Engineer

Búsqueda de texto completo

buscar documentos por palabras y ordenarlos por relevancia

Qué es

La búsqueda de texto completo encuentra los documentos que contienen las palabras de una consulta, en cualquiera de sus formas, y puede colocar primero las mejores coincidencias. En inglés también se escribe full text search, sin guion. Un documento es aquí lo que la aplicación trata como unidad de búsqueda: un correo electrónico, un artículo, una incidencia de soporte o el título y el cuerpo de una fila juntos.

La diferencia con una búsqueda por patrones está en el trabajo previo a cualquier comparación. Primero se divide el texto en tokens: palabras, números, direcciones de correo y otras clases. Después se normaliza cada token como lexema, una forma elegida para que las variantes de una misma palabra coincidan. Las mayúsculas se convierten en minúsculas, un algoritmo de reducción a la raíz elimina terminaciones como el plural inglés y se descartan las palabras vacías, tan comunes que no aportan información. El resultado se guarda con la posición de cada lexema; así, una función de relevancia puede favorecer después documentos donde las palabras de la consulta estén próximas. La consulta pasa por el mismo proceso: una búsqueda de «payments» y un documento que dice «payment» acaban coincidiendo.

PostgreSQL lo incorpora en la base de datos. Su tipo `tsvector` guarda un documento normalizado; `tsquery`, una consulta; y el operador `@@` indica si uno coincide con la otra. Una configuración de búsqueda de texto determina la normalización: qué analizador divide el texto y qué diccionarios convierten tokens en lexemas. La distribución estándar incluye configuraciones para muchos idiomas, entre ellos `english`, `danish`, `norwegian` y `swedish`; el comando de psql `\dF` enumera las instaladas. SQLite ofrece la misma idea mediante su extensión FTS5: una tabla virtual guarda el índice y el operador `MATCH` lo consulta.

Utilice el concepto en una decisión

Aplique «¿Qué es la búsqueda de texto completo?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Detectar las consultas y los índices que faltan en un sistema cada año más pesado y corregir lo que realmente importa.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 30 días de datos eléctricos en 180 ms — 180 ms para agregar un mes de lecturas de contadores

30 días de datos eléctricos en 180 ms Póngase en contacto →

Por qué importa

La documentación de PostgreSQL explica qué les falta a los operadores anteriores. `LIKE` , `ILIKE` y las expresiones regulares carecen de soporte lingüístico: buscar «satisfy» no encuentra un documento que diga «satisfies», salvo que alguien enumere todas las formas derivadas. No ordenan por relevancia, por lo que mil coincidencias vuelven sin un orden útil. Tampoco pueden usar un índice ordinario para este tipo de búsqueda: cada consulta lee todos los documentos. Un patrón como `%cat%` también coincide con «category» y «concatenate»: eso añade ruido, no recupera más documentos pertinentes.

Quien escribe palabras en un buscador espera encontrar documentos sobre ellas. En un archivo de contratos, un servicio de soporte o un catálogo de productos, la búsqueda de texto completo marca la diferencia entre un buscador en el que se confía y uno que se abandona tras el tercer fallo. El orden por relevancia importa tanto como las coincidencias: quien busca «late payment» quiere ver primero la carta de reclamación, no las cien facturas que mencionan «payment» una sola vez.

Integrarla en la base de datos ya utilizada tiene una ventaja práctica. El índice de búsqueda comparte transacción con los datos: una fila se puede buscar en cuanto se confirma y una fila eliminada desaparece de los resultados en ese mismo momento. Los filtros por fechas, clientes o permisos son cláusulas `WHERE` ordinarias junto a la condición de texto. Un servidor de búsqueda separado puede hacer más, pero es otro sistema que hay que alimentar y mantener sincronizado con el primero. Cada desfase produce un resultado ausente o uno que ya debería haber desaparecido.

La búsqueda de texto completo también tiene un límite que conviene conocer desde el principio: compara palabras, no significado. Una consulta de «car» no encuentra «automobile» salvo que un diccionario de sinónimos relacione ambas.

La búsqueda vectorial, que compara representaciones numéricas del significado, cubre ese caso; ambas suelen combinarse.

