Projet Films-Cinéma

Matière : Système de Gestion de Documents

Participants : Yann MOURELON
Daniel PINSON

Directeur du projet : Nadine CULLOT

Langage de programmation utilisé : Python, Mongo
Librairies utilisées : pymongo, datetime

2023

Introduction **3**
Conception de la base de donnée **3**
Cahier des charges des fonctionnalités 4
Diagramme UML 4
Exemple de document JSON 5
Requête Mongodb **7**
Pymongo **8**
Conclusion **8**

Introduction

Mongodb est une base de données orientée document et dite noSQL (not only SQL). Mongodb permet de manipuler des objets au format JSON sans schéma prédéterminé ce qui signifie que des clés peuvent être ajoutées sans devoir reconfigurer la base de données. Le format JSON est très facile d'utilisation bien que la lisibilité de ce type de document pour une base de données peut s'avérer assez difficile.

Exemple de structure de donnée SQL/JSON

SQL

ID	Nom	Prenom
1	MOURELON	Yann
2	PINSON	Daniel

JSON

```
{
    "_id": 1,
    "nom": "MOURELON",
    "prenom": "Yann"
},
{
    "_id": 2,
    "nom": "PINSON",
    "prenom": "Daniel"
}
```

L'objectif de ce projet est de monter une base de données permettant de faire de l'analyse décisionnelle concernant les salles de cinéma d'une ville en utilisant la technologie Mongo et de faire différents types de requêtes afin de montrer différents aspects de celle-ci. Ensuite nous devons passer par Pymongo qui permet de faire des opérations plus compliquées sur des collections.

Conception de la base de donnée

Le cahier des charges de notre base de données comporte les critères suivants. Une personne peut utiliser cette base de données pour faire une application afin que des utilisateurs puissent rechercher des films. Aussi un gestionnaire de cinéma peut manipuler cette base de données pour avoir des informations sur ceux-ci.

Il est possible de modéliser la base de donnée en ne faisant qu'une collection cinéma et en imbriquant les films dans ceux-ci, on utiliserait donc une base de donnée orientée cinéma avec imbrication. Cependant, ce genre de collection rendrait plus coûteux les requêtes du type "tous les films de X catégorie".

Une autre modélisation orientée film est possible mais rend les requêtes du type "Les cinémas qui sont ouverts à 12h difficile.

Nous avons donc décidé qu'une base de données centrée sur la collection **cinemas** est préférable à une centralisation sur la collection **films**. Nous avons aussi choisi de créer 2 collections différentes : **cinemas** et **films** avec une référence des films dans cinemas afin de ne pas surcharger la collection **cinemas** avec une répétition de **films**. Grâce à cela, nous avons une meilleure visualisation des données et une référence sur l'id d'un film nous permet de retrouver toutes les informations que l'on souhaite depuis **cinemas**.

Le gros défaut de faire plusieurs collections liées est qu'un *find* ne suffit pas. Heureusement, il existe une méthode appelée *aggregate* en Mongo nous permettant de faire aisément des liens entre plusieurs collections à l'aide de l'opérateur *$lookup*.

