Engineer

Fritextsökning

sökning i dokument efter deras ord, sorterad efter relevans

Vad det är

Fritextsökning hittar de dokument som innehåller orden i en sökfråga, oavsett vilken böjningsform orden står i, och kan ställa de bästa träffarna först. Det kallas också fulltextsökning, och på engelska full-text search. Ett dokument är här det som programmet behandlar som en enhet att söka i: ett mejl, en artikel, ett supportärende eller titeln och brödtexten från en rad, ihopslagna.

Skillnaden mot en vanlig mönstersökning ligger i arbetet som görs innan något jämförs. Texten delas först upp i tokens: ord, tal, e-postadresser och andra klasser. Varje token normaliseras sedan till ett lexem, en form som är vald så att varianter av samma ord blir lika. Versaler blir gemener, ändelser kapas så att ordet förs tillbaka till sin stam, och stoppord, ord som är så vanliga att de inte säger något om innehållet, sorteras bort. Resultatet sparas tillsammans med positionen för varje lexem, så att rankningen senare kan föredra dokument där sökorden står nära varandra. Sökfrågan går igenom samma behandling, så en sökning på ”räkningar” och ett dokument som skriver ”räkning” möts på halva vägen.

PostgreSQL har det inbyggt. Datatypen `tsvector` håller ett dokument i normaliserad form, `tsquery` håller en sökfråga, och operatorn `@@` avgör om den ena matchar den andra. En textsökningskonfiguration bestämmer hur normaliseringen går till: vilken parser som delar upp texten och vilka ordlistor som gör om tokens till lexem. Standardinstallationen har konfigurationer för många språk, bland dem `swedish`, `danish`, `norwegian` och `english`, och kommandot `\dF` i psql listar de som är installerade. SQLite erbjuder samma idé genom tillägget FTS5, där en virtuell tabell håller indexet och operatorn `MATCH` söker i det.

Använd begreppet i ett beslut

Använd ”Vad är fritextsökning?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Hittar databasfrågorna och de saknade indexen bakom ett system som har blivit tyngre för varje år, och åtgärdar dem som spelar roll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. 30 dagars eldata på 180 ms — 180 ms för att aggregera en månads mätaravläsningar

30 dagars eldata på 180 ms Hör av er →

Varför det spelar roll

PostgreSQL-dokumentationen förklarar vad de äldre operatorerna saknar. `LIKE`, `ILIKE` och reguljära uttryck har inget språkligt stöd, så en sökning på ”betalade” missar ett dokument som skriver ”betalat”, om inte någon skriver ut alla böjningsformer för hand. De rangordnar inte, så tusen träffar kommer tillbaka i en ordning som inte säger något om vilka som är bäst. Och de kan inte använda ett vanligt index för den här sortens sökning, så varje fråga måste läsa varje dokument. Ett mönster som `%lag%` träffar dessutom både ”bilaga” och ”förslag”, och det är brus, inte bättre täckning.

Människor skriver ord i en sökruta och väntar sig att hitta dokumenten som handlar om dem. I ett avtalsarkiv, en kundtjänst eller en produktkatalog är fritextsökning skillnaden mellan en sökruta som folk litar på och en som de slutar använda efter tredje missen. Rankningen betyder lika mycket som själva träffen: den som söker på ”sen betalning” vill ha påminnelsen överst, inte de hundra fakturor som nämner betalning en gång.

Att sökningen bor i databasen man redan driver har också en praktisk sida. Sökindexet uppdateras i samma transaktion som datan, så en rad blir sökbar i samma ögonblick som den committas, och en raderad rad försvinner ur resultaten lika fort. Filter på datum, kund eller behörighet är vanliga `WHERE`-villkor bredvid textvillkoret. En separat söktjänst kan mer, men den är ytterligare ett system att mata och hålla i takt med det första, och varje glapp mellan dem syns som en träff som saknas eller en som borde ha försvunnit.

Fritextsökning har också en gräns som är värd att känna till från början: den matchar ord, inte betydelse. En sökning på ”avtal” hittar inte ”kontrakt” om inte en synonymordlista kopplar ihop dem. Vektorsökning, som jämför numeriska representationer av betydelse, täcker det fallet, och de två kombineras ofta.

Så ser det ut i praktiken

Ta en tabell med dokument som har en titel, en brödtext och en uppgift om var varje dokument kommer ifrån. En lagrad genererad kolumn håller den sökbara formen aktuell vid varje insert och update, och ett GIN-index gör den snabb att söka 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('swedish', coalesce(title, '')), 'A')
                || setweight(to_tsvector('swedish', coalesce(body, '')), 'D')
               ) STORED
);

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

Tre detaljer bär det hela. Konfigurationen anges med namn, `'swedish'`, i stället för att lämnas åt sessionens `default_text_search_config`; det är formen av `to_tsvector` med två argument som PostgreSQL kan lagra och indexera. `coalesce` gör om en saknad titel eller brödtext till en tom sträng, eftersom en enda NULL annars gör hela dokumentet till NULL. Och `setweight` märker varje lexem med var det kom ifrån: A för titeln, D för brödtexten. Rankningsfunktionerna använder vikterna `{0.1, 0.2, 0.4, 1.0}` för D, C, B och A om man inte anger andra, så ett sökord i titeln väger tio gånger så tungt som samma ord i brödtexten.

