Compte rendu TP 2 : MongoDB

William Zehren

INF3

Table des matières

Demande 1 (Version 1) : 3
Demande 1 (Version 2) : 5
Demande 2 : 8
Demande 3 : 9

Pour réaliser ce TP, il a fallu créer et maintenir un dépôt Github. Ce dernier contient donc tous les scripts et les données relatives à la création de la base de données MongoDB. Le code ayant été réalisé en partenariat avec Antonin Durand, nous utilisons un dépôt commun et un code commun. Cependant, notre compte rendu et la compréhension que nous avons eue de ce TP est différente.

Lien : DurandAntonin/NouveauxParadigmesBasesDeDonneesFI3 (github.com)

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, il a été nécessaire d'étudier en premier lieu les tables SQL à notre disposition. Il a donc été nécessaire de commencer par essayer d'interpréter la fonction de chaque table dans la base de données de Ile De France Mobilités :

small_arrets_lignes → Stocke des tuples correspondants à des arrêts/stations, avec les informations de ces derniers (longitude, latitude, ville, code postal)
small_routes → Stocke des tuples correspondants à des lignes avec leur nom
small_trips → Stocke des tuples correspondants à des trains dans diverses lignes (C, H, ...) mais à des horaires différents
small_stop_times → Stocke des tuples correspondants à des arrêts pour différents trains donnés à des horaires donnés. On a aussi le numéro de l'arrêt dans le trajet de chaque train (1 pour le départ, 20 pour le terminus par exemple)

Il a également fallu interpréter 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, il a été possible d'imaginer une structure pour implémenter les données dans une base de données MongoDB.

Il a donc fallu choisir de définir deux collections pour former la première base de données MongoDB :

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",
		}
	]
}
```


Cette collection servira à déterminer le trajet emprunté par un train dans le bon ordre d’arrêt, cela en fonction de sa ligne (C, B, …), de son arrêt de départ et d’arrivée et de son horaire de départ/arrivée.
Cette collection stocke également les diverses informations liées à chaque arrêt.
collectionRoutes:


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

Cette collection servira à stocker les identifiants des différentes lignes pour les réutiliser dans la collection **collectionTrips**.

Conclusion :

Cette première version de la base de données est déjà utilisable et fonctionnelle. Cependant, sa structure fait qu'elle contiendra trop de redondances qui peuvent être supprimées en restructurant la base de données. Cette structure de base de données produit également trop de documents, compte tenu de la quantité de données à notre disposition.
En effet, l'utilisation de cette structure favorise la production excessive de documents car ils sont créés en fonction du **tripId**. Il en a finalement été déduit que pour réduire la quantité de documents, il était plus intéressant de faire les documents en fonction du **tripHeadsign**. La structure de notre base de données a donc été modifiée pour régler ce problème.

Demande 1 (Version 2) :

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

collectionRoutes:


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


Cette collection sert toujours à stocker les identifiants des différentes lignes pour les réutiliser dans la collection **CollectionTrips**.

collectionTrips:


```
{
        "routeId" : "IDFM:C01727",
        "tripHeadsign" : "MONA",
        "idArretDepart" : " IDFM:monomodalStopPlace:44411",
        "listeArrets" : [
                {
                        "stopId" : "IDFM:monomodalStopPlace:44411",
                        "stopName" : "Pontoise",
                        "stopLon" : "2.0957859968423524",
                        "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",
                        },
                ]

        }
    ]
}
```


Dans cette version de la collection **collectionTrips**, il en a été déduit que le **tripId** pouvait uniquement servir à retrouver l'heure d'arrivée et de départ du train (**arrivalTime** et **departureTime**). Ces attributs ont donc été réunis dans une liste d'attributs appelée **listeArrivalsTime**.
Pour résumer, chaque document contient les informations d'un train avec la liste des arrêts pour chaque arrêt, une liste des horaires de passage pour chaque train.

Conclusion :

Cette nouvelle structure permet donc d'avoir les mêmes accès aux données qu'avec la structure précédente, tout en produisant moins de documents. Cela rend donc cette dernière plus viable et c'est celle qui va être utilisée pour réaliser 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.

Pour réaliser les scripts d'insertion dans la base de données, la première version du dataset : « new_dataset » à été utilisée
Les documents ont été séparés en fichiers distincts, chaque fichier contenant 10 documents.

Plusieurs classes ont donc été créés(Voir sur le dépôt GitHub), chacune représentant la partie qu'elle concerne de notre base de données. Par exemple, la classe PHP **ArrivalsTime** contient exactement les mêmes champs que chaque objet contenu dans le champ **listArrivalsTime**.

Chaque classe (voir les différentes classes sur le dépôt GitHub) contient une méthode permettant de sérialiser (transformer en JSON) toutes les données stockées dans notre base de données PostgreSQL.

Chaque classe contient également des méthodes appelée Getters et Setters pour chaque champ de cette dernière. Ils permettent d'interagir avec les champs des classes tout en respectant l'encapsulation.

Le script permettant de transformer nos données SQL en JSON (voir dépôt GitHub : scriptPostgresToMongoDB).
Le script va donc en premier lieu se connecter à PostgreSQL pour ensuite exécuter des requêtes vers ce dernier pour récupérer les données qui y sont stockées.
Il va ensuite transformer ces données en différents objets qui vont ensuite être transformés en JSON grâce aux méthodes écrites dans les différentes classes correspondant aux tables.
Il utilise donc les différentes classes préalablement créées pour récupérer les champs correspondants sous forme de chaine JSON.

Pour chaque arrêt, le script d'insertion ne prend cependant que les 20 premiers trains en fonction de leur heure d'arrivée à l'arrêt (**tripId**) pour des raisons de temps d'insertion.

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
  }
])
```