

May 2, 2024

UNIVERSITÉ DE BOURGOGNE

Rapport Système Gestion de Document

HAMMANI Assia

Table of Contents

<i>List of Figures</i>	ii
1 Introduction	1
2 1ère partie : Conception et mise en œuvre de la base de données MongoDB	1
2.1 Choix de la modélisation des données	1
2.2 Description des données modélisées	2
2.3 Fonctionnalités proposées	3
3 2ème partie : Requêtage de la base de données MongoDB	4
3.1 Requêtes d’insertion	4
3.2 Requêtes de mise a jour de document	5
3.3 Requêtes simple	7
3.4 Requêtes avancées	8
3.5 Utilisation de Map-Reduce	9
4 3ème partie : Python et pyMongo	10
4.1 Scripts Python	10

List of Figures

1	Diagramme de classe	2
2	Utilisation de createCollection	4
3	Utilisation de insertMany	5
4	Utilisation de updateOne pour ajouter un commentaire	5
5	Utilisation de updateOne pour modifier la note d'un commentaire	6
6	Utilisation de updateOne pour modifier le prix du billet	6
7	Correction de l'erreur d'id du film Joker	6
8	Modification prise en compte pour le film Joker	6
9	Utilisation de find et count	7
10	Utilisation de findOne	7
11	Utilisation de find et \$gt	7
12	Utilisation de \$exists	8
13	Utilisation de aggregate, \$gt, \$group et \$avg	8
14	Utilisation de aggregate	8
15	Utilisation de aggregate, \$unwind, \$match,\$group, count et \$project	9
16	Utilisation de MapReduce	10
17	Resultat du code fichier Script1.py	11
18	Resultat du code fichier Script2.py	11
19	Resultat du code fichier Script6.py	11

Développement d'une base de données MongoDB et d'un système d'analyse décisionnelle pour les films de cinéma

1 Introduction

Ce projet vise à développer une base de données MongoDB et un ensemble de fonctionnalités permettant de réaliser de l'analyse décisionnelle sur les films de cinéma. L'objectif est de mettre en pratique les éléments vus en cours, notamment la conception et la mise en œuvre d'une base de données MongoDB, son requêtage, ainsi que l'utilisation du langage Python et du driver PyMongo pour implémenter des fonctionnalités avancées.

2 1ère partie : Conception et mise en œuvre de la base de données MongoDB

L'objectif est de concevoir et mettre en œuvre une base de données MongoDB adaptée aux besoins du projet. Ce cahier des charges présente les spécifications pour le développement d'une base de données MongoDB et d'un ensemble de fonctionnalités pour l'analyse décisionnelle des films de cinéma.

2.1 Choix de la modélisation des données

Nous avons choisi de modéliser les données avec une structure JSON imbriquée. Cela a permis une gestion efficace des informations cinématographiques pour plusieurs raisons.

Tout d'abord, l'imbrication des structures JSON permet de représenter clairement et de façon organisée l'ensemble des informations liées à un film. Les données telles que le titre, la description, la durée, ect..., sont ainsi regroupées de manière cohérente, ce qui facilite la gestion et la recherche d'informations spécifiques.

Cette approche permet de représenter les relations entre les différentes entités comme le réalisateur, les acteurs et les commentaires. Par exemple, en incluant les détails du réalisateur dans le document principal du film, nous pouvons accéder facilement à ces informations sans avoir à consulter une autre collection de données.

La manipulation des données est également simplifiée grâce à l'imbrication des structures. Pour supprimer un film par exemple, on supprime le film grâce à une de ses données comme son id, son titre, ect... Cela entraîne la suppression de toutes les informations associées, telles que les commentaires et les détails des acteurs.

Enfin, cette modélisation offre une grande flexibilité pour l'intégration de nouvelles fonctionnalités ou de champs supplémentaires. L'ajout d'informations sur les récompenses reçues par un film, par exemple, ne requiert que l'implémentation d'un nouveau champ dans la structure JSON existante sans perturber l'organisation initiale.

Ci dessous le diagramme de classe représentant la structure de la collection :

