IIC2413 — Bases de Datos

Búsqueda en textos

Para esta actividad es mejor que trabajes con el pedazo grande de Wikidata, wiki_500000.josn. Impórtalo igual que antes:

```
mongoimport --db test --collection entidades --drop --file wiki_500000.json --jsonArray
```

1. Búsqueda básica

Imagina que quieres todas las entidades de Wikidata en donde aparece "chile" en la descripción. La forma más obvia (y la única que sabríamos hacer hasta el momento) sería usar el equivalente MongoDB para el operador `LIKE` en SQL. La consulta que obtiene las descripciones en inglés que contienen "chile"se ve así (el operador `$regex` es el equivalente al `LIKE`):

```
db.entidades.find({"descriptions.en.value":{$regex:"chile"}},
                                        {"descriptions.en.value":1});
```

1. Corre esta consulta y espera los resultados. Ahora tienes unos minutos sin computador para hacer lo que quieras.
2. Discute con tu compañera o compañero de al lado: ¿por qué se demora tanto esta consulta? ¿Cómo se podría mejorar el rendimiento?

Text Search

Como lo han discutido, no hay una forma evidente de indexar el campo `descriptions.en.value` para que retorne las descripciones que contienen ciertas palabras clave. Antes de ver cómo funciona, veamos cómo hacerlo en mongo.

1. Crea un índice de texto en ese campo (el valor `"text"` indica que el índice es de texto):

   ```
   db.entidades.createIndex({"descriptions.en.value":"text"})
   ```

2. Ahora puedes hacer consultas! La siguiente consulta busca todas las descripciones que tengan chile (esta si que funciona rápido)

   ```
   db.entidades.find({$text: {$search: "chile"}},{"descriptions.en.value":1});
   ```

3. Existen varias otras cosas que hacer, como hacer un "or"(en este caso entre chile y town):

```
db.entidades.find({$text: {$search: "chile town"}},{"descriptions.en.value":1});
```

4. Buscar frases enteras (en este caso la frase `in chile`) poniéndolas entre comillas (la comilla debe ir acompañada del comando de escape, así: `\"`):

```
db.entidades.find({$text: {$search: "\"in chile\""}},{"descriptions.en.value":1});
```

5. Combinar más de una frase para hacer un "and"(en este caso todo lo que contenga chile y town):

```
db.entidades.find({$text: {$search: "\"chile\" \"town\""}},
                                                    {"descriptions.en.value":1});
```

6. Y hacer "not"(en este caso todo lo que contiene chile y no contenga war):

```
db.entidades.find({$text: {$search: "chile -war"}},{"descriptions.en.value":1});
```

1.1. ¿Cómo funciona?

Limitémonos a pensar solo en las descripciones de los documentos de Wikidata. Llamemos D al conjunto de descripciones. Asume también que tenemos un índice en el campo `_id` de los documentos, por lo que, dado un `_id`, podemos encontrar rápidamente la descripción que corresponde a ese `_id`.

1. Imagina que P es el conjunto de todas las palabras que aparecen en D. ¿Cómo podríamos crear una estructura que nos permita acceder rápidamente a todos los `_id` de descripciones que tienen una palabra en P?

2. Acá hay una idea: Hacer una matriz de $|P|$ por $|D|$ entradas: las columnas son los documentos y las filas son las palabras, y hay un 1 en la entrada $[i, j]$ si la palabra i aparece en el documento j.

3. Discute las ventajas y desventajas de la idea. ¿Qué tamaño tiene esta matriz? ¿Qué tan rápido es conseguir todos los documentos que mencionan Chile?

4. Piensa en cómo hacer algo más eficiente que permita recuperar información igual de rápido.

Consultas booleanas
Imagina que tienes un índice invertido sobre las descripciones de artículos en wikidata.

