Compte rendu TP 2 : MongoDB

Antonin Durand
INF3

Table des matières

Introduction 2
Demande 1 (Version 1) : 3
Demande 1 (Version 2) : 6
Demande 2 : 8
Demande 3 : 9

Introduction

Pour réaliser ce TP, nous avons créé et maintenu un dépôt Github. Ce dernier contient donc tous les scripts et les données relatives à la création de la base de données MongoDB.

Lien : DurandAntonin/NouveauxParadigmesBasesDeDonneesFI3 (github.com)

Le premier objectif de ce deuxième TP est d'importer les données contenues dans une base de données relationnelle Postgres, pour les stocker dans une base de données MongoDB.
Le deuxième objectif de ce TP est de montrer pourquoi une bd de documents est plus efficace dans notre problème.

MongoDB est une base de données de documents, dont son fonctionnement diffère grandement d'une base de données relationnelles. Les documents d'une collection (équivaut à un tuple en SQL) sont sous format JSON, et cette structure nous permet de résoudre les problèmes évoqués lors du première modélisation de la base de données réseaux de transport, tels que
En effet, l'avantage principal d'une base de données de documents comme MongoDB, dans notre cas, est qu'il est possible de stocker informations dans des structures hiérarchiques, des listes, en partie possible grâce à la redondance des données.
De plus, la syntaxe d'une requête dans MongDB nous permet de stocker directement dans des variables ou même d'effectuer des requêtes récursives.
Cela nous permet de répondre aux dernières questions du TP comme « *le nombre minimal de changements pour un trajet entre deux stations ?* », ou encore « *quelles sont les stations X et Y dont le trajet le plus simple demande un nombre de changements minimal?* ». Questions quasiment impossibles à répondre dans une base de données relationnelles du fait qu'il y a une relation d'ordre, d'antériorité, entre les différents arrêts (tuple dans la table).

Demande 1 (Version 1) :

Proposer une organisation des données du premier TP sous la forme d'une base de données MongoDB. L'organisation vise à permettre l'évaluation efficace des requêtes données dans le premier TP. Il faut définir :

1) *Les collections*
2) *Pour chaque collection, expliquer la structure des documents (Objet JSON). Donner un exemple*

Avant de commencer à réfléchir à la structure de notre base de données MongoDB, nous avons étudié les tables SQL que nous avions à notre disposition.
Nous avons donc interprété la fonction de chaque table dans la base de données de Ile De France Mobilités :

small_arrets_lignes ◊ Chaque tuple correspond à un arrêt/station, avec les informations de ce dernier (longitude, latitude, ville, code postal)
small_routes ◊ Chaque tuple correspond à une ligne avec son nom
small_trips ◊ Chaque tuple correspond à un train dans une ligne (C, H, ...) mais à des horaires différents
small_stop_times ◊ Chaque tuple correspond à un arrêt pour un train donné à un horaire donné. On a aussi le numéro de l'arrêt dans le trajet du train (1 pour le départ, 20 pour le terminus par exemple)

Nous avons également interprété l'utilité des champs les plus importants retrouvés dans chaque table :

route_id ◊ id d'une ligne (B, C, …)
trip_id ◊ id du train, correspond à un des trains en fonction de son heure de départ
trip_headsign ◊ correspond à l'identifiant d'un train (défini par IDFM)
stop_id ◊ id d'un arrêt
stop_sequence ◊ ordre d'arrivée d'un train aux arrêts de son trajet

Après réalisé cette étude sur les tables de la base de données d'origine, nous avons pu imaginer une structure pour implémenter les données dans une base de données MongoDB.

Nous avons donc choisi de définir deux collections pour former notre première base de données MongoDB :

- La première collection « **collectionTrips** » permet de stocker des informations issues de plusieurs tables sql. Chaque document représente le trajet effectué par un bus pour un trip d'une ligne. On aura par exemple un document pour le bus de 8h01 pour le trip MONA présent sur la ligne C.
Chaque document contient ainsi les informations du train, de la ligne, ainsi qu'une liste d'arrêts et les horaires de ce dernier.