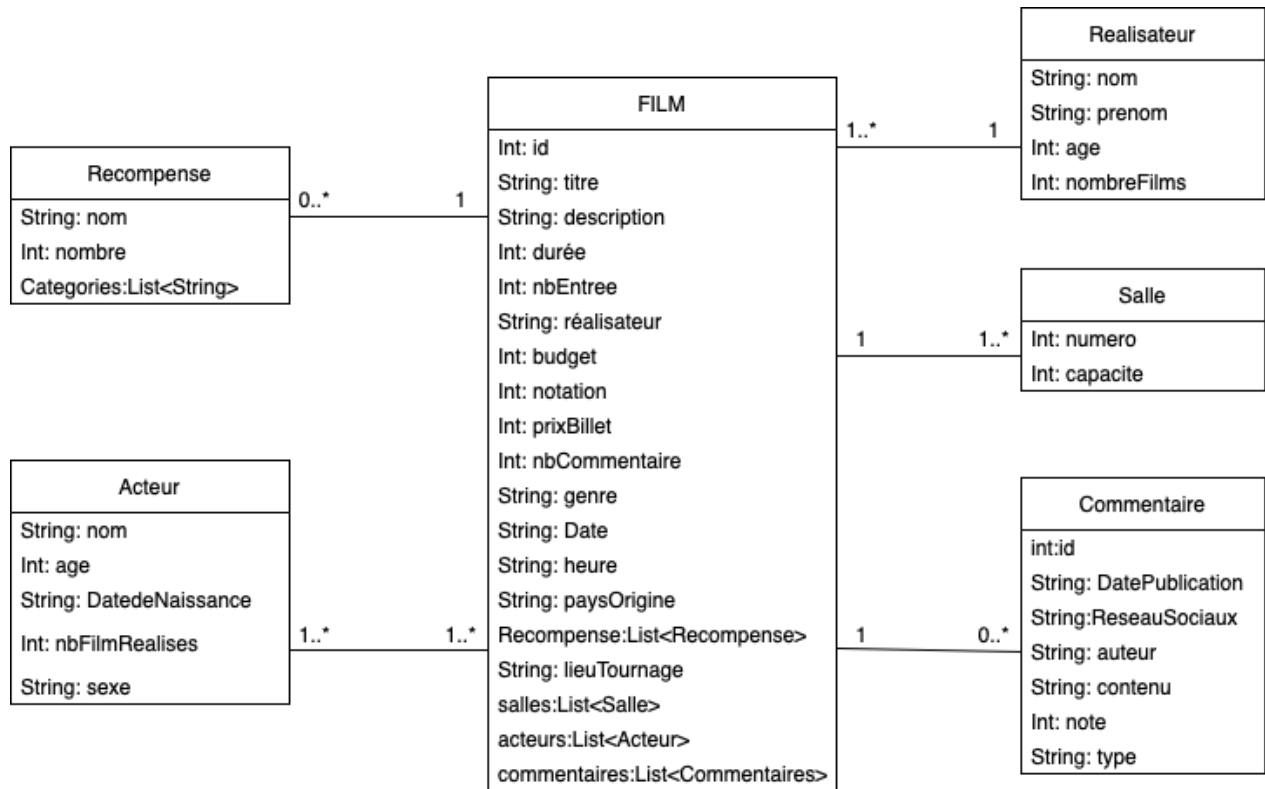


Figure 1: Diagramme de classe

2.2 Description des données modélisées

Nous avons une collection CentreFilm qui de base contient 3 documents après la première insertion, correspondant à trois films distincts.

Pour chaque film il y a:

id : identifiant unique attribué à chaque film dans la base de données. Permet d'identifier de manière unique chaque entrée

titre: titre du film

description: description du film

durée: durée du film en heures et minutes

nbEntree: nombre d'entrées vendues pour le film

réalisateur: objet imbriqué contenant les informations sur le réalisateur du film, telles que le nom, le prénom, l'âge et le nombre de films réalisés

budget: budget alloué à la production du film

notation: note moyenne donnée au film par les spectateurs

prixBillet: prix d'un billet pour voir le film

nbCommentaire: nombre total de commentaires publiés sur le film

genre: genre cinématographique auquel appartient le film

date: date de sortie du film

heure: heure de projection du film

paysOrigine: pays d'origine du film

Recompenses: liste des récompenses remportées par le film. Chaque récompense est représentée par un objet contenant le nom de la récompense, le nombre de fois qu'elle a été remportée et éventuellement des catégories spécifiques

lieuTournage: lieu où le film a été tourné

salles: tableau d'objets imbriqués, chaque objet représentant une salle avec des propriétés comme "numero" et "capacite"

acteurs: tableau d'objets imbriqués, chaque objet représentant un acteur avec des propriétés comme "nom", "age", "DatedeNaissance", "nbFilmRealises" et "sexe".

commentaires: tableau d'objets imbriqués, chaque objet représentant un commentaire contenant un identifiant unique, la date de publication, le réseau social sur lequel il a été publié, l'auteur, le contenu du commentaire, éventuellement une note attribuée et le type de commentaire.

Les propriétés comme "id", "titre", "description", "duree", "nbEntree", "budget", "notation", "prixBillet", "nbCommentaire", "genre", "date", "heure", "paysOrigine", "Recompenses", et "lieuTournage" sont des propriétés simples qui contiennent des valeurs comme des nombres ou des chaînes de caractères. La structure de données pour la propriété "Recompenses" varie selon les cas. En effet, si le film a remporté des récompenses, alors la propriété "Recompenses" est un tableau d'objets. Chaque objet représente une récompense spécifique avec des propriétés telles que "Nom", "Nombre" et éventuellement "Catégories". Sinon c'est un objet vide. Le document utilise donc principalement des objets imbriqués pour structurer les données de manière hiérarchique.

