Búsqueda de texto completo en MongoDB

Federico Nocetti, Martín Feldman
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay

Resumen

La búsqueda de texto completo es esencial en la gestión de bases de datos documentales para realizar búsquedas eficientes en grandes volúmenes de texto. Este proyecto se enfoca en desarrollar funcionalidades de búsqueda de texto completo en MongoDB, un motor de bases de datos documentales. A diferencia de las capacidades de búsqueda de texto completo de MongoDB Atlas, que solo están disponibles en ese servicio, este proyecto busca construir un modelo de datos y herramientas que permitan la indexación y búsqueda de texto utilizando un lenguaje de consultas simple sin depender de MongoDB Atlas.La solución propuesta incluye características como búsquedas basadas en características del idioma, ignorar mayúsculas y minúsculas, ordenamiento por relevancia y búsqueda por frases mediante la implementación de un índice invertido completo utilizando una *Aggregation Pipeline* en MongoDB.

I. Introducción

La búsqueda de texto completo (*Full Text Search*) es una funcionalidad fundamental en la gestión de bases de datos documentales, ya que permite realizar búsquedas eficientes y precisas en grandes volúmenes de texto. A diferencia de las búsquedas basadas en coincidencias exactas de palabras clave, la búsqueda de texto completo permite encontrar resultados relevantes incluso cuando las palabras clave no coinciden exactamente.

El objetivo principal de este proyecto es desarrollar funcionalidades de búsqueda de texto completo en un motor de bases de datos documentales. En particular, nos enfocaremos en el motor MongoDB, que ha sido estudiado durante el curso. Si bien MongoDB ofrece capacidades avanzadas de búsqueda de texto completo, estas funcionalidades solo están disponibles para su uso en MongoDB Atlas. La limitación de acceder únicamente a estas capacidades a través de MongoDB Atlas plantea una restricción para aquellos que deseen utilizarlas en instancias fuera de este servicio. Por lo tanto, en este trabajo se construirá un modelo de datos y una serie de herramientas que permitan la indexación y búsqueda de texto utilizando un lenguaje de consultas rudimentario sin necesidad de utilizar el mencionado servicio.

A continuación, se presenta la estructura del documento. La Sección II aborda los trabajos relacionados con el tema que se estudia en este documento. En la Sección III se describe en detalle el trabajo realizado, incluyendo las decisiones tomadas y el análisis, diseño e implementación de la solución propuesta. La Sección IV presenta el cuerpo de texto utilizado, junto con las pruebas realizadas sobre él y su correspondiente análisis y comparación con las funcionalidades equivalentes proporcionadas por MongoDB Atlas. Por último, en la Sección VI se evalúan los resultados obtenidos en este trabajo y se plantean posibles extensiones para futuras investigaciones

II. Trabajos relacionados

Como fue mencionado anteriormente, las funcionalidades implementadas en este trabajo ya son provistas por MongoDB Atlas, y desde allí fue que obtuvimos algunas de las ideas a llevar a cabo en este trabajo. AtlasSearch está implementado utilizando Apache Lucene [1] para el manejo de los índices. Lo mismo sucede con Apache Solr [2], ElasticSearch [3] y OpenSearch [4], por lo que todos ellos implementan conjuntos similares de funcionalidades. Particularmente, para AtlasSearch destacan las siguientes funcionalidades en contraposición a otras alternativas dentro del ecosistema de MongoDB [5]:

- Características específicas del idioma. Implica la utilización de tokenizers y normalizadores que respeten las características del idioma, como ser los acentos diacríticos.
- Búsquedas que no distinguen entre mayúsculas y minúsculas
- Resaltado de fragmentos de texto que brinden contexto a los resultados
- Búsquedas especializadas en autocompletado de textos, también conocidas como “buscar mientras escribes” o *search as you type*. Estas búsquedas utilizan un tipo particular de tokenización donde el resultado no son palabras enteras sino fragmentos de palabras de un tamaño dado. Por ejemplo, para “búsqueda”, los 3-gramas son “bus”, “usq”, “squ”, “que”, “ued” y “eda”.
- Búsquedas basadas en más de diez términos. Esta característica es de importancia debido a limitaciones en los índices de texto de MongoDB.
- Ordenamiento por relevancia. El objetivo es ordenar los resultados considerando que tan bien satisface el criterio de búsqueda. AtlasSearch utiliza un algoritmo conocido como BM25, donde lo más importante a saber es que la relevancia

de un documento crece con la cantidad de ocurrencias de los términos de búsqueda, y es ponderada según el tamaño del cuerpo de texto.
- Búsquedas por frases. Que consiste en buscar coincidencia de frases enteras en el texto en lugar de términos individuales.

