Requêtes textuelles et géo-spatiales

Corrigé de TD pour un cours dispensé à l'université de Rennes 2

Romain Tavenard

1 Chargement de la base `discours`

0. Chargez la base `elections2007` disponible sous forme de fichier JSON sur CURSUS et nommez la collection contenant ces données `discours` (*cf.* l'aide de la commande `mongoimport` si besoin). Ajoutez, sur l'attribut `content` de la collection `discours`, un index textuel en langue française (*cf.* doc de MongoDB).

a. Dans votre terminal "annexe" (si votre serveur tourne sur le port `1234`) :

```
mongoimport --host localhost:1234 --jsonArray --db elections2007 --collection discours --fil
```

b. Dans votre terminal "client" :

```
> use elections2007
> db.discours.createIndex({"content": "text"})
```

Si vous n'avez plus la base `food` à votre disposition sur votre serveur MongoDB, il faudra la recharger également, et mettre en place l'index géo-spatial comme vu au TD précédent.

2 Requêtes textuelles

Pour cette partie, vous travaillerez sur la base `elections2007`.

2.1 Expressions régulières

Lorsque l'on souhaite interroger des champs textuels, il existe, en plus des outils précédemment introduits, deux méthodes spécifiques :

1. l'utilisation d'expressions régulières ;
2. le recours à un index textuel qui permet d'effectuer des recherches de type "moteur de recherche".

Les expressions régulières sont des moyens de mettre des contraintes sur les chaînes de caractères acceptables. Parmi ces contraintes, on peut citer :

— `/blabla/` : on n'accepte que les chaînes de caractères contenant la sous-chaîne `"blabla"`;
— `/blabla/i` : on n'accepte que les chaînes de caractères contenant la sous-chaîne `"blabla"` en ne s'intéressant pas à la casse (c'est le sens du caractère `i` placé après le deuxième slash) : la chaîne `"xyuBLAblahui"` sera par exemple jugée acceptable;
— `/^blabla/` : on n'accepte que les chaînes de caractères commençant par la chaîne `"blabla"` (c'est le sens du caractère `^`) ;
— `/blabla$/` : on n'accepte que les chaînes de caractères se terminant par la chaîne `"blabla"` (c'est le sens du caractère `$`) ;
— `/bla.bla/` : on n'accepte que les chaînes de caractères contenant une sous-chaîne de la forme `"bla"` suivi d'un caractère quelconque (c'est le sens du caractère `.`) suivi de `"bla"` ;
— `/bla.*bla/` : on n'accepte que les chaînes de caractères contenant une sous-chaîne de la forme `"bla"` suivi d'une série de caractères quelconques (c'est le sens des caractères `.*`) suivi de `"bla"`. On pourra consulter la page wikipedia consacrée aux expressions régulières [lien] pour avoir un aperçu des possibilités supplémentaires de ces expressions régulières.

En langage MongoDB, une requête utilisant une expression régulière sera de la forme :

```
> db.nomDeLaCollection.find({"cle": /blabla/})
```

1. Affichez la liste des discours pour lesquels l'orateur a un prénom qui commence par la lettre J.

```
> db.discours.find({"name": /^J/i})
```

2. Affichez la liste des discours pour lesquels l'orateur a un nom qui se termine par la chaîne `"et"`.

```
> db.discours.find({"name": /et$/i })
```

3. Affichez la liste des discours pour lesquels l'orateur a un nom qui commence par la lettre S.

```
> db.discours.find({"name": / S/ })
```

4. Affichez la liste des discours pour lesquels l'orateur a un prénom composé.

```
> db.discours.find({"name": /-.* / })
```

2.2 Requêtes de type "moteur de recherche"

Dans certains cas, on souhaitera chercher dans des champs textuels comme on le ferait à l'aide d'un moteur de recherche. Pour cela, il faut qu'un index spécifique soit mis en place sur le champ en question. Nous verrons plus tard dans ce cours la notion d'index, mais vous pouvez dès maintenant vérifier qu'un tel index existe :

```
> db.nomDeLaCollection.getIndexes()
```

Lorsqu'un tel index existe, MongoDB saura que c'est sur le champ correspondant qu'il faudra effectuer les requêtes textuelles et il ne sera donc plus utile de le préciser. On obtiendra alors une requête de la forme :

```
> db.nomDeLaCollection.find({$text : {$search : "ma requête"}})
```

Par défaut, lorsque l'on effectue une requête contenant plusieurs termes, un `OU` logique est effectué : les documents retournés sont ceux qui contiennent au moins l'un des termes. On peut également effectuer une requête impliquant une expression exacte, qui sera encadrée de guillemets échappés par un caractère `"\"` :

```
> db.nomDeLaCollection.find({$text : {$search : "\"ma requête\""}})
```

Enfin, on peut exclure les documents contenant certains termes en précédant chacun de ces termes par le symbole `"-"`. Toutefois, si l'on mélange plusieurs types de conditions, un `ET` logique est effectué entre les différents types, comme par exemple :

- chaîne exacte et termes isolés ;
- deux chaînes exactes distinctes ;
- inclusion de certains termes et exclusion d'autre termes.