- La deuxième collection « collectionRoutes » permet de stocker les noms de chaque ligne. Il était aussi possible de stocker ces informations dans chaque document de la première collection, cependant, il y aurait trop de redondances.

collectionTrips :


```
{
        "route_id" : "IDFM:C01727",
        "trip_id" : "IDFM:TN:SNCF:28bf25f5-5248-48be-96b0-c62b689c4543",
        "trip_headsign" : "MONA",
        "listeArrets" : [
                {
                        "stop_id" : "IDFM:monomodalStopPlace:44411",
                        "stop_name" : "Pontoise",
                        "stop_lon" : "2.0957859968423524",
                        "stop_lat" : "49.04673337596684",
                        "OperatorName" : "SNCF",
                        "Nom_commune" : "Pontoise",
                        "Code_insee" : "95500",
                        "arrival_time" : "12:58:50",
                        "departure_time" : "12:58:50",
                        "stop_sequence" : "1",
                },
                {
                        "stop_id" : "IDFM:monomodalStopPlace:47599",
                        "stop_name" : "Saint-Ouen-l'Aumône”,
                        "stop_lon" : "2.105230857870794",
                        "stop_lat" : "49.04530690087292",
                        "OperatorName" : "SNCF",
                        "Nom_commune" : "Saint-Ouen-l'Aumône",
                        "Code_insee" : "95572",
                        "arrival_time" : "13:00:20",
                        "departure_time" : "13:01:00",
                        "stop_sequence" : "2",
                }
        ]
}
```


collectionRoutes:


```
{
        "route_id" : "IDFM:C01727",
        "route_short_name" : "C",
        "route_long_name" "C":
}
```

Conclusion :

Cette première version de la base de données des réseaux de transport dans MongoDB est fonctionnelle, cependant deux problèmes se posent.
Le premier est l'importante redondance des données qui est excessive ici. En effet, on crée 1 document par trip_id et par route_id, où trip_id dépend de trip_headsign, trip_id et de l'horaire d'arrivée du premier arrêt. Ainsi, pour un même trip_headsign, comme MONA, plusieurs dizaines, voire centaines de documents seraient crées. Cela a pour conséquences la multiplication des données identiques dans plusieurs documents, comme les informations des arrêts, et la lenteur de notre programme pour convertir les données de la bd postgres en Json pour les insérer dans cette collection.
Le deuxième problème est la taille des documents qui pouvaient atteindre plusieurs MO. Ainsi, cette base de données n'est pas maintenable si l'on devait stocker les informations de toutes les lignes, dans plusieurs régions par exemple.

Nous avons donc réfléchi à une autre structure de cette collection, qui va être présentée dans la partie **Demande 1 (Version 2)**.

Demande 1 (Version 2) :

Dans cette nouvelle version de la structure de notre base de données MongoDB, nous avons décidé de garder une structure sur deux collections, nous avons gardé le même schéma pour la collection **collectionRoutes**, les changements majeurs se retrouvent donc dans la collection **collectionTrips**.

- La collection « **collectionRoutes** » sert toujours à stocker les identifiants des différentes lignes pour les réutiliser dans notre **CollectionTrips**.

- Quant à la collection « collectionTrips », sa structure change. En effet, comme nous l'avons énoncé dans la partie précédente, cette collection produisait trop de documents, avec trop de redondances, et était moins pertinente pour les demandes de la partie 3. Nous avons donc modifier la structure de la collection, de telle sorte qu'on a maintenant 1 seul document par trip_headsign. Ce dernier contient toujours la liste des arrêts pour le trip_headsign, mais chaque arrêt possède maintenant une liste des horaires pour lesquels le train passe.
Ainsi, nous avons 1 document pour représenter le trajet MONA sur la ligne C, avec une liste des arrêts, et pour le premier arrêt « Pontoise », il y a la liste des horaires de passage (8h01, 8h15, …), avec à chaque fois l'identifiant du train pour cet horaire.

collectionRoutes:


```
{
        "routeId" : "IDFM:C01727",
        "routeShortName" : "C",
        "routeLongName" :"C",
}
```


collectionTrips:


```
{
        "routeId" : "IDFM:C01727",
        "tripHeadsign" : "MONA",
        "idArretDepart" : " IDFM:monomodalStopPlace:44411",
        "listeArrets" : [
                {
                        "stopId" : "IDFM:monomodalStopPlace:44411",
                        "stopName" : "Pontoise",
                        "stopLon" : "2.095785996842352 4",
                        "stopLat" : "49.04673337596684",
                        "operatorName" : "SNCF",
                        "nomCommune" : "Pontoise",
```

```
                "codeInsee" : "95500",
                "stopSequence" : "1",
                "listArrivalsTime" : [
                        {
                                "arrivalTime" : "12:58:50",
                                "departureTime" : "12:58:50",
                                "tripId" = "IDFM:TN:SNCF:28bf25f5-5248-48be-96b0-
                        c62b689c4543",

                        }
                ]
        },
        {
                "stopId" : "IDFM:monomodalStopPlace:47599",
                "stopName" : "Saint-Ouen-l'Aumône”,
                "stopLon" : "2.105230857870794",
                "stopLat" : "49.04530690087292",
                "operatorName" : "SNCF",
                "nomCommune" : "Saint-Ouen-l'Aumône",
                "codeInsee" : "95572",
                "arrivalTime" : "13:00:20",
                "departureTime" : "13:01:00",
                "stopSequence" : "2",
                "listeArrivalsTime" : [
                        {
                                "arrivalTime" : "13:00:20"
                                "departureTime" : "13:01:00",,
                                "tripId" = "IDFM:TN:SNCF:28bf25f5-5248-48be-96b0-
                        c62b689c4543",
                        },
                ]

        }
    ]
}
```