Diagramme UML

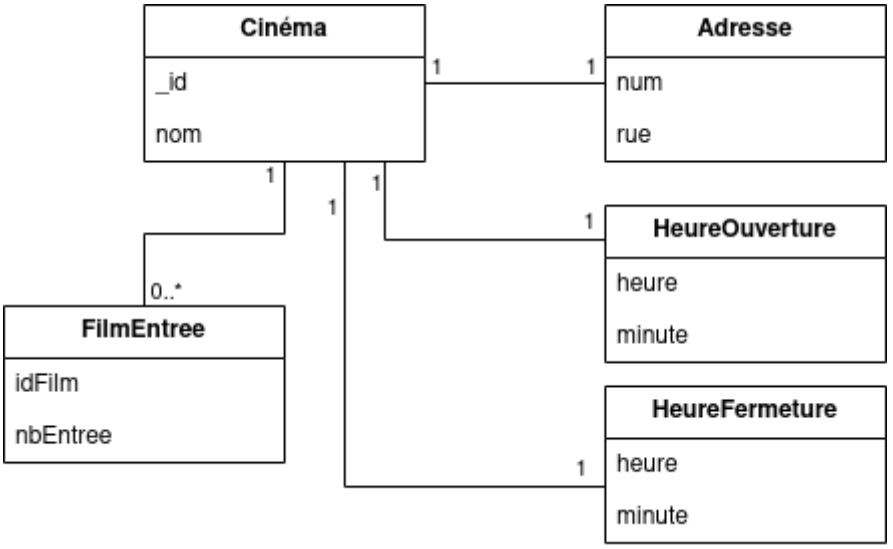

Schéma de la collection **cinemas**

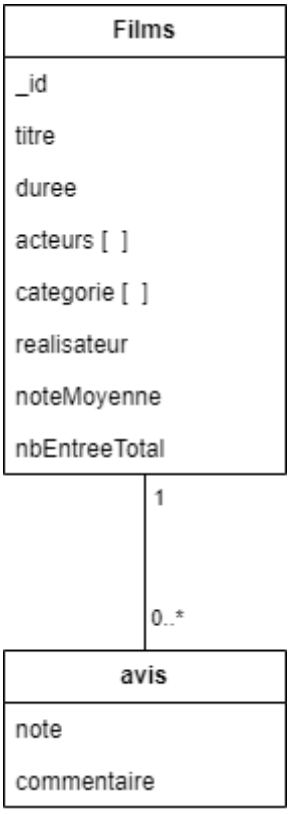

Schéma de la collection **films**

Exemple de document JSON

Exemple de document cinema


```
{
  "_id": 1,
  "nom": "Olympia",
  "adresse": {
    "num": 16,
    "rue": "avenue du Marechal Foch"
  },
  "heureOuverture": {
    "heure": 8,
    "minutes": 0
  },
  "heureFermeture": {
    "heure": 23,
    "minutes": 0
  },
  "filmEntrees": [
    { "idFilm": 1, "nbEntrees": 2000000 },
    { "idFilm": 3, "nbEntrees": 7000000 }
  ]
}
```


Exemple de document films


```
{
  "_id": 1,
  "titre": "Le Loup de Wall Street",
  "duree": 179,
  "realisateur": "Martin Scorsece",
  "acteurs": [ "Leonardo DiCaprio", "Jonah Hill", "Margot Robbie", "Kyle Chandler", "Rob Reiner" ],
  "categorie": [ "Drame", "Action", "Comédie", "Biographie" ],
  "noteMoyenne": 4,
  "avis": [
    {
      "note": 4,
      "commentaire": "'Le Loup de Wall Street' est un film fascinant qui nous plonge dans le monde sans scrupules de la finance. Les performances de Leonardo DiCaprio et de Jonah Hill sont remarquables, et leur chimie à l'écran est palpable. Martin Scorsese a réalisé un film captivant qui dépeint les excès et les dangers de la culture de Wall Street avec une précision choquante."
```

```
    },
    {
      "commentaire": "Le film 'Le Loup de Wall Street' est un véritable
tour de force de la part de Martin Scorsese et de son équipe. La
narration est intelligente et captivante, avec des moments d'humour noir
qui maintiennent l'attention du public tout au long de ses trois heures.
La performance de DiCaprio est particulièrement impressionnante, car il
incarne parfaitement le personnage sombre et complexe de Jordan
Belfort."
    },
    {
      "note": 3.5,
    }
  ],
  "nbEntreeTotal": 3009494
}
```


Requête Mongodb

L'objectif est de proposer un large panel de requêtes pouvant être nécessaire pour avoir des informations sur les films et leur diffusion, et des fonctionnalités analytiques jugées pertinentes.
En cela nous avons décidé de faire les requêtes suivantes qui sont dans le fichier requetes.txt :

1. Les films qui sont diffusés dans au moins 2 cinémas (pour donner une idée de sa popularité) (aggregate)
2. Le nombre d'entrée total de chaque chaque cinema (*Map-Reduce*)
3. Tous les avis d'un film (aggregate)
4. Tous les films de moins de 2h (aggregate)
5. Les films réalisés par X (find)
6. Les films de X catégorie (find)
7. Les X films les mieux notés (find et aggregate)
8. Ajout d'un film
9. Modifications des horaires d'ouverture du cinéma
10. Modifier le nombre d'entrées d'un film

Ainsi, il est possible d'avoir plus d'informations sur les films, de sorte à pouvoir décider quoi aller regarder. Par exemple, l'utilisateur peut voir les avis d'un film, savoir lesquels il peut aller voir en 2h… et un gestionnaire de cinéma peut voir le nombre d'entrées total de tous les cinémas de la ville.