5. Explica como buscar todos los documentos que mencionan la palabra p y la palabra q

6. Explica como buscar todos los documentos que mencionan a la palabra p y no a la palabra q
7. Explica como buscar todos los documentos que mencionan a la palabra p pero no a la palabra q
8. Explica como buscar todos los documentos que mencionan a la frase "$p\ q\ r$"
9. Explica por que es problemático buscar todos los documentos que no contengan la palabra p

1.2. Otras ventajas: Stemming, Stop Words

Ejecuta esta consulta. Le estás pidiendo a MongoDB que te explique cómo fue procesada.

```
db.entidades.find({$text: {$search: "\"chile\" Town -war"}},
                                        {"descriptions.en.value":1}).explain(true);
```

Mira `parsedTextQuery`, y el objeto `terms`. Verás que está buscando, entre otras cosas, por la palabra `town` (sin mayúscula). Esto es común en los buscadores de texto, y se conoce como *Stemming*: tratamos de reducir las palabras complicadas a sus raíces, lo que incluye eliminar signos de puntuación, pasar a minúsculas, y transformar cosas como "chiles" a "chile".

Ejecuta ahora esta consulta, y fíjate qué pasa con la palabra "in":

```
db.entidades.find({$text: {$search: "towns in chile"}},
                                  {"descriptions.en.value":1}).explain(true);
```

Las palabras que no se incluyen (por ser muy comunes) se llaman Stop Words.

1. Encuentra otro ejemplo de stemming en inglés, y otra Stop Word que no sea "in".
2. Imagina ahora ejemplos de stemming en español. ¿Hay alguno que cambie entre español e inglés? ¿qué stop words hay en español?
3. Mira la diferencia entre ``towns` (cómo término) y `"\"towns\""` (como frase).

1.3. Índices Compuestos

Quizá queremos buscar tanto en el título como en el contenido del documento. A esta idea se le llama un índice compuesto. Para indexar las descripciones y los labels en wikidata (inglés), ingresa (ojo que mongo solo admite un índice invertido por collection, así que tienes que borrar el anterior):

```
db.entidades.createIndex(
             {"descriptions.en.value":"text","labels.en.value":"text"})
```

Listo!

1. Busca todos los documentos que contienen chile, ahora usando el índice compuesto. Busca otras consultas donde quede claro que se usa también el label.

Bonus: pesos a distintos campos

Averigua como hacer que el label sea más importante que la descripción a la hora de buscar en índices compuestos. **Ayuda:** Lo que puedes hacer es asignar pesos (weights) distintos a cada campo, y ordenar por su puntaje (mira más abajo).

2. Búsqueda con rankeo

Imagina que quieres información sobre todos los mamíferos de chile. Ejecutas la siguiente consulta:

```
db.entidades.find({$text: {$search: "chilean mammal"}},
                        {"descriptions.en.value":1);
```

Tienes una mezcla de mamíferos y de cosas relacionadas en chile. Cómo decide MongoDB qué resultados entregar? Internamente, MongoDB asigna un puntaje de relevancia a las distintas descripciones que hacen match. Hay una manera más explícita de ordenar los resultados por relevancia. Ejecuta primero la consulta que muestra el puntaje.

```
db.entidades.find({$text: {$search: "chilean mammal"}},
                        {"descriptions.en.value":1, score: {$meta: "textScore"}});
```

Como ves, el resultado tiene resultados de descripciones que contienen "chilean.º "mammal", y a cada uno se le asigna un puntaje de acuerdo a la relevancia. Para ordenar los resultados por su relevancia, ejecuta en vez lo siguiente:

```
db.entidades.find({$text: {$search: "chilean mammal"}},{"descriptions.en.value":1,
           score: { $meta: "textScore" }}).sort( { score: { $meta: "textScore" }});
```

1. (en grupos) Averigua de qué depende el puntaje de MongoDB. Para esto, crea una nueva collection con unos 5-10 documentos (que sean puro texto), y anda modificando a medida que vas investigando. **Ayuda:** generalmente el puntaje es proporcional a la cantidad de veces que aparece en el documento, e inversamente proporcional a la cantidad de documentos en los que aparece. ¿Es esto cierto en MongoDB?