Ensimag

MongoDB
Base de données : Yelp_academic

Auteur :
Ghita benjelloun
Andrès Castellanos-paez

Encadrant :
Simon PAGEAUD

1er décembre 2018

Table des matières

0.1 Introduction

Le projet vise à comprendre et exploiter les fonctionnalités d'un système de base de données comme MongoDB.
Dans la suite du rapport, on commencera par présenter l'ensemble des données que nous avons récupérés pour le projet. Ensuite, on enchaînera avec les étapes qui nous ont permis d'atteindre notre objectif, lequel est de créer des requêtes complexes en s'imprégnant des situations de la vie réelle tout en dressant des comparaisons avant et après utilisation des indexes et d'autres caractéristiques de MongoDB pour augmenter la vitesse des requêtes.
Pour la création des requêtes, on a utilisé le site web de documentation de MongoDB comme référence du language [2].

0.2 Ensemble des données

Les données utilisées dans ce projet ont été importé de la plateforme web « kaggle » [1] organisant des compétitions en science des données. Sur cette plateforme, les entreprises proposent des problèmes en science des données et offrent un prix aux datalogistes obtenant les meilleures performances.
Les données sur lesquels on a choisi de travailler appartiennent à l'entreprise « Yelp » [3], une multinationale qui développe, héberge et commercialise Yelp.com et l'application mobile Yelp qui publient des avis participatifs sur les commerces locaux.
Ces données de taille totale égale à 5.3GO, sont distribuées sur 5 fichiers sous format JSON, une extension qui facilite l'importation dans la base de données MongoDB.
Après importation de ces données sur notre base de données "yelp_academic" sur mongoDB, on a fait subir à ces données plusieurs traitements disponible sur le script python "create_collections.py" graçe au framework pymongo, et visant essentiellement à corriger des types de données quand il le faut, tel que Date au lieu de String dans la collection tip. Aussi, on a du rajouté 2 collections lesquelles sont "ceo" et "bank_account" dont le but est de créer des types de relations qu'on avait pas dans les données bruts et donc considérer tous les types de relations et lesquelles sont : 1-N, N-N et 0-N.

Ci-dessous la description des relations et la vue generale dans la figure 1 entre nos collections finales.

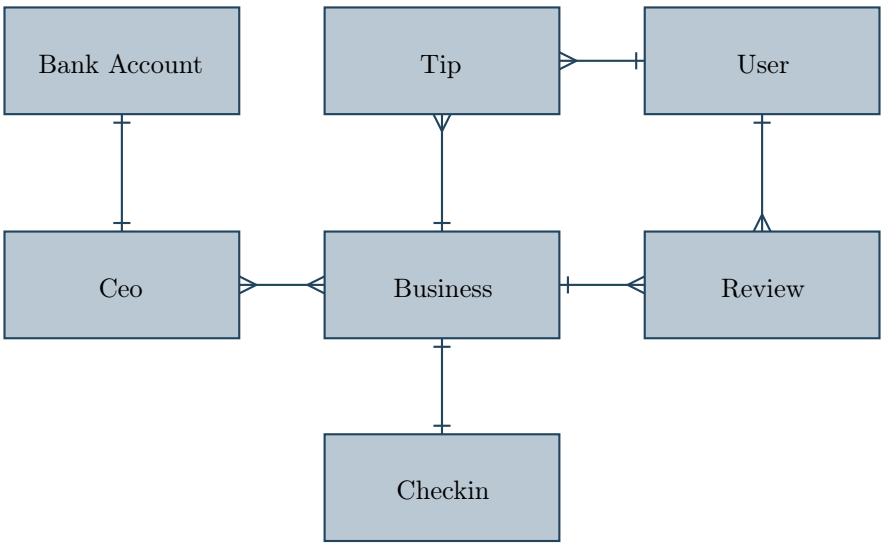

FIGURE 1 – Relationships diagram

Les collections crées et le type de données sont les suivants :

1. Business : Contient les informations relatives à chaque business, notamment le nom du business, son adresse, l'id de son chef, le nombre de stars qu'il détient, s'il est ouvert ou pas, des documents imbriqués dans le champs "attributes" pour décrire s'il a un parking, s'il offre le service de livraison, l'id de son chef qu'on a ajouté nous même après avoir inclu la collection "chefs", ainsi que d'autres éléments très utiles à connaître pour chaque business.

2. Checkin : contient le nombre de fois qu'un business a été visité par les utilisateurs de l'application par jour.

3. Review : C'est la collection la plus lourde de la base de donnée, et qui contient un examen de, business décrit par le champs "business_id", fait par, un user décrit par "user_id", et dressant son commentaire sous le champs "text", et un nombre de stars sous le champs "stars" avec la date et d'autres données utiles.

