Engineer

Повнотекстовий пошук

пошук документів за словами, упорядкований за релевантністю

Що це таке

Повнотекстовий пошук знаходить документи, що містять слова запиту, у будь-якій формі, якої ці слова набули, і вміє ставити найкращі збіги першими. Англійською це full-text search, і саме так називаються відповідні розділи документації PostgreSQL та SQLite. Документом тут вважають те, що застосунок шукає як одне ціле: лист, статтю, звернення до підтримки чи заголовок і текст одного рядка таблиці разом.

Від порівняння за шаблоном його відрізняє робота, яку виконують ще до будь-якого порівняння. Спершу текст розбивають на токени: слова, числа, адреси електронної пошти та інші класи. Потім кожен токен нормалізують у лексему, тобто форму, обрану так, щоб варіанти одного слова збігалися. Великі літери стають малими, стемер відтинає закінчення, а стоп-слова, такі часті, що не несуть жодного сигналу, відкидаються. Результат зберігають разом із позицією кожної лексеми, і згодом функція ранжування може віддати перевагу документам, де слова запиту стоять поруч. Запит проходить той самий шлях, тож пошук за словом «рахунки» і документ, у якому написано «рахунок», зустрічаються посередині, якщо словник знає мову тексту.

PostgreSQL вбудовує все це в саму базу даних. Тип `tsvector` зберігає документ у нормалізованому вигляді, тип `tsquery` зберігає запит, а оператор `@@` перевіряє, чи запит збігається з документом. Конфігурація текстового пошуку визначає, як саме відбувається нормалізація: який парсер ділить текст і які словники перетворюють токени на лексеми. Стандартний дистрибутив постачає конфігурації для багатьох мов, серед них `english`, `danish`, `norwegian` і `swedish`, а команда psql `\dF` показує встановлені. Української конфігурації серед них у PostgreSQL 18, поточній версії

документації, немає, і про це краще знати заздалегідь: нижче показано, як її зібрати. SQLite пропонує ту саму ідею через розширення FTS5, де індекс тримає віртуальна таблиця, а запити до неї роблять оператором `MATCH`.

Застосуйте поняття до рішення

Застосуйте «Що таке повнотекстовий пошук?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Пошук запитів і відсутніх індексів у системі, що рік за роком важчає, та виправлення тих, що справді мають значення.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. 30 днів даних про електроенергію за 180 мс — 180 ms на агрегацію місяця показань лічильників

30 днів даних про електроенергію за 180 мс Зв'язатися з нами →

Чому це важливо

Документація PostgreSQL прямо пояснює, чого бракує старшим операторам. `LIKE`, `ILIKE` і регулярні вирази не мають лінгвістичної підтримки, тож запит «договір» пропускає документ зі словом «договору» чи «договорі», якщо хтось не випише кожну форму вручну. В українській це відчутніше, ніж в англійській: слово змінює не лише закінчення, а часом і голосну в корені. Ці оператори не ранжують результати, тож тисяча збігів повертається без корисного порядку. І вони не можуть використати звичайний індекс для такого пошуку, тож кожен запит читає кожен документ. Шаблон `%мова%` до того ж збігається з «умова» і «самовар», а це вже шум, а не повнота.

Люди вводять слова в поле пошуку й чекають документів про ці слова. В архіві договорів, службі підтримки чи каталозі товарів повнотекстовий пошук відділяє поле пошуку, якому довіряють, від поля, яким перестають користуватися після третьої марної спроби. Ранжування важить не менше, ніж сам збіг: той, хто шукає «прострочена оплата», хоче бачити вгорі лист-нагадування, а не сотню рахунків, де слово «оплата» трапляється по разу.

Те, що пошук живе в базі даних, яку ви вже маєте, дає практичну перевагу. Пошуковий індекс змінюється в тій самій транзакції, що й дані, тож рядок можна знайти, щойно його зафіксовано, а видалений рядок зникає з результатів тієї ж миті. Фільтри за датою, клієнтом чи правами доступу стають звичайними умовами `WHERE` поруч із текстовою. Окремий пошуковий сервер уміє більше, але це друга система, яку треба наповнювати й тримати в узгодженні з першою, і кожен розрив між ними перетворюється на результат, якого бракує, або на такий, що мав би вже зникнути.

