

TP n°3

Requêtes MongoDB

Le langage de requête MongoDB est un langage de recherche dit “par motif” (pattern). Il consiste à interroger une collection en donnant un objet “motif/pattern” en JSON dont chaque attribut est interprété comme une contrainte sur la structure des objets à rechercher. L’interpréteur mongo permet de programmer en Javascript.

1) Sélections

On veut parcourir toute une collection. On utilise alors `find()` sans argument.

`db.movies.find ()`

Savoir combien de documents comprend le résultat?

`db.movies.count ()`

Les options `skip` et `limit` permettent de “paginer” le résultat. La requête suivante affiche 12 documents à partir du dixième inclus. MongoDB fournit les documents dans l’ordre où il les trouve dans les fichiers.

`db.movies.find ().skip(9).limit(12)`

On peut trier explicitement, ce qui rend le résultat plus déterministe.

`db.movies.find ().sort({"title": 1}).skip(9).limit(12)`

La spécification du tri repose sur un objet JSON, et ne prend en compte que les noms d’attribut sur lesquels s’effectue le tri. La valeur (ici, celle du titre) ne sert qu’à indiquer si on trie de manière ascendante (valeur 1) ou descendante (valeur -1).

Note : tout tri implique que le système constitue l’intégralité du résultat au préalable, ce qui induit une latence (temps de réponse) potentiellement élevée. Sans tri, le système peut délivrer les documents au fur et à mesure qu’il les trouve.

2) Critères de recherche

Si on connaît l’identifiant, on effectue la recherche ainsi.

`db.movies.find ({"_id": "movie:2"})`

Une requête sur l’identifiant ramène au plus un seul document. Dans un tel cas, on peut utiliser `findOne`.

`db.movies.findOne ({"_id": "movie:2"})`

Cette fonction renvoie toujours un document au plus, alors que la fonction `find` renvoie un curseur sur un ensemble de documents (même si c’est un singleton). La différence est surtout importante quand on utilise une API pour accéder à MongoDB avec un langage de programmation.

Sur le même modèle, on peut interroger n’importe quel attribut.

`db.movies.find ({"title": "Alien"})`

Ca marche bien pour des attributs atomiques (à une seule valeur), mais comment faire pour interroger des objets ou des tableaux imbriqués? On utilise dans ce cas des chemins, un peu à la

XPath, mais avec une syntaxe plus “orienté-objet”. Voici comment on recherche les films de Quentin Tarantino.

```
db.movies.find ({"director.last_name": "Tarantino"})
```

Et pour les acteurs, qui sont eux-mêmes dans un tableau? Ça fonctionne de la même manière.

```
db.movies.find ({"actors.last_name": "Tarantino"})
```

La requête s’interprète donc comme: “Tous les films dont l’un des acteurs se nomme Tarantino”.

Conformément aux principes du semi-structuré,

```
db.movies.find ({"actor.last_name": "Tarantino"})
```

Note : Contrairement à une base relationnelle, une base semi-structurée ne proteste pas quand on fait une faute de frappe sur des noms d’attributs, ou quand on fait référence à des attributs ou des chemins qui n’existent pas.

On peut introduire des expressions régulières. Tous les films dont le titre commence par Re? Voici:

```
db.movies.find ({"title": /^Re/}, {"actors": null, "summary": 0} )
```

Pas d’apostrophes autour de l’expression régulière. On peut aussi effectuer des recherches par intervalle.

```
db.movies.find( {"year": { $gte: "2000", $lte: "2005" } }, {"title": 1} )
```

3) Projections

On peut aussi faire des projections, en passant un second argument à la fonction find().

```
db.movies.find ({"actors.last_name": "Tarantino"}, {"title": true, "actors": 'j' } )
```

Le second argument est un objet JSON dont les attributs sont ceux à conserver dans le résultat. La valeur des attributs dans cet objet-projection ne prend que deux interprétations. Toute valeur autre que 0 ou null indique que l’attribut doit être conservé. Si on choisit au contraire d’indiquer les attributs à exclure, on leur donne la valeur 0 ou null. Par exemple, la requête suivante retourne les films sans les acteurs et sans le résumé.

```
db.movies.find ({"actors.last_name": "Tarantino"}, {"actors": null, "summary": 0})
```

4) Opérateurs ensemblistes

Les opérateurs du langage SQL **in**, **not in**, **any** et **all** se retrouvent dans le langage d’interrogation, mais MongoDB ne permet pas d’imbriquer des requêtes.