4. Tip : comporte des informations émises par un utilisateur donné ici décrit par le champs "user_id" sur un business décrit sous le champs "business_id", à savoir la date, le commentaire "text" et s'il a émis de like ou pas.

5. User : décrit les informations relatives à un utilisateur de l'application "yelp" de l'entreprise, à savoir son nom, la moyenne des stars qu'il émet sur les business, (ces stars sont visibles dans la collection "review"), la date de son inscription chez yelp "yelping_since" ...etc.

6. ceo : C'est une collection parmi celles qu'on a créé et ajouté à la base de données à l'aide du script python "generate_collections.py" pour considéré la relation 1-1 avec les "bank_account" et N-N avec les businesses. Elle contient les données relatives aux businesses où il est considéré parmi les chefs, et l'information relative à son compte bancaire.

7. bank_account : contient les données relatives au numéro bancaire ainsi qu'au déteneur du compte ceo_id qui devra avoir un et un seul compte bancaire.

0.3 Indexes : Avantages

Dans les bases de données, un index est une structure de données utilisée par le SGBD pour lui permettre de retrouver rapidement les données. L'utilisation d'un index simplifie alors et accélère les opérations de recherche, de tri, de jointure ou d'agrégation effectuées par le SGBD.

Pour cela, on a opté pour les indexes appliqués sur des champs qu'on a choisi avec soin et dont on explicitera le choix ci-après :

1. stars : Ce champs appartenant à la collection "Review", la plus grande collection de presque 4.5GO est important à indexer dans la mesure où on l'interroge dans la querie number4 et number3 et vu la grandeur de la collection, l'exécution des deux requêtes peut prendre beaucoup de temps sans indexes sur ce champs, qui dans le cas de la querie3nous facilitent les opérations de tri et de recherche effectués sur le champs "stars" et dans la querie 4 nous facile l'opération de selection.
2. stars : Ce champs appartenant à la collection Business, est utilisé deux fois dans la querie1 et la querie5, et donc vu son utilisation fréquente, il serait intéressant de créer des indexes sur ce champs-ci.
3. useful : De même pour "stars", ce champs appartenant à la collection "Review", la plus grande collection de presque 4.5GO est important à indexer dans la mesure où on l'interroge dans la querie number3 et donc encore une fois, vu la grandeur de la collection, l'exécution peut prendre beaucoup de temps sans indexes sur ce champs, qui dans ce cas nous facilitent les opérations de tri, de recherche sur critère (review.useful>4) et sélection sur critère effectués sur le champs "useful" dans la querie 3.

4. date : c'est un champs appartenant à la table tipe contenant beaucoup de document. Ce champs est interpellé deux fois, la première pour faire la conversion de son type de string à date, graçe au fichier date_conversion.js dans le dossier script, et l'autre dans la query3. Sans cet index, ces opérations prennent beaucoup de temps.
5. text : Ce champs appartenant à la collection "Review", celle-ci etait trop grande, on a jugé nécessaire d'ajouter des indexes pour accélérer notre traitement.
6. Business_id : (et non _id généré automatiquement par la base de données) Comme on peut regarder dans la figure 1 business a une relation avec 4 collections différentes et la relation est basée sur ce champ. Pour cela, on doit créer un index pour le champs business_id sur chaque collection qui contient tel champ afin de faciliter des opérations de jointures qu'on peut prévoir.
7. User_id : De même pour "stars" et "useful", ce champs appartenant à la collection "Review", la plus grande collection de presque 4.5GO est important à indexer dans la mesure où on l'interroge dans la querie number4 et qui dans ce cas nous facilitent les opérations de joiture avec la collection user.
8. User_id : Champs appartenant à la collection User, la deuxième plus grande collection de presque 2.1GO est important à indexer dans la mesure où on l'interroge dans la querie number4, et qui dans ce cas nous facilitent les opérations de joiture avec la collection Review par rapport au temps et à l'efficacité de la requête.

0.4 Queries

— Requête 1 :

Langage du système :

```
db.ceo.aggregate([
{$lookup:{from:"business", localField:"_id",foreignField:"ceo_list",as:"
    ↪ list_entreprises"}},
{$unwind:"$list_entreprises"},
{$group:{
"_id":"$_id",
avg:{$avg: "$list_entreprises.stars"}
}}]).forEach(function(doc){print(doc._id,doc.avg)})
```

Langage naturel :

Cette requête calcule la moyenne des étoiles "stars" des entreprises "business" de chaque ceo et affiche pour chaque ceo son id et la moyenne.

Indexes :

L'indexe utilisé ici est sur le champs "stars" car c'est un champs réutilisé dans une autre query et donc on a jugé intéressant de mettre des indexes sur les champs fréquemment requêtés.