Повнотекстовий пошук має й межу, яку варто знати з самого початку: він порівнює слова, а не зміст. Запит «автомобіль» не приведе до документа про «легковик», якщо словник синонімів не зіставить одне з одним. Векторний пошук, що порівнює числові представлення змісту, закриває саме цей випадок, і два підходи часто поєднують.

Як це виглядає на практиці

Почнімо з мови. Вбудована конфігурація `simple` зводить слова до нижнього регістру, але закінчень не відтинає, тож «договір» і «договору» залишаються для неї двома різними лексемами. Задокументований шлях до кращого результату — словник Hunspell, підключений через шаблон `ispell`. Файлів Ispell PostgreSQL не постачає, тому словник беруть окремо: файли `.aff` і `.dic` перейменовують на `.affix` і `.dict`, за потреби перекодовують в UTF-8 і кладуть у каталог `tsearch_data` всередині `$SHAREDIR`, шлях до якого показує `pg_config --sharedir`. Після цього конфігурацію можна оголосити:

```
CREATE TEXT SEARCH DICTIONARY ukrainian_hunspell (
    TEMPLATE = ispell, DictFile = uk_ua, AffFile = uk_ua
);

CREATE TEXT SEARCH CONFIGURATION ukrainian (COPY = pg_catalog.simple);

ALTER TEXT SEARCH CONFIGURATION ukrainian
    ALTER MAPPING FOR word, hword, hword_part
    WITH ukrainian_hunspell, simple;
```

`uk_ua` тут — базова назва файлів `uk_ua.dict` і `uk_ua.affix`, яку ви обираєте самі, коли перейменовуєте їх. Типи токенів `word`, `hword` і `hword_part` охоплюють слова з літер поза ASCII, зокрема кирилицю, разом зі словами через дефіс і їхніми частинами. Порядок словників має значення. Документація попереджає, що словник Ispell зазвичай знає обмежений набір слів, тож за ним має стояти ширший, а словник `simple` у типовому режимі приймає кожен токен і тому корисний лише в кінці списку. Так слово, якого немає в Hunspell, принаймні потрапить до індексу в нижньому регістрі. Перевірити результат допомагають `ts_lexize('ukrainian_hunspell', 'договору')`, що показує лексеми одного слова, і `ts_debug('ukrainian', 'текст')`, що показує, як конфігурація розбирає цілий рядок.

Далі таблиця документів із заголовком, текстом і місцем, звідки кожен документ узявся. Збережений згенерований стовпець оновлює пошукову форму під час кожного вставлення й оновлення, а індекс GIN робить пошук швидким:

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

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

Тут працюють три деталі. Конфігурацію названо явно, `'ukrainian'`, а не залишено на `default_text_search_config` сеансу: саме форму `to_tsvector` із двома аргументами PostgreSQL може зберігати й індексувати. `coalesce` перетворює відсутній заголовок чи текст на порожній рядок, бо один NULL інакше зробив би NULL-ом увесь документ. А `setweight` позначає кожну лексему за місцем походження: A для заголовка, D для тексту. Якщо не вказати інакше, функції ранжування беруть для D, C, B і A ваги `{0.1, 0.2, 0.4, 1.0}`, тож слово запиту в заголовку важить удесятеро більше, ніж те саме слово в тексті.

Для текстового пошуку документація рекомендує індекс GIN: це інвертований індекс з окремим записом для кожної лексеми й стисненим списком рядків, які її містять. GiST теж підходить, але зберігає інформацію з втратами, тому для відсіювання хибних збігів треба перевіряти рядок таблиці.

Запит бере слова так, як їх увів користувач:

```
SELECT id, title, source,
       ts_rank_cd(search_vec, query) AS rank
FROM   documents,
       websearch_to_tsquery('ukrainian',
                            'неоплачений рахунок -чернетка') AS query
```

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

`websearch_to_tsquery` читає введення так, як його читає поле пошуку в браузері. Звичайні слова мають трапитися всі, у будь-якій формі, яку словник зводить до однієї лексеми, текст у подвійних лапках має трапитися як фраза, слово «or» пропонує альтернативу, а мінус перед словом виключає його, тож цей запит відкидає кожен документ, де згадано чернетку. Функція ніколи не повертає синтаксичної помилки, тому вона безпечна для сирого введення. `ts_rank_cd` ранжує за щільністю покриття, методом зі статті Кларка, Кормака й Тудоупа 1999 року, що також винагороджує слова запиту, які стоять поруч. Сусідня функція `ts_rank` рахує, як часто слова трапляються. Обидві приймають прапорець нормалізації: 1, наприклад, ділить ранг на одиницю плюс логарифм довжини документа, щоб довгий документ не вигравав лише тому, що він довгий.