Cómo se aplica en la práctica

Partamos de una tabla de documentos con título, cuerpo y procedencia. Una columna generada almacenada mantiene actualizada la forma que se busca en cada inserción y actualización; un índice GIN agiliza las búsquedas:

```
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);
```

Tres detalles son decisivos. Se indica explícitamente la configuración, `'english'`, en lugar de dejarla al `default_text_search_config` de la sesión; la forma de dos argumentos de `to_tsvector` es la que PostgreSQL puede almacenar e indexar. `coalesce` convierte un título o cuerpo ausente en una cadena vacía, pues un solo NULL haría que todo el documento fuera NULL. Y `setweight` etiqueta cada lexema según su origen: A para el título y D para el cuerpo. Las funciones de relevancia usan los pesos {0.1, 0.2, 0.4, 1.0} para D, C, B y A salvo indicación contraria, de modo que una palabra de la consulta en el título pesa diez veces más que esa misma palabra en el cuerpo.

GIN es el tipo de índice que la documentación prefiere para búsqueda de texto: un índice invertido con una entrada por lexema y una lista comprimida de las filas que lo contienen. GiST también sirve, pero permite falsos positivos y debe comprobar la fila de la tabla para descartarlos.

La consulta recibe las palabras tal como las escribió el usuario:

```
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` interpreta la entrada como un buscador web. Todas las palabras simples deben aparecer, en cualquiera de las formas que agrupe el algoritmo de reducción a la raíz del inglés; el texto entre comillas dobles debe aparecer como frase; la palabra «or» ofrece una alternativa y un signo menos delante excluye una palabra. Así, esta consulta descarta todo documento que mencione «draft». La función nunca produce un error de sintaxis, por lo que es la opción segura para entradas sin procesar. `ts_rank_cd` ordena por densidad de cobertura, un método de un artículo de 1999 de Clarke, Cormack y Tudhope que también premia la proximidad entre las palabras de la consulta. Su función hermana `ts_rank` cuenta cuántas veces aparecen. Ambas aceptan una opción de normalización: 1, por ejemplo, divide la puntuación por uno más el logaritmo de la longitud del documento, para que uno largo no gane solo por ser largo.

Esa misma necesidad está presente en el trabajo de la empresa con datos empresariales para agentes de IA. La página describe cómo se convierten antiguos PDF, documentos escaneados y archivos de correo en texto consultable, conservando su procedencia junto a ellos, para poder encontrarlos y no solo almacenarlos. La columna `source` anterior cumple esa tarea: una coincidencia vale más cuando se puede abrir el original.

SQLite logra el mismo resultado con menos preparación. Una tabla virtual declarada con `CREATE VIRTUAL TABLE notes USING fts5(title, body)` se consulta con `WHERE notes MATCH 'invoice NOT draft'` ; `ORDER BY bm25(notes)` coloca primero las mejores coincidencias. FTS5 las devuelve con valores numéricos menores, por lo que el orden ascendente predeterminado es el correcto. El tokenizador predeterminado es `unicode61` ; el tokenizador `porter` añade reducción a la raíz y, según su documentación, está diseñado solo para el inglés.

Dónde surgen los problemas

Un índice que la consulta nunca usa. Un índice de expresión sobre `to_tsvector('english', body)` solo sirve para consultas que repitan exactamente esa expresión. Una consulta con `to_tsvector(body)` , que depende de la

configuración predeterminada, lee toda la tabla. Guarde el vector en una columna generada, como arriba, y consulte esa columna: así la consulta no puede apartarse del índice.

Un NULL que oculta un documento. `to_tsvector(NULL)` devuelve NULL; concatenar un título NULL con un cuerpo da NULL, por lo que una fila sin título desaparece silenciosamente de todas las búsquedas. Envuelva cada campo en `coalesce` .

Entrada sin procesar en `to_tsquery` . `to_tsquery` espera su propia sintaxis de operadores y tolera menos errores de entrada que las otras funciones de conversión. Quien escriba un paréntesis sin pareja o un ampersand suelto verá una página de error. Pase la entrada por `websearch_to_tsquery` , o por `plainto_tsquery` cuando no se quieran operadores en absoluto.