— Requête 2 :

Langage du système :

```
db.tip.aggregate(
   [
     {
       $group:
         {
           _id: { dayOfYear_Number: { $dayOfYear: "$date"}, year: { $year: "
               ↪ $date" } },
```

```
          totalLikes: { $sum: "$like" },
          nbre_participants: { $sum: 1 }
        }
      }
    ]
).forEach( function(doc){ print(doc)})
```

Langage naturel :
Le but de cette requête est d'utiliser la date de chaque document et d'extraire le champs de l'année, et deviner le numéro du jour de l'année (numéro par rapport à 365j) à partir du champs "date" des documents, pour à la fin afficher le nombre des "likes" faits par groupement ($group) de numéro du jour puis groupement par année de la collection "tip".

Indexes :
Ici, on utilise les Indexes sur le champs "date", car on l'utilise aussi au préalable pour faire une conversion du type du champs "date" de string (à la récupération des données) au type Date (voir le dossier "script" puis le fichier "date_conversion"), et d'autre part car la collection "tip" contient beaucoup de documents et la conversion au type de date prend beaucoup de temps sans l'utilisation des indexes.

— Requête 3 :

Langage du système :

```
    db.review.aggregate(
  [
    {
      $match: {
        useful: {
          $gt: 4
        }
      }
    },
    { $sort: { stars: -1,useful:-1 }},
    {$limit:5}
  ]
).forEach(function(doc)
{print("\n"+"the business's id"+ doc.business_id+"\n"+"text is"+"\n"+ doc.
    ↪ text+"\n"+"stars_number: "+doc.stars+"\n"+"useful_degree: "+doc.
    ↪ useful)}
)
```

Langage naturel :

Cette requête permet d'extraire les documents de la collection "review" qui sont utiles "useful > 4" et on affiche le résultat par ordre décroissant des étoiles "stars" des documents du "review" fait par le user. A l'aide de la fonction "forEach" on n'affiche que l'id du business, le commentaire du user, le nombre des étoiles qu'il a émis pour ce business et le degré de "useful" spécifié.
Indexes :
Ici, on a choisi de mettre des indexes sur les champs "useful" et "stars" de la collection "review", car review est une collection trop lourde de 4.5GO, et sans les indexes, les requêtes prennent beaucoup de temps.

— Requête 4 :
Langage du système :

```
db.user.aggregate([
  {
    $lookup:
```

```
        {
          from: "review",
          let: { user_id_user: "$user_id", average_stars_user: "
              ↪ $average_stars" },
          pipeline: [
             { $match:
                { $expr:
                   { $and:
                      [
                        { $eq: [ "$user_id", "$$user_id_user" ] },
                        { $gte: [ "$$average_stars_user","$stars" ] }
                      ]
                   }
                }
             },
             { $project: { user_id:1, stars:1, text:1, business_id:1} }
          ],
          as: "req_data"
        }
    }
]).forEach(function(doc){if(doc.req_data.length>0) {
  print("***\nuser_id=",doc.user_id);
  doc.req_data.forEach(function(d){print("\nbusiness_id=",d.business_id,"\
      ↪ nstarts=",d.stars,"\ntext=",d.text)})})})
```

Langage naturel :

Cette requête fait une jointure entre la collection "User" et la "Review" sur l'égalité du user_id et tel que "average_stars > stars", ces conditions de jointure sont utilisées avec le "pipeline". A la fin, on affiche par groupement sur le user id, la projection sur le business_id, les review.stars et les "text" (commentaires) qu'il a émis dans le "review" sur le business_id.
Indexes :
Ici, on utilise les indexes sur le "user_id" des deux collections car les collections sont lourdes, les "stars" car la review est trop grande (4.5GO), et le text, car la requête est trop complexe on a jugé nécessaire d'ajouter les indexes pour économiser beaucoup de temps.

— Requête 5 :

Langage du système :

```
db.business.find(
{"attributes.RestaurantsDelivery":"True","is_open":1},
{"name":1,"stars":1,"_id":0,"categories":1}).sort({"stars":-1}).
forEach(function(doc){ print(
  "name="+doc.name,",",
  "stars="+doc.stars,",",
  "categories="+doc.categories)})
```

Langage naturel :
Cette requête utilise les documents imbriqués et recherche les restaurants qui offrent le service de la livraison et qui sont ouverts, parmi les "business" et les affichent par ordre décroissant des stars, tout en faisant une projection sur le nom du restaurant (ou business), le nombre de ses stars et sa catégorie.
Indexes :
Ici, on utilise les indexes sur "stars" car ce champs est utilisé deux fois dans la querie1 et la querie5, et donc vu son utilisation fréquente, on a jugé intéressant de créer des indexes.

Bibliographie

[1] *Kaggle : Your Home for Data Science*. URL : https://www.kaggle.com/.

[2] *MongoDB Docs*. URL : https://docs.mongodb.com/.

[3] *Yelp Dataset Description*. URL : https://www.yelp.com/dataset/documentation.