Par exemple, la requête suivante retournera tous les documents contenant le terme `"bla"` `ET` ne contenant pas le terme `"bli"` :

```
> db.nomDeLaCollection.find({$text : {$search : "bla -bli"}})
```

Enfin, il est possible d'afficher les scores (type tf-idf) obtenus par les différents documents à l'aide de la commande :

```
> db.nomDeLaCollection.find({$text: {$search: "ma requête"}},
                            {"score": {$meta: "textScore"}})
```

Dans ce cas, il sera souvent utile de trier les résultats par ordre de pertinence décroissante :

```
> db.nomDeLaCollection.find({$text: {$search: "ma requête"}},
                            {"score": {$meta: "textScore"}}).sort(
                                {score: {$meta: "textScore"}})
```

5. Affichez la liste des documents contenant l'expression exacte suivante : `"l'effet sera nul"`.

```
> db.discours.find({$text: {$search: "\"l'effet sera nul\""}})
```

6. Affichez la liste des discours dans lesquels ont été prononcés :

- le terme écologie
- l'un des termes écologie ou politique
- les deux termes écologie et politique

```
> db.discours.find({$text: {$search: "écologie"}})
> db.discours.find({$text: {$search: "écologie politique"}})
> db.discours.find({$text: {$search: "\"écologie\" \"politique\""}})
```

7. Vérifiez que les scores retournés dans la deuxième requête de la question précédente tendent à favoriser les documents contenant les deux termes (les autres documents ont des scores plus faibles).

```
> db.discours.find({$text: {$search: "écologie politique"}},
                   {"name": true, "score": {$meta: "textScore"}}).sort(
                       {score: {$meta: "textScore"}})
```

8. Affichez la liste des documents comportant le terme "famille" mais pas le terme "politique".

```
> db.discours.find({$text: {$search: "famille -politique"}})
```

3 Requêtes géo-spatiales

Dans le cas de données géo-spatiales, on peut avoir à effectuer des requêtes spécifiques, comme par exemple :

— trouver les éléments de la base les plus proches d'un point donné ;
— trouver les éléments de la base entièrement inclus dans un polygone donné ;
— trouver les éléments de la base dont au moins une partie est incluse dans un polygone donné (ce qui peut être utile si la base représente des rues par exemple et que l'on veut retourner toutes les rues appartenant au moins partiellement au polygone défini).

Pour cela, il faut tout d'abord s'assurer qu'un index géo-spatial (de type `2dsphere` pour des coordonnées à la surface de la terre) existe sur le champ visé.

9. Vérifiez qu'un index géo-spatial existe sur l'un des champs de la collection `NYfood` de la base `food`.

```
> use food
> db.NYfood.getIndexes()
```

MongoDB définit trois mots-clés correspondant aux situations définies précédemment :

— `$near` ;
— `$within`.

La syntaxe correspondante est toujours de la même forme (remplacer `$near` par `$within` au besoin) :

```
> db.nomDeLaCollection.find({"clé": {$near : {$geometry : ref}}})
```

où `ref` est une variable correspondant à la référence à laquelle on souhaite se comparer :

— un point dans le cas du mot-clé `$near` :

```
> var ref = {"type": "Point", "coordinates": [longitude, latitude]}
```

— un polygone dans le cas du mots-clé `$within` :

```
> var ref = {"type": "Polygon", "coordinates": [[[long1, lat1],
                                                 [long2, lat2],
                                                 [long3, lat3],
                                                 [long4, lat4],
                                                 [long1, lat1]]]}
```

Ainsi, il faudra veiller à ce que les polygones que vous créez sont bien fermés (les dernières coordonnées sont égales aux premières).

10. Trouver les restaurants les plus proches de Crown Heights (supposé ponctuel de coordonnées approximatives `[-73.923, 40.676]`).

```
> var CrownHeights = {"type": "Point", "coordinates": [-73.923, 40.676]}
> db.NYfood.find({"address.loc": {$near : {$geometry : CrownHeights}}})
```

11. Trouver les restaurants chinois les plus proches de Crown Heights.

```
> db.NYfood.find({"address.loc": {$near: {$geometry: CrownHeights}},
                  "cuisine": "Chinese"})
```

12. Même question en ne conservant que les restaurants situés à moins de 500m de Crown Heights (voir la doc MongoDB si besoin).

```
> db.NYfood.find({"address.loc": {$near: {$geometry: CrownHeights,
                                          $maxDistance: 500}},
                  "cuisine": "Chinese"})
```

13. Trouver tous les restaurants du quartier d'East Village à New York (vous pourrez utiliser Google Maps pour connaître les coordonnées du polygone entourant ce quartier).

```
> var eastVillage = {"type": "Polygon",
                     "coordinates": [[[-73.9917900, 40.7264100],
                                      [-73.9917900, 40.7321400],
                                      [-73.9829300, 40.7321400],
                                      [-73.9829300, 40.7264100],
                                      [-73.9917900, 40.7264100]]]}
> db.NYfood.find({"address.loc": {$within: {$geometry: eastVillage}}})
```