Та сама потреба проходить крізь роботу компанії з даними для ШІ-агентів. Сторінка описує, як старі PDF, скани й поштові архіви перетворюються на текст, у якому можна шукати, з посиланням на джерело, щоб їх можна було знайти, а не лише зберігати. Стовпець `source` у таблиці вище виконує саме це завдання: збіг вартий більшого, коли читач може відкрити оригінал.

SQLite доходить до подібного результату з меншими церемоніями. Віртуальну таблицю, оголошену як `CREATE VIRTUAL TABLE notes USING fts5(title, body)`, запитують умовою `WHERE notes MATCH 'рахунок NOT чернетка'`, а `ORDER BY bm25(notes)` ставить найкращі збіги першими: FTS5 повертає кращі збіги як менші числа, тож звичайний порядок за зростанням і є правильним. Типовий токенізатор `unicode61` зводить кирилицю до нижнього регістру, але не стемить, тож «договір», «договору» і «договорі» для нього три різні терміни. Токенізатор `porter`, що додає стемінг, за документацією призначений лише для англійської. Для українського тексту в SQLite практичніший токенізатор `trigram`: він індексує кожну послідовність із трьох символів, тож запит `MATCH 'догов'` знаходить усі три форми незалежно від регістру. Ціна — більший індекс, а підрядок з одного чи двох символів, за документацією, рядків не повертає.

Де виникають проблеми

Індекс, якого запит не торкається. Індекс за виразом, побудований на `to_tsvector('ukrainian', body)`, обслуговує лише запити, що повторюють саме цей вираз. Запит, записаний як `to_tsvector(body)`, покладається на типову конфігурацію сеансу й читає всю таблицю. Зберігайте вектор у згенерованому стовпці, як вище, і звертайтеся до стовпця: тоді запит не може розійтися з індексом.

Один NULL, що ховає документ. `to_tsvector(NULL)` повертає NULL, а конкатенація відсутнього заголовка (NULL) з текстом теж дає NULL, тож рядок без заголовка мовчки випадає з кожного пошуку. Обгортайте кожне поле в `coalesce`.

Сире введення в `to_tsquery`. `to_tsquery` чекає власного синтаксису операторів і прощає менше, ніж інші функції перетворення, тож користувач, що ввів незакриту дужку чи зайвий амперсанд, отримує сторінку з помилкою. Передавайте введений текст через `websearch_to_tsquery` або через `plainto_tsquery`, коли оператори не потрібні зовсім.

Не та мова. Текст однією мовою, проіндексований конфігурацією для іншої, стемлять чужі правила й фільтрують чужі стоп-слова, тож половина форм так і не зустрічається. Обирайте конфігурацію за мовою тексту. Колекція кількома мовами може зберігати назву конфігурації в окремому стовпці й індексувати `to_tsvector(config_name, body)`, якщо запити звертаються до того самого стовпця.

Словник змінився, а сеанси про це не знають. Сеанс читає файли словника лише раз, коли вперше його використовує, тож після оновлення `uk_ua.dict` чи файлу синонімів відкриті з'єднання й далі працюють зі старою версією. Виконайте `ALTER TEXT SEARCH DICTIONARY` для цього словника; документація допускає навіть формальну команду, що не змінює параметрів, і сеанси перечитають файли. Пам'ятайте й про вже збережені вектори: їх обчислено старим словником, тож якщо зміна стосується того, до якої лексеми зводиться слово, старі рядки треба перезаписати, інакше вони не зустрінуться з новими запитами.

Апостроф, що розрізає слово. У словах «об'єкт», «м'ясо» чи «зв'язок» апостроф належить слову, але токенізатори часто вважають його розділовим знаком. SQLite FTS5 з `unicode61` ділить «об'єкт» на «об» і «єкт»,

коли апостроф набрано як `'` чи `’`, і тримає слово цілим лише з `’` (U+02BC), який Unicode відносить до літер. Документ, набраний з `’`, і запит із `'` так і не зустрінуться. Парсер PostgreSQL визначає, що є літерою, за налаштуванням локалі `lc_ctype`, тож перевірте свою базу через `ts_debug`. Зводьте всі апострофи до одного символу однією функцією і в тексті перед індексацією, і в запиті.

Ранжування кожного збігу. Ранжування читає `tsvector` кожного рядка, що збігся, і документація попереджає, що це може впиратися у введення-виведення й працювати повільно. Запит з одного слова, що збігається з половиною таблиці, змушує базу ранжувати половину таблиці, перш ніж `LIMIT` залишить десять рядків. Звужуйте кандидатів звичайними умовами, як-от діапазоном дат чи клієнтом, і тримайте вектор збереженим, щоб його не обчислювали заново для кожного рядка.

Ранг, показаний як відсоток. Функції ранжування не бачать колекції загалом, тож ранг неможливо чесно перетворити на «релевантність 87%». Прапорець нормалізації 32 стискає ранги в проміжок від нуля до одиниці, але документація називає це косметичною зміною. Використовуйте ранг, щоб упорядкувати результати, і ні для чого більше.

Підсвічування, що несе розмітку. `ts_headline` повертає уривок із позначеними словами запиту, а саме цього й потребує сторінка результатів. Проте PostgreSQL попереджає, що її вивід не гарантовано безпечний для прямого вставлення у вебсторінку. Документ, що містить HTML, може занести на сторінку скрипт. Прибирайте розмітку з вхідного тексту або пропускайте вивід через HTML-санітайзер. Функція працює з оригінальним текстом, а не з вектором, тож викликайте її лише для рядків на сторінці, після `LIMIT`, а не для кожного збігу.

Документи, завеликі для одного вектора. `tsvector` має бути меншим за 1 МБ і зберігає не більше 256 позицій на лексему. Ціла книга чи річний архів протоколів в одному рядку впирається в обидва обмеження. Діліть довгі документи на розділи чи сторінки: це ще й дає кожному результату точніше місце, на яке він вказує.

Фрагменти й артикули. Повнотекстовий пошук порівнює цілі лексеми, тож запит «рах» рахунків не знаходить, а артикул на кшталт `AB-1042-X` парсер може розбити на частини. Тримайте звичайний індекс на стовпцях, за

якими шукають точне значення, а в SQLite для пошуку підрядків беріть токенізатор `trigram`.

Індекс SQLite, що розходиться зі своєю таблицею. Таблиця FTS5, оголошена з `content=`, читає текст з іншої таблиці, і документація залишає узгодженість обох на користувача. Коли вони розходяться, запити повертають рядки, що ніби суперечать одне одному. Створіть тригери на вставлення, оновлення й видалення, які показує документація, у тій самій міграції, що й віртуальну таблицю.

Що почитати далі

- Introduction to full text search — визначення документів, токенів, лексем і конфігурацій у PostgreSQL і пояснення, чого бракує `LIKE` та регулярним виразам.
- Tables and Indexes — пошук у таблиці, індекс за виразом проти збереженого згенерованого стовпця і чому конфігурацію треба називати явно.
- Controlling Text Search — ваги, `websearch_to_tsquery`, `ts_rank` і `ts_rank_cd`, прапорці нормалізації та застереження щодо `ts_headline`.
- Dictionaries — шаблони словників, Ispell і Hunspell, де лежать файли словників і коли сеанс їх перечитує.
- Limitations — обмеження розміру лексем, векторів, позицій і запитів.
- SQLite FTS5 Extension — віртуальні таблиці, оператор `MATCH`, токенізатори `unicode61`, `porter` і `trigram`, `bm25()` і таблиці із зовнішнім вмістом.

Словник

Обирайте за фактичними умовами

Надайте репрезентативний набір даних, запитання до нього та поточне вузьке місце. Узгодимо вимірювання якості, пропускної здатності та відновлення.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке повнотекстовий пошук?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/full-text-search/