Además cabe notar que AtlasSearch plantea algunas limitaciones:
- Imposibilidad de mantener instalaciones locales, solamente siendo posible utilizar las funcionalidades en la plataforma Atlas.
- No es posible indexar documentos como parte de una transacción ACID.

III. Solución propuesta

Considerando las funcionalidades implementadas por AtlasSearch, se definió que este trabajo abarque las siguientes funcionalidades:
- Búsquedas con características específicas del idioma. Particularmente nos centramos en la normalización de los acentos diacríticos propios del Español
- Búsquedas que no distinguen mayúsculas y minúsculas
- Búsquedas basadas en más de 10 términos
- Ordenamiento por relevancia
- Búsqueda por frases
- Posibilidad de mantener instalaciones locales
- Indexación de documentos como parte de una transacción ACID

Para poder llevar a cabo estas búsquedas de texto completo de manera eficiente se decidió crear un índice invertido completo [6]. En el contexto de las bases de datos, un índice invertido es una estructura de datos donde se guarda una asignación de claves a valores en la base. En este caso, las claves están constituidas por términos contenidos en los documentos y los valores son los propios documentos que los contienen. Se le llama índice invertido completo cuando además de guardar el documento donde el texto aparece, se guarda la posición donde lo hace. El propósito de utilizar esta estructura de datos es optimizar las búsquedas de texto completo. En el contexto de la base documental, los términos que son clave del mapeo se almacenan en un campo indizado por la base de datos. Esto significa que localizar documentos que tienen al menos una ocurrencia de un término es realizado eficientemente mediante un escaneo de índice. Para lograr los objetivos de considerar características del idioma y no distinguir mayúsculas de minúsculas, se utilizaron algoritmos de normalización y tokenización acordes. El proceso de indizar documentos necesita partir del texto y procesarlo hasta obtener un arreglo de todos los términos que lo componen. El proceso de dividir el texto en sus términos se conoce como tokenización y usualmente es responsable de parte del soporte de las características del idioma. Por ejemplo, y como es el caso de este trabajo, saber que las palabras del español están separadas por espacios en blanco y, en algunos casos, por guiones, barras, paréntesis o signos de interrogación y exclamación. A su vez, para introducir tolerancia a la diferencia de mayusculas y minusculas, falta de acentos diacríticos, o incluso la falta de la virgulilla de la letra ñ, es necesario previamente normalizar el texto, de modo que todas estas características conflictivas sean reducidas a una representación en la que todas las variantes sean equivalentes. A modo de ejemplo: “nino” sea equivalente a “niño” y “Martín” sea equivalente a “martin”. De esta forma el texto “Martín salió de viaje.” es primero normalizado a “martin salio de viaje” y luego descompuesto en los términos “martin”, “salio”, “de” y “viaje”. Para que el proceso de normalización y tokenización sea efectivo, debe aplicarse también a los términos de búsqueda. De modo que si se busca por “Martin”, “martín” o “Salió”, todos ellos incluyen el texto que analizamos anteriormente entre sus resultados.

Para la construcción del índice inverso en MongoDB se ha implementado una *Aggregation Pipeline* que cumple con los siguientes pasos:
- Es ejecutada sobre la colección de documentos
- Ejecuta el normalizador para normalizar el campo de interés, lo que implica convertirlo en minúsculas y eliminar diacríticos y signos de puntuación.
- Se aplica el tokenizador, que divide el campo normalizado en términos según los espacios en blanco y cuenta las ocurrencias de cada término.
- Se agrupan todos documentos donde cada grupo es un término, y los componentes son los documentos en los que ocurre.
- Finalmente se escriben los resultados en una colección “indice”, que contiene el índice inverso.

En la colección “índice”, cada documento corresponde a un término identificado en el campo analizado, y contiene información sobre los documentos en los que aparece dicho término. Para cada término, se guarda una lista de documentos en el campo ”documentos”, donde cada elemento de la lista incluye el identificador del documento (siguiendo el patrón de documentos referenciados), las ocurrencias del término en ese documento y el recuento total de ocurrencias del término en el conjunto de documentos.

Por otro lado, se construyeron dos *pipelines* que implementan la funcionalidad de búsqueda, una que permite la búsqueda de palabras y otra de frases.