GIN är den indextyp som dokumentationen rekommenderar för textsökning: ett inverterat index med en post per lexem och en komprimerad lista över de rader där lexemet förekommer. GiST fungerar också, men det är inte exakt och måste kontrollera själva tabellraden för att sålla bort falska träffar.

Frågan tar emot orden som användaren skrev, precis som de skrevs:

```
SELECT id, title, source,
       ts_rank_cd(search_vec, query) AS rank
FROM   documents,
       websearch_to_tsquery('swedish', 'obetald faktura -utkast') AS query
```

```
WHERE  search_vec @@ query
ORDER  BY rank DESC
LIMIT  10;
```

`websearch_to_tsquery` läser indata på samma sätt som en sökruta på webben. Vanliga ord måste alla förekomma, i vilken form som helst som Snowball-algoritmen för svenska för tillbaka till samma lexem. Text inom citattecken måste förekomma som en sammanhängande fras, ordet ”or” ger ett alternativ, och ett minustecken framför ett ord utesluter det, så den här frågan hoppar över alla dokument som nämner ett utkast. Funktionen ger aldrig syntaxfel, och därför är den det säkra valet för rå indata från användare. `ts_rank_cd` rangordnar efter cover density, en metod från en artikel av Clarke, Cormack och Tudhope från 1999 som också belönar sökord som står tätt ihop. Systerfunktionen `ts_rank` tittar på hur ofta orden förekommer. Båda tar en normaliseringsflagga: 1 delar till exempel rangen med ett plus logaritmen av dokumentets längd, så att ett långt dokument inte vinner bara för att det är långt.

Svenskan har dessutom en egen utmaning: sammansatta ord. Snowball-algoritmen bakom konfigurationen `swedish` tar bara bort ändelser och delar aldrig upp ett ord. I projektets egen exempellista blir ”jakthund” stammen ”jakthund”, och på samma sätt träffar en sökning på ”faktura” inte ett dokument som bara skriver ”leverantörsfaktura”. PostgreSQL kan dela upp sammansättningar med en Ispell- eller Hunspell-ordlista vars affixfil markerar vilka ord som får ingå i dem, med raden `compoundwords controlled z`. Dokumentationen visar principen med en norsk ordlista, där `ts_lexize('norwegian_ispell', 'sjokoladefabrikk')` ger `{sjokoladefabrikk,sjokolade,fabrikk}`: hela ordet och båda delarna, så att dokumentet hittas både på helheten och på varje del. PostgreSQL levereras utan sådana ordlistefiler, så de måste hämtas och läggas i katalogen `tsearch_data`. En Ispell-ordlista känner dessutom bara till ett begränsat urval ord, så i konfigurationen bör den följas av en bredare ordlista, till exempel Snowball, som tar hand om resten.

Företagets arbete med företagsdata redo för AI-agenter har samma behov. Sidan beskriver hur gamla PDF:er, inskannade dokument och e-postarkiv blir sökbar text med källan sparad intill, så att de kan hittas och inte bara förvaras. Kolumnen `source` ovan har just den uppgiften: en träff är mer värd när läsaren kan öppna originalet.

SQLite når samma resultat med mindre ceremonier. En virtuell tabell som skapas med `CREATE VIRTUAL TABLE notes USING fts5(title, body)` frågas med `WHERE notes MATCH 'faktura NOT utkast'`, och `ORDER BY bm25(notes)` ställer de bästa träffarna först: FTS5 ger bättre träffar lägre värden, så den vanliga stigande sorteringen är den rätta. Standardtokenizern heter `unicode61`; tokenizern `porter` lägger till stemming, men enligt dokumentationen bara för engelska, så svenska böjningsformer slås inte ihop i FTS5 utan en tokenizer som man lägger till själv.

Var det går fel

Ett index som frågan aldrig rör. Ett uttrycksindex byggt på `to_tsvector('swedish', body)` hjälper bara frågor som upprepar exakt det uttrycket. En fråga skriven som `to_tsvector(body)`, som förlitar sig på standardkonfigurationen, läser hela tabellen. Lagra vektorn i en genererad kolumn som ovan och sök i kolumnen; då kan fråga och index inte glida isär.

En NULL som gömmer ett dokument. `to_tsvector(NULL)` ger NULL, och en NULL-titel ihopslagen med en brödtext ger också NULL, så en rad utan titel försvinner tyst ur alla sökningar. Linda in varje fält i `coalesce`.

Rå indata i `to_tsquery`. `to_tsquery` förväntar sig sin egen operatorsyntax och tål stökig indata sämre än de andra konverteringsfunktionerna, så en användare som skriver en parentes utan motsvarighet eller ett lösryckt &-tecken får en felsida. Skicka användarens indata genom `websearch_to_tsquery`, eller genom `plainto_tsquery` när det inte ska finnas några operatorer alls.

