hvilken som helst form, den danske ordstammeregel fører tilbage til samme leksem. Tekst i anførselstegn skal forekomme som en sammenhængende frase, ordet »or« giver et alternativ, og et minus foran et ord udelukker det, så forespørgslen her springer alle dokumenter over, der nævner en kladde. Funktionen melder aldrig syntaksfejl, og derfor er den det sikre valg til rå brugerinput. `ts_rank_cd` rangerer efter cover density, en metode fra en artikel af Clarke, Cormack og Tudhope fra 1999, som også belønner søgeord, der står tæt på hinanden. Søsterfunktionen `ts_rank` ser på, hvor ofte ordene forekommer. Begge tager et normaliseringsflag: 1 dividerer for eksempel rangen med én plus logaritmen af dokumentets længde, så et langt dokument ikke vinder alene, fordi det er langt.

Det samme behov går igen i virksomhedens arbejde med virksomhedsdata klar til AI-agenter. Siden beskriver, hvordan gamle PDF'er, scannede dokumenter og mailarkiver bliver til søgbar tekst med kilden gemt ved siden af, så de kan findes og ikke kun arkiveres. Kolonnen `source` ovenfor har netop den opgave: et træf er mere værd, når læseren kan åbne originalen.

SQLite når samme resultat med færre ceremonier. En virtuel tabel oprettet med `CREATE VIRTUAL TABLE notes USING fts5(title, body)` forespørges med `WHERE notes MATCH 'faktura NOT kladde'`, og `ORDER BY bm25(notes)` sætter de bedste træf først: FTS5 giver bedre træf lavere værdier, så den almindelige stigende sortering er den rigtige. Standardtokenizeren hedder `unicode61`; tokenizeren `porter` tilføjer stemming, men ifølge dokumentationen kun til engelsk, så danske tekster får ingen bøjningsformer lagt sammen i FTS5 uden en tokenizer, man selv tilføjer.

Hvor det går galt

Et indeks, forespørgslen aldrig rører. Et udtryksindeks bygget på `to_tsvector('danish', body)` hjælper kun forespørgsler, der gentager præcis det udtryk. En forespørgsel skrevet som `to_tsvector(body)`, der læner sig op ad

standardkonfigurationen, læser hele tabellen. Gem vektoren i en genereret kolonne som ovenfor, og søg i kolonnen; så kan forespørgsel og indeks ikke glide fra hinanden.

Én NULL, der skjuler et dokument. `to_tsvector(NULL)` giver NULL, og en NULL-titel sat sammen med en brødtekst giver også NULL, så en række uden titel forsvinder i stilhed fra alle søgninger. Pak hvert felt ind i `coalesce` .

Rå input i `to_tsquery` **.** `to_tsquery` forventer sin egen operatorsyntaks og er mindre tilgivende over for input end de andre konverteringsfunktioner, så en bruger, der taster en parentes uden partner eller et løst &-tegn, får en fejlside. Send brugerinput gennem `websearch_to_tsquery` , eller gennem `plainto_tsquery` , når der slet ikke skal være operatorer.

Det forkerte sprog. Dansk tekst indekseret med den engelske konfiguration bliver skåret til efter engelske regler og filtreret med engelske stopord, så halvdelen af bøjningsformerne aldrig mødes. Vælg konfigurationen efter tekstens sprog. En samling på flere sprog kan gemme konfigurationens navn i en kolonne for sig og indeksere `to_tsvector(config_name, body)` , så længe forespørgslerne bruger samme kolonne.

Rangering af hvert eneste træf. Rangeringen læser `tsvector` for hver række, der matcher, og dokumentationen advarer om, at det kan være I/O-bundet og langsomt. En forespørgsel på ét ord, der matcher halvdelen af tabellen, får databasen til at rangere halvdelen af tabellen, før `LIMIT` beholder ti. Indsnævr kandidaterne med almindelige betingelser først, for eksempel et datointerval eller en kunde, og hold vektoren gemt, så den ikke beregnes igen for hver række.

En rang vist som procent. Rangeringsfunktionerne bruger ingen viden om samlingen som helhed, så en rang kan ikke omsættes til et retvisende »87 % relevant«. Normaliseringsflaget 32 presser rangen ind mellem nul og én, men dokumentationen kalder det en rent kosmetisk ændring. Brug rangen til at sortere resultaterne og til intet andet.

Fremhævninger med markup. `ts_headline` returnerer et uddrag, hvor søgeordene er markeret, og det er netop, hvad en resultatside vil have. PostgreSQL advarer om, at uddraget ikke er garanteret sikkert at sætte direkte ind i en webside. Et dokument med HTML kan bære et script med ind på siden.

Fjern markup fra input, eller send output gennem en HTML-sanitizer. Funktionen arbejder desuden på den oprindelige tekst og ikke på vektoren, så kald den kun for rækkerne på den viste side, efter `LIMIT`, og ikke for hvert træf.

Dokumenter, der er for store til én vektor. En `tsvector` skal holde sig under 1 MB og gemmer højst 256 positioner pr. leksem. En hel bog eller et års referater i én række rammer begge grænser. Del lange dokumenter op i afsnit eller sider, så hvert resultat også får et mere præcist sted at pege hen.

Brudstykker og varenumre. Fuldtekstsøgning sammenligner hele leksemer, så »fakt« finder intet om fakturaer, og et varenummer som `AB-1042-X` kan blive delt i stykker af parseren. Behold et almindeligt indeks på de kolonner, folk slår op på eksakt værdi, eller brug i SQLite tokenizeren `trigram`, som gør hver sammenhængende række på tre tegn til et token og dermed understøtter søgning på delstrenge.

»Får«, der bliver til »far« i SQLite. Tokenizeren `unicode61` fjerner som standard diakritiske tegn fra latinske bogstaver, og »å« regnes med, så FTS5 gemmer »får« og »far« som samme token og blander dem sammen i træffene. Bogstaverne »æ« og »ø« lader den være i fred. Opret tabellen med `tokenize = 'unicode61 remove_diacritics 0'`, når »å« skal være sit eget bogstav.

Et SQLite-indeks, der glider fra sin tabel. En FTS5-tabel oprettet med `content=` henter sin tekst fra en anden tabel, og dokumentationen overlader det til brugeren at holde de to i overensstemmelse. Kommer de ud af trit, giver forespørgslerne resultater, der ser ud til at modsige hinanden. Opret de triggere til insert, update og delete, som dokumentationen viser, i samme migrering som den virtuelle tabel.

Læs mere

- Introduction to full text search — PostgreSQL's definition af dokumenter, tokens, leksemer og konfigurationer, og hvad `LIKE` og regulære udtryk mangler.
- Tables and Indexes — søgning i en tabel, udtryksindeks over for en gemt genereret kolonne, og hvorfor konfigurationen skal nævnes ved navn.
- Controlling Text Search — vægte, `websearch_to_tsquery`, `ts_rank` og `ts_rank_cd`, normaliseringsflag og advarslen om `ts_headline`.
- Preferred Index Types for Text Search — GIN og GiST sammenlignet, også hvorfor GiST må tjekke rækkerne igen.

- Limitations — størrelsesgrænserne for leksemer, vektorer, positioner og forespørgsler.
- SQLite FTS5 Extension — virtuelle tabeller, operatoren `MATCH`, de indbyggede tokenizere, `bm25()` og external content-tabeller.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring et repræsentativt datasæt, de spørgsmål det skal besvare, og den aktuelle flaskehals. Vi kan aftale målinger af kvalitet, gennemløb og gendannelse.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er fuldtekstsøgning?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/full-text-search/