Ambas búsquedas comienzan de la misma forma:

- Las *pipelines* son ejecutadas como agregaciones del sistema, es decir, sobre la base de datos en sí, y no sobre una colección. Esto da la posibilidad de utilizar otro conjunto de etapas, entre las que se encuentra $documents, que permite especificar los documentos que va a consumir la *pipeline*.
- Utilizando la etapa $documents se especifica un documento de entrada que contiene un campo "criterio" que contiene los términos de búsqueda.
- Al igual que en la indización, este documento de entrada es pasado por los procesos de normalización y tokenización.
- Se realiza una búsqueda sobre la colección "indice" buscando documentos cuya clave (recordar que se corresponde a un término en el documento) coincida con alguno de los términos en el criterio.
- Se reagrupan los resultados de forma que cada documento sea incluido una vez, pero conteniendo todos los datos necesarios para el cálculo de los puntajes. Qué información es necesaria varía levemente entre los tipos de búsqueda.

A partir de aquí, las búsquedas simples se limitan a calcular el puntaje de los documentos y ordenarlos acordemente. Los puntajes son calculados como una versión simplificada del algoritmo BM25, que conserva la idea principal: se suman las ocurrencias de cada término y se divide por el largo del texto del documento a puntuar. Para las búsquedas por frases enteras la *pipeline* aplica un algoritmo que calcula la intersección de las ocurrencias de los términos de modo que sólo se conservan aquellos documentos en los que ocurren todos los términos solicitados, y además en el orden correcto. Esta solución no es óptima, y ha sido objeto de estudio [7]. Es de conocimiento que no existen algoritmos eficientes para realizar la intersección pero, a los efectos de mantener la simplicidad, este trabajo utiliza esta estrategia. Luego de calcular la intersección se calculan los puntajes utilizando el mismo criterio que para las búsquedas simples.

Finalmente, las características de permitir instalaciones locales, búsquedas por más de 10 términos e indizar como parte de una transacción quedan cubiertas por el hecho de no utilizar ninguna funcionalidad que no esté disponible en instalaciones locales de MongoDB, la estructura de datos elegida, y la posibilidad de utilizar la *Aggregation Pipeline* dentro de transacciones. Particularmente en el caso de las transacciones, no es posible utilizarlas cuando se usa la etapa $out, que es la encargada de escribir la colección "indice". Esto lo puede solucionar el usuario realizando la escritura en capa de aplicación.

El código fuente de la implementación está disponible en el repositorio [10]. Particularmente el archivo Proyecto_final.ipynb es un notebook de Jupyter con el que se pueden ejecutar los algoritmos.

IV. Experimentación

El conjunto de datos utilizados para realizar las pruebas fue un conjunto de reseñas en español realizadas en el sitio Amazon.com, el cual fue obtenido del sitio Huggin Face [8]. A partir de este conjunto de datos cargamos la base de nuestro sistema tomando solamente los títulos y cuerpos de dichas reseñas, obteniendo así 5000 documentos para realizar las pruebas.

Se decidió comparar los resultados y tiempos de ejecución de las consultas realizadas mediante el modelo de datos desarrollado y los tiempos de ejecución de las consultas equivalentes en MongoDB Atlas. En un primer intento se intentó replicar la base en una instancia de MongoDB Atlas para de esta forma poder realizar una comparación lo más justa posible, pero esto no fue posible ya que nuestro sistema hace uno del operador $function, el cual no está disponible en el tier gratuito de MongoDB Atlas. Por lo que se decidió usar la base local para nuestras consultas y cargar con los mismos datos una base en Atlas para realizar las operaciones de $search allí. Para esto también fue necesario crear un índice inverso a partir de la interfaz de usuario provista por Atlas. El mismo fue personalizado para que sea lo más parecido posible al utilizado en nuestro sistema. A continuación, se detallan las comparaciones de algunas consultas en ambos sistemas.

Criterio	**Cantidad de resultados de la solución**	**Tiempo de ejecución de la solución**	**Cantidad de resultados de Atlas**	**Tiempo de ejecución de Atlas**
kindle	3	4	3	3
precio tenia	665	3,6	661	2
precio tenia envio amazon	879	3,8	875	2
si y no	3851	3,5	3849	2

Búsqueda simple - *search* con '*text*'.

Criterio	Cantidad de resultados de la solución	Tiempo de ejecución de la solución	Cantidad de resultados de Atlas	Tiempo de ejecución de Atlas
el producto	267	5	265	3
el producto es muy	3	4	3	3
por el precio que tiene	40	4	40	2
por el precio que tiene no se puede pedir	5	4	5	3

Búsqueda de frase - $*search* con '*phrase*'.