El idioma equivocado. Un texto danés indexado con la configuración inglesa se reduce según las reglas inglesas y se filtra con sus palabras vacías: la mitad de las variantes no llega a coincidir. Elija la configuración del idioma del texto. Una colección multilingüe puede guardar el nombre de la configuración en su propia columna e indexar `to_tsvector(config_name, body)` , siempre que las consultas indiquen esa misma columna.

Calcular la relevancia de cada coincidencia. El cálculo lee el `tsvector` de cada fila coincidente; la documentación advierte que puede quedar limitado por la E/S y ser lento. Una consulta de una sola palabra que coincida con media tabla hace que la base de datos puntúe esa mitad antes de que `LIMIT` conserve diez filas. Reduzca primero los candidatos con condiciones ordinarias, como un intervalo de fechas o un cliente, y mantenga el vector almacenado para no recalcularlo en cada fila.

Una puntuación presentada como porcentaje. Las funciones de relevancia no usan información de la colección en su conjunto, por lo que una puntuación no se puede convertir honradamente en «87 % de relevancia». La opción de normalización 32 transforma cada puntuación en un valor entre 0 y 1, pero la documentación llama cosmético a ese cambio. Use la puntuación solo para ordenar resultados.

Fragmentos destacados que arrastran marcado. `ts_headline` devuelve un extracto con las palabras de la consulta destacadas, justo lo que necesita una página de resultados. PostgreSQL advierte que su salida «no tiene garantías de ser segura para incluirla directamente en páginas web». Un documento con HTML puede introducir un script en la página. Elimine el marcado de la entrada o pase la salida por un saneador HTML. Además, la función trabaja sobre el texto original, no sobre el vector: llámela solo para las filas de la página, después de `LIMIT`, no para todas las coincidencias.

Documentos demasiado grandes para un vector. Un `tsvector` debe ocupar menos de 1 MB y guarda como máximo 256 posiciones por lexema. Un libro entero o un año de actas en una fila tropieza con ambos límites. Divida los documentos largos en secciones o páginas: cada resultado también podrá señalar un lugar más preciso.

Fragmentos y números de pieza. La búsqueda de texto completo compara lexemas enteros: «inv» no encuentra nada sobre «invoices», y el analizador puede dividir un número de pieza como `AB-1042-X`. Mantenga un índice ordinario en las columnas que se buscan por valor exacto o, en SQLite, use el tokenizador `trigram`, que toma cada secuencia de tres caracteres como token y permite buscar subcadenas.

Un índice SQLite que deja de coincidir con su tabla. Una tabla FTS5 declarada con `content=` lee el texto de otra tabla; la documentación deja al usuario mantener su coherencia. Si se desincronizan, las consultas devuelven filas que parecen contradecirse. Cree los disparadores de inserción, actualización y eliminación que muestra la documentación en la misma migración que la tabla virtual.

Lecturas adicionales

- Introducción a la búsqueda de texto completo — La definición de documentos, tokens, lexemas y configuraciones de PostgreSQL, y las carencias de `LIKE` y las expresiones regulares.
- Tablas e índices — Cómo buscar en una tabla, índices de expresión frente a una columna generada almacenada y por qué hay que indicar explícitamente la configuración.
- Control de la búsqueda de texto — Pesos, `websearch_to_tsquery`, `ts_rank` y `ts_rank_cd`, opciones de normalización y la advertencia de `ts_headline`.

- Tipos de índice preferidos para búsqueda de texto — Comparación de GIN y GiST, incluido por qué GiST debe volver a comprobar las filas.
- Límites — Los límites de tamaño de lexemas, vectores, posiciones y consultas.
- Extensión SQLite FTS5 — Tablas virtuales, operador `MATCH`, tokenizadores incluidos, `bm25()` y tablas de contenido externo.

Glosario

Elija según sus restricciones reales

Traiga un conjunto de datos representativo, las preguntas que debe responder y el cuello de botella actual. Podemos acordar cómo medir la calidad, la capacidad de procesamiento y la recuperación.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es la búsqueda de texto completo?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

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

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/full-text-search/