2.3 Fonctionnalités proposées

- Recherche par titre de film
- Filtrage par genre de film (ex : films de drame)
- Filtrage par date de sortie (ex : films sortis après une certaine date)
- Filtrage par pays d'origine (ex : films produits aux États-Unis)
- Recherche par réalisateur
- Recherche par acteur
- Filtrage par notation du film (ex : films avec une note supérieure à une certaine valeur)
- Filtrage par budget du film (ex : films avec un budget inférieur à un certain montant)
- Filtrage par durée du film

- Analyse des commentaires des spectateurs pour obtenir des informations sur l'opinion vis à vis du film (opinions positives/négatives, etc.)

En terme de statistique nous pouvons:

- Calculer la moyenne des notations des films
- Calculer le budget moyen des films
- Analyser la répartition des films par genre
- Évaluer les réalisateurs en fonction du nombre de films réalisés
- Identifier les acteurs les plus fréquemment impliqués dans des films à succès
- Analyser la satisfaction des spectateurs en fonction du budget des films

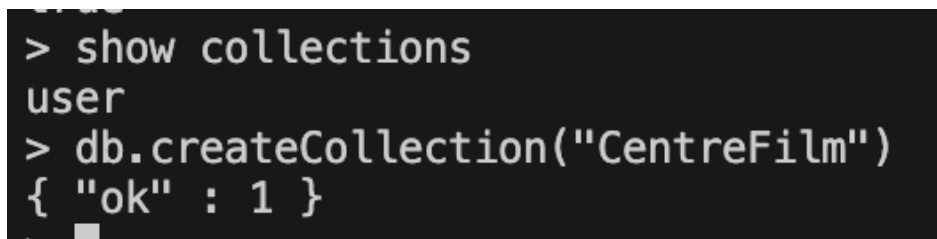
Grâce à ces fonctionnalités, les utilisateurs peuvent se renseigner et choisir le film qu'il souhaite visionner et les gestionnaires peuvent visualiser les statistiques des films proposées, analyser les tendances pour identifier les films populaires pour pouvoir ainsi continuer à satisfaire les utilisateurs.

3 2ème partie : Requêtage de la base de données MongoDB

L'objectif de cette partie est de proposer un ensemble de requêtes permettant d'obtenir des informations sur les films, leur diffusion,ect...

3.1 Requêtes d'insertion

On crée la collection: `db.createCollection("CentreFilm")`.



```
> show collections
user
> db.createCollection("CentreFilm")
{ "ok" : 1 }
```

Figure 2: Utilisation de createCollection

Puis pour insérer la collection dans la database: `db.CentreFilm.insertMany(le contenu de la collection)`.

Le résultat de cette requête indiquera que l'opération d'insertion a été réalisée avec succès. Chaque document inséré s'est vu attribuer un identifiant unique (ObjectId) généré par

MongoDB. Dans notre cas, trois documents ont été insérés avec succès, et les identifiants générés pour ces documents sont :

```
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("662644ae88a293412bd43929"),
    ObjectId("662644ae88a293412bd4392a"),
    ObjectId("662644ae88a293412bd4392b")
  ]
}
```

Figure 3: Utilisation de insertMany

3.2 Requêtes de mise à jour de document

Pour effectuer des modifications dans la collection nous utilisons des requêtes de mise à jour. Par exemple (cf figure 4), l'utilisateur Emile vient juste de voir le film Monk, il souhaite ajouter un commentaire à ce film. Nous effectuons donc la mise à jour pour prendre en compte ce nouveau commentaire.

```
}
> db.CentreFilm.updateOne(
...   { "titre": "Monk" },
...   {
...     $push: {
...       "commentaires": {
...         "id": 30,
...         "DatePublication": "19/10/2021",
...         "ReseauSociaux": "Twitter",
...         "auteur": "Emile",
...         "contenu": "Un film qui mêle humour et suspense.",
...         "note": 8,
...         "type": "textuel et note"
...       }
...     }
...   }
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Figure 4: Utilisation de updateOne pour ajouter un commentaire

Nous pouvons voir dans la figure 4 que le commentaire a bien été ajouté.

Pour modifier l'ancienne note du commentaire de Guimzy avec la nouvelle note 6 nous effectuons la requête suivante (cf figure 5):


```

... { "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.CentreFilm.updateOne(
...   { "titre": "Le parrain", "commentaires.auteur": "Guimzy" },
...   {
...     $set: {
...       "commentaires.$.note": 6
...     }
...   }
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>

```

Figure 5: Utilisation de updateOne pour modifier la note d'un commentaire

Pour modifier le prix du billet du film Le Parrain a 10 euros (cf figure 6):

```

> db.CentreFilm.updateOne(
...   { "titre": "Breaking Bad" },
...   { $set: { "prixBillet": 10 } }
... )
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>

```

Figure 6: Utilisation de updateOne pour modifier le prix du billet

Lors de l'insertion du quatrième document dans la collection, une erreur est ajoutée. En effet, l'id attribué à ce nouveau film