A partir de esta comparación podemos decir que los tiempos de ejecución de nuestro sistema son muy similares a los que se pueden obtener utilizando las características de full text search provistas por Atlas. También notamos que algunos resultados válidos no eran obtenidos con el operador $search de Atlas, por ejemplo cuando el documento contenía "precio.caro" esta coincidencia no fue considerada válida. Desconocemos la razón de esto ya que depende de la implementación que realiza Atlas, pero consideramos que en estos casos nuestra implementación provee al usuario la posibilidad de encontrar más datos relevantes a su consulta.

V. Conclusiones y trabajo futuro

Concluimos que resulta factible la implementación de funcionalidades de búsqueda de texto completo utilizando las construcciones básicas provistas por instalaciones locales de MongoDB. La utilización del operador $function resultó de vital importancia permitiendo la construcción de algoritmos que, si bien por su complejidad no fueron probados con etapas de *pipeline*, resultan más eficientes implementados con código imperativo en comparación a utilizar las etapas de *pipeline* que tienen un enfoque más funcional. Además, las pruebas realizadas muestran que si bien la implementación de Atlas es más eficiente (lo que se explica no solo en el manejo de bajo nivel de los recursos sino también en el nivel de estudio y optimización) la pérdida de performance al utilizar las *pipelines* de este trabajo no es sustantiva y puede ser perfectamente justificada por algunos requerimientos de aplicación, como ser la necesidad de indexar dentro de transacciones.

Además, el tamaño relativamente reducido del juego de datos que utilizamos para realizar las pruebas puede estar escondiendo problemas de escalabilidad: si bien las pruebas muestran tiempos similares, puede que escalado a millones de documentos la diferencia pueda hacerse notable.

Como trabajos futuros para extender lo realizado en este se destacan:

- Implementar highlighting. Esta opción permite ver el contexto original en el cual el término buscado fue encontrado. Para esto deberíamos llevar la posición de la ocurrencia del término y el resultado de la operación debería devolver las tres palabras anteriores y las tres palabras siguientes
- Implementar stemming, este proceso se refiere a la reducción de palabras a su raíz léxica, como reducir las palabras "conectado", "conectó" y "conectaba" a "conect". De esta forma, si un documento contiene una de las variaciones de su raíz léxica se considerará como una coincidencia válida durante la consulta. Aplicando esta técnica se aumentan las probabilidades de recuperar todos los documentos relevantes a las búsquedas.
- Sustituir las utilizaciones de $function por etapas que implementen la misma funcionalidad, de modo que la implementación resulte utilizable en Atlas, y se pueda estudiar el desempeño en su ejecución.
- Implementar algoritmos de normalización y tokenización que permitan la búsqueda de sugerencias y autocompletado de palabras.
- Realizar pruebas de desempeño utilizando un juego de datos de tamaño más considerable.

Referencias

[1] Atlas Search Architecture (s. f.). https://www.mongodb.com/docs/atlas/atlas-search/atlas-search-overview/fts-architecture
[2] Apache Solr (s.f.). https://solr.apache.org/
[3] ¿Qué es Elasticsearch? (s. f.). https://www.elastic.co/es/what-is/elasticsearch
[4] OpenSearch (s.f.). https://opensearch.org/
[5] Marcus Eagan (2022). A Decisioning Framework for MongoDB $regex and$text vs Atlas Search. https://www.mongodb.com/developer/products/atlas/atlas-search-vs-regex/
[6] Paul E. Black (2006). "full inverted index", en Dictionary of Algorithms and Data Structures. https://www.nist.gov/dads/HTML/fullInvertedIndex.html
[7] Manish Patil, Sharma V. Thankachan, Rahul Shah, Wing-Kai Hon, Jeffrey Scott Vitter, Sabrina Chandrasekaran (2011) Inverted Indexes for Phrases and Strings. http://www.cs.nthu.edu.tw/ wkhon/papers/PTSHVC11.pdf
[8] Hugging Face (s.f.). amazon_reviews_multi https://huggingface.co/datasets/amazon_reviews_multi/viewer/es/test
[9] Highlight Search Terms in Results (s.f.). https://www.mongodb.com/docs/atlas/atlas-search/highlighting/
[10] Enlace a repositorio del trabajo . https://gitlab.fing.edu.uy/martin.feldman/bdnr2023-G1