Pour la 3ème requête, il nous a paru pertinent d'utiliser un *Map-Reduce*.

```
var map = function(){emit(this.nom, this.filmEntrees);}
var reduce = function(nomCinema, filmEntrees){
    var somme = 0;
    for (let i = 0; i < filmEntrees[0].length; i++) {
        somme += filmEntrees[0][i].nbEntrees;
    }
    return somme;
}
db.cinemas.mapReduce(map, reduce, {out: {inline: 1}})
```

Pour cela, nous définissons les fonctions *Map* et *Reduce*.
Map va donner un ensemble clé-valeur à la fonction *reduce*, et on donne ici les noms de cinémas en clés, ainsi que le tableau contenant tous les films et les entrées de celui-ci en valeur.
Reduce reçoit un couple clé-valeur de ce qui est envoyé par map. On initialise une somme à 0, puis on calcule la somme des entrées de toutes les entrées du cinéma, que l'on renvoie comme résultat. *Reduce* étant appelé jusqu'à ce que tous les couples clé-valeur aient été appelés, on récupère bien le nombre d'entrées total de chaque cinéma.
Cependant, le paradigme *Map-Reduce* est devenu obsolète en MongoDB 5.0 et la documentation conseille plutôt d'utiliser des agrégations.
Dans ce projet, *Map-Reduce* n'est pas l'opération la plus pertinente car elle a été pensée pour pouvoir faire des requêtes sur des donnés dans un système distribué, or nous sommes ici en local.

Pymongo

Pymongo est une bibliothèque python permettant de communiquer facilement avec une base de données mongodb grâce à son large panel d'outils. Avec cette librairie, on peut utiliser le même type de requête que l'on utilise pour communiquer directement avec la base de données mais permet aussi, grâce à python, d'exécuter des requêtes et des manipulations de données plus complexes.

La notion de clé étrangère n'existe pas en mongodb, donc pour avoir accès aux informations se trouvant dans la collection film par le biais de la collection **cinemas**, nous devons faire correspondre l'élément `"idFilm"` avec l'identifiant `"_id"` qui constitue la clé primaire de la collection **films**. Python permet de faciliter ceci en stockant dans des variables les résultats des requêtes pour ensuite les manipuler comme de simples objets python que l'on connaît.

Nous avons créés 4 méthodes en python:

- `note_moyenne_cinemas(nom_cinema)`: ⇒ affiche la moyenne de tous les films joués dans un cinéma donné. Cela nous donne un aperçu de la qualité des films que le cinéma propose.
- `cinemas_ouvert_actuellement()`: ⇒ Récupère l'heure actuelle et affiche tous les cinémas de Dijon qui sont ouverts à cette heure-ci.

- `nombre_avis_Pos_Neg_Neutre(nom_film):` ⇒ Affiche le nombre d'avis positif / négatif / neutre déduit en fonction de leurs notes. Si cette note est > 2.5 alors l'avis est positif, si elle est < 2.5 alors l'avis est négatif et si elle est = 2.5 alors l'avis est neutre. En plus d'afficher le nombre d'avis positif, négatif et neutre d'un avis, on affiche aussi les avis complets en fonction de leur note.
- `creation_Commentaire():` ⇒ Lance une création de commentaire interactif en demandant le nom du film que l'on veut commenter, on entre ensuite la note comprise entre 0 et 5 par tranche de 0.5 puis on nous demande de rentrer un commentaire. Si celui-ci est vide alors on crée un avis ne contenant qu'une note.

Pour notre méthode `note_moyenne_cinemas(nom_cinema):` Nous avons dans un premier temps récupéré les identifiants des films contenus dans la collection **cinemas** pour pouvoir les faire correspondre avec les identifiants de la collection **films**. Cela permet de récupérer les notes des films qui ont été joués dans un cinéma afin de faire une moyenne des notes qui représenterait la qualité moyenne des films joués dans les différents cinémas. Ici, python nous offre une meilleure visualisation des résultats et une simplification de traitement des données récupérées. Dans les autres requêtes, nous utilisons python pour pouvoir récupérer des informations extérieures et faire des requêtes plus pertinentes.

Conclusion

Comme c'est la première fois que l'on crée et manipule une base de données noSQL, nous avons dû penser différemment la structure utilisée pour gérer nos données. L'utilisation de mongoDB se fait sans trop de difficulté grâce à une documentation en ligne très bien structurée et un moteur reposant sur le paradigme clé-valeur qui donne un accès direct à une donnée spécifique. Cependant la gestion de l'affichage et des éléments que les requêtes nous retourne n'est pas intuitive au premier abord, mais une courte période d'adaptation est suffisante pour en comprendre le fonctionnement.

L'utilisation d'une librairie mongo (ici, Pymongo) nous ouvre énormément de possibilités sur la façon de manipuler les données et d'en extraire des informations pertinentes, ce qui facilite grandement le développement d'applications performantes basées sur MongoDB. Par exemple, avec Pymongo, on peut effectuer des requêtes complexes, trier et filtrer les données, utiliser des agrégations et des index pour optimiser les performances de la base de données, et bien plus encore. Tout cela permet de maximiser l'utilité et la flexibilité de MongoDB dans divers contextes d'application.