Fel språk. Svensk text som indexeras med den engelska konfigurationen kapas efter engelska regler och filtreras med engelska stoppord, så hälften av böjningsformerna möts aldrig. Välj konfiguration efter textens språk. En samling på flera språk kan lagra konfigurationens namn i en egen kolumn och indexera `to_tsvector(config_name, body)`, så länge frågorna använder samma kolumn.

En ordlista som ändrats bakom ryggen på sessionerna. En databassession läser sina ordlistefiler första gången de används och sedan inte igen. Den som lägger in nya ord i en Ispell-ordlista ser dem därför bara i nya anslutningar, och en anslutningspool kan hålla kvar de gamla länge. Kör `ALTER TEXT SEARCH DICTIONARY` på ordlistan, gärna som en tom ändring där ingen parameter får nytt värde, så läser de befintliga sessionerna filerna på nytt. En tesaurus används när dokumenten indexeras, så efter en ändring i den måste de lagrade vektorerna byggas om.

Rankning av varenda träff. Rankningen läser `tsvector` för varje rad som matchar, och dokumentationen varnar för att det kan vara I/O-bundet och långsamt. En fråga på ett ord som matchar halva tabellen får databasen att ranka halva tabellen innan `LIMIT` behåller tio. Smalna av kandidaterna med vanliga villkor först, till exempel ett datumintervall eller en kund, och håll vektorn lagrad så att den inte beräknas på nytt för varje rad.

En rang som visas i procent. Rankningsfunktionerna använder ingen kunskap om samlingen som helhet, så en rang kan inte göras om till ett rättvist ”87 % relevant”. Normaliseringsflaggan 32 pressar in rangen mellan noll och ett, men dokumentationen kallar det en rent kosmetisk ändring. Använd rangen för att sortera resultaten och till inget annat.

Markeringar som bär med sig markup. `ts_headline` returnerar ett utdrag där sökorden är markerade, och det är just vad en resultatsida vill ha. PostgreSQL varnar för att utdraget inte garanterat är säkert att lägga direkt i en webbsida. Ett dokument med HTML kan föra in ett skript på sidan. Ta bort markup ur indata eller skicka utdata genom en HTML-sanerare. Funktionen arbetar dessutom på originaltexten och inte på vektorn, så anropa den bara för raderna på sidan som visas, efter `LIMIT`, och inte för varje träff.

Dokument som är för stora för en vektor. En `tsvector` måste hålla sig under 1 MB och sparar högst 256 positioner per lexem. En hel bok eller ett års mötesprotokoll i en rad slår i båda gränserna. Dela upp långa dokument i avsnitt eller sidor, så att varje resultat också får en mer exakt plats att peka på.

Ordfragment och artikelnummer. Fritextsökning jämför hela lexem, så ”fakt” ger inga träffar på fakturor, och ett artikelnummer som `AB-1042-X` kan delas i bitar av parsern. Behåll ett vanligt index på kolumnerna som folk slår upp på exakt värde, eller använd tokenizern `trigram` i SQLite, som gör varje sammanhängande följd av tre tecken till en token och därmed klarar sökning på delsträngar.

”Får” som blir ”far” i SQLite. Tokenizern `unicode61` tar som standard bort diakritiska tecken från latinska bokstäver, och å, ä och ö räknas dit, så FTS5 sparar ”får” och ”far”, ”för” och ”for” som samma token och blandar ihop dem i träffarna. Skapa tabellen med `tokenize = 'unicode61 remove_diacritics 0'` när de svenska bokstäverna ska få vara egna bokstäver.

Ett SQLite-index som glider ifrån sin tabell. En FTS5-tabell som skapas med `content=` hämtar sin text från en annan tabell, och dokumentationen överlåter åt användaren att hålla de två i samklang. Glider de isär ger frågorna resultat som verkar motsäga varandra. Skapa triggrarna för insert, update och delete som dokumentationen visar, i samma migrering som den virtuella tabellen.

Läs mer

- Introduction to full text search — PostgreSQLs definition av dokument, tokens, lexem och konfigurationer, och vad `LIKE` och reguljära uttryck saknar.
- Tables and Indexes — sökning i en tabell, uttrycksindex jämfört med en lagrad genererad kolumn, och varför konfigurationen måste anges med namn.
- Controlling Text Search — vikter, `websearch_to_tsquery`, `ts_rank` och `ts_rank_cd`, normaliseringsflaggor och varningen om `ts_headline`.
- Dictionaries — Snowball-, Ispell- och Hunspell-ordlistor, uppdelning av sammansatta ord och när sessioner läser ordlistefilerna.
- Swedish stemming algorithm — Snowball-projektets beskrivning av den svenska algoritmen, med ändelserna den tar bort och en exempellista med ord och stammar.
- SQLite FTS5 Extension — virtuella tabeller, operatorn `MATCH`, de inbyggda tokenizerna med `remove_diacritics`, `bm25()` och tabeller med externt innehåll.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med ett representativt dataset, frågorna det ska besvara och dagens flaskhals. Vi kan avtala mått på kvalitet, genomströmning och återställning.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är fritextsökning?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/full-text-search/