On cherche les films dans lesquels joue au moins un des artistes dans une liste (on suppose que l’on connaît l’identifiant).

```
db.movies.find({"actors._id": {$in: ["artist:34", "artist:98", "artist:1"]}})
```

Le **in** exprime le fait que l’une des valeurs du premier tableau (**actors._id**) doit être égale à l’une des valeurs de l’autre.

Pour exprimer le fait que toutes les valeurs de premier tableau se retrouvent dans le second (en d’autres termes, une inclusion), on utilise la clause **all**.

```
db.movies.find({"actors._id": {$all: ["artist:23","artist:147"]}})
```

Le **not in** correspond à l’opérateur **\$nin**.

```
db.artists.find({"_id": {$nin: ["artist:34","artist:98","artist:1"]}})
```

Comment trouver les films qui n'ont pas d'attribut summary?

```
db.movies.find({"summary": {$exists: false}}, {"title": 1})
```

5) Opérateurs Booléens

Quand on exprime plusieurs critères, c'est une conjonction (**and**) qui est appliquée. On peut l'indiquer explicitement. Voici la syntaxe (les films tournés avec Leonardo DiCaprio en 1997):

```
db.movies.find({$and : [{"year": "1997"}, {actors.last_name: "DiCaprio"}] })
```

Bien entendu il existe un opérateur **or** avec la même syntaxe. Les films parus en 1997 ou avec Leonardo DiCaprio.

```
db.movies.find({$or : [{"year": "1997"}, {actors.last_name: "DiCaprio"}] })
```

Que faire quand on doit croiser des informations présentes dans plusieurs collections?

En relationnel, on effectue des jointures. Avec Mongo, il faut bricoler.

6) Jointures

La jointure, au sens de: associer des objets distincts, provenant en général de plusieurs collections, pour appliquer des critères de recherche croisés, n'existe pas en MongoDB. On peut considérer que cette limitation est cohérente avec une approche documentaire dans laquelle les documents sont supposés indépendants les uns des autres.

Le serveur ne sachant pas effectuer de jointures, on en est réduit à les faire côté client. Cela revient à appliquer l'algorithme de jointures par boucle imbriquées en stockant des données temporaires dans des structures de données sur le client, et en effectuant des échanges réseaux entre le client et le serveur, ce qui dans l'ensemble est inefficace.

Exemple 1 : Considérons la requête “Donnez tous les films dont le directeur est Clint Eastwood”.

Note : on considère que dans la base moviesref un film ne contient que la référence au metteur en scène.

1. La première étape dans la jointure côté client consiste à chercher l'artiste Clint Eastwood et à le stocker dans une variable côté client.

```
eastwood = db.artists.findOne({"first_name": "Clint", "last_name": "Eastwood"})
```

2. Une seconde requête va récupérer les films dirigés par cet artiste.

```
db.movies.find({"director._id": eastwood['_id']}, {"title": 1})
```

Exemple 2 : Prenons la requête suivante:

```
select m.titre, a.* from Movie m, Artist a where m.id_director = a.id
```

Voici l'algorithme de jointure par boucles imbriquées, sous le shell de MongoDB :

```
var lesFilms = db.movies.find()
while (lesFilms.hasNext()) {
  var film = lesFilms.next();
  var mes = db.artists.findOne({"_id": film.director._id});
  printjson(film.title);
  printjson(mes);
}
```

}

Combien de fois sont exécutées les requêtes dans le programme javascript ci-dessus ?

Exercices

Requêtes sur la base des films.

1. tous les titres;
2. tous les titres des films parus après 2000;
3. le résumé de Spider-Man;
4. qui est le metteur en scène de Gladiator?
5. titre des films avec Kirsten Dunst;
6. quels films ont un résumé?
7. les films qui ne sont ni des drames ni des comédies.
8. affichez les titres des films et les noms des acteurs.
9. dans quels films Clint Eastwood est-il acteur mais pas réalisateur (aide: utilisez l'opérateur de comparaison \$ne).
10. Difficile: Comment chercher les films dont le metteur en scène est aussi un acteur? Pas sûr que ce soit possible sans recourir à une auto-jointure, côté client...