Conclusion :

Cette nouvelle structure permet donc d’avoir les mêmes accès aux données qu’avec la structure précédente, tout en produisant moins de documents, et en augmentant la rapidité de notre programme à produire plus de documents. Cela rend donc cette dernière plus viable et c’est celle que nous allons utiliser dans les demandes 2 et 3.

Demande 2 :

Créer des scripts pour restructurer les données du TP2 conformément à la structure de document (Objet JSON) choisie. Les scripts peuvent être définis dans un langage choisi librement.

Tous les scripts qui ont été rédigés pour réaliser ce TP se trouvent sur le dépôt GitHub référencé au début de ce compte rendu.

Cette demande a été réalisée en deux parties.

La première partie est en PHP et permet de convertir les données contenues dans la bd Postgres, en fichiers Json reflétant la structure des 2 collections.
La base de données Postgres contient les données issues du premier dataset qui nous a été donnée, car il permet selon moi d'être plus fidèle à la réalité, avec ses nombreux trip_id et trip_headsign.
J'ai aussi crée plusieurs classes en PHP qui reflètent la structure Json qu'auront les documents des 2 collections :
- La première classe est « **CollectionLignes** », contenant un champ qui est une liste d'instances de la classe « **Lignes** » contenant les noms d'une ligne (C, A, …).
    Ces 2 classes me permettent de créer des documents en Json pour la collection « collectionRoutes ».

- Ensuite, on a eu deuxième classe « **CollectionTrips** », contenant un champ qui est une liste d'instances de la classe «**TrainHeadsign** ». Cette dernière est composée de champs comme la route_id, trip_headsign, id_arret_depart, ainsi qu'une liste d'arrêts.
Chaque élément de cette liste est une instance de la classe « **Arret** », et permet de stocker les différentes informations d'un arrêt, comme son id, nom, sa longitude, latitude, …
Parmi ses champs, on a un champ de type liste contenant la liste des horaires pour cette arrêt et pour ce trip_headsign. Chaque objet de cette liste est une instance de la classe « **ArrivalsTime** », permettant de stocker les horaires de départ, d'arrivée ainsi que l'identifiant du train pour cet horaire, et pour cet arrêt.
Si on remonte jusqu'à la classe « CollectionTrips », on obtient une classe qui permet de créer des objets ayant la même structure que la collection MongoDB « collectionTrips ».
Enfin, chaque classe possède une méthode « **serialize** », qui comme son nom l'indique, permet de transformer « le format » de l'objet. Dans notre cas, cette méthode retourne l'objet sous format json.

Après avoir créer ces différentes classes reflétant la structure des collections, j'ai crée un script en PHP « **scriptPostgresToMongoDb** » qui se connecte au serveur Postgres, à l'aide de la classe « **PostgreSQL** » que j'ai créée pour le TP2, et exécute des requêtes SQL pour récupérer les différentes données de telle sorte qu'on puisse créer des objets des classes énoncées précédemment. A noter, que pour éviter que ce script ne mettre trop de temps à s'exécuter, 20 horaires pour chaque arrêt, maximum.
De plus, 10 documents seront présents par fichier json pour éviter de surcharger la base de données MongoDB lors de l'insertion des documents dans la deuxième partie.

Nous avons utilisé la première version du dataset : « new_dataset » pour réaliser l'insertion dans notre base de données.
Nous avons séparé les documents en fichiers distincts, chaque fichier contenant 10 documents.

La deuxième partie de cette demande est l'insertion des documents contenus dans les fichiers json générés précédemment. Pour ce faire, la librairie **pimongo** a été utilisée car cette dernière permet de manipuler une base de données MongoDB, mais ce qui nous intéresse le plus est la connexion au serveur, et de la base de données, mais aussi et surtout la méthode « **insert_many** » qui permet d'insérer dans une collection, un ou plusieurs documents à partir d'un fichier json.
Le script en python « **script_insertion_mongodb_json** », s'occupe donc de se connecter à la bonne bd dans MongoDB, de load chaque fichier json contenant 20 documents, puis de les insérer dans la bonne collection.

Demande 3 :

Exprimer les requêtes données dans le premier TP sous la forme des requêtes MongoDB.

Requête 1: Étant donnée une ligne X (par exemple, le RER B), quelles sont ses stations?

```
db.collectionTrips.distinct("listeArrets.stopName", { "routeId": "IDFM:C01999" });
```

Requête 2: Quelles sont toutes les stations situées au sud (géographiquement) d'une station X donnée?

```
db.CollectionTrips.distinct("listeArrets", {"listeArrets.stopLat" : {$lt : '49'}})
```

Requête 3: Quelles sont toutes les stations au sud d'une station X donnée et qui sont desservies par le même train?

```
db.collectionTrips.find({ "listeArrets.stopId": "IDFM:monomodalStopPlace:44411" }, { "listeArrets": 1 })
```

Requête 4: Étant donné deux stations X et Y, y a-t-il une connexion directe de X à Y (c'est une question non triviale, prenez par exemple les stations Robinson et Massy-Palaiseau)?

```
db.collectionTrips.find({"listeArrets.stopId": { $in: ["IDFM:monomodalStopPlace: 43115","IDFM:monomodalStopPlace: 43238"] }}, { "listeArrets": 1 })
```

Requête 5: Étant donné deux stations X et Y de la même ligne, combien de stations les séparent? (Attention: requête imprécise)

```
db.collectionTrips.find({"routeId": "IDFM:C01727","listeArrets.stopId": { $in: ["id_station_X","id_station_Y"] }}, {"listeArrets": 1})
```

Requête 6: Étant donné deux stations X et Y pour lesquelles il existe une connexion directe, quelle est la durée minimale d'un trajet de X à Y?

```
db.collectionTrips.aggregate([
 {
  $match: {
   "routeId": "IDFM:C01727",
   "listeArrets.stopId": { $in: ["id_station_X", "id_station_Y"] }
  }
 },
 {
  $unwind: "$listeArrets"
 },
 {
  $match: {
   "listeArrets.stopId": { $in: ["id_station_X", "id_station_Y"] }
  }
 },
 {
  $group: {
   _id: null,
   minDuration: { $min: "$listeArrets.listeArrivalsTime.departureTime" }
  }
 }
])
```

Requête 7: Étant donné deux stations X et Y pour lesquelles il n'existe pas de connexion directe, y a-t-il une connexion avec un seul changement entre les deux stations?

```
db.collectionTrips.aggregate([
 {
  $match: {
   "routeId": "IDFM:C01727",
   "listeArrets.stopId": { $in: ["id_station_X", "id_station_Y"] }
  }
 },
 {
  $unwind: "$listeArrets"
 },
 {
  $match: {
   "listeArrets.stopId": { $in: ["id_station_X", "id_station_Y"] }
  }
 },
 {
  $group: {
   _id: "$_id",
   stops: { $addToSet: "$listeArrets.stopId" },
   count: { $sum: 1 }
  }
 },
 {
  $match: {
   stops: { $all: ["id_station_X", "id_station_Y"] },
   count: 2
  }
 }
])
```

Requête 8: Combien de changements de ligne sont nécessaires pour aller de la station X à la station Y?

```
db.collectionTrips.aggregate([
 {
  $match: {
   "listeArrets.stopId": { $in: ["id_station_X", "id_station_Y"] }
  }
 },
 {
  $unwind: "$listeArrets"
 },
 {
  $match: {
   "listeArrets.stopId": { $in: ["id_station_X", "id_station_Y"] }
  }
 },
 {
  $group: {
   _id: "$_id",
   routes: { $addToSet: "$routeId" }
  }
 },
 {
  $project: {
   _id: 0,
   changeCount: { $size: "$routes" }
  }
 }
])
```

Requête 9: Quel est le nombre maximal de changements pour un trajet entre deux stations? (pire cas)

```
db.collectionTrips.aggregate([
 {
  $unwind: "$listeArrets"
 },
 {
  $group: {
   _id: "$_id",
   stops: { $addToSet: "$listeArrets.stopId" },
   routes: { $addToSet: "$routeId" }
  }
 },
 {
  $project: {
   _id: 0,
   changeCount: { $size: "$routes" },
   stops: 1
  }
 },
 {
  $sort: { changeCount: -1 }
 },
 {
  $limit: 1
 }
])
```

Requête 10: Quelles sont les stations X et Y pour lesquelles le trajet le plus simple nécessite un nombre maximal de changements?

```
db.collectionTrips.aggregate([
 {
  $unwind: "$listeArrets"
 },
 {
  $group: {
   _id: "$listeArrets.stopId",
   routes: { $addToSet: "$routeId" }
  }
 },
 {
  $project: {
   _id: 1,
   changeCount: { $size: "$routes" }
  }
 },
 {
  $sort: { changeCount: -1 }
 },
 {
  $limit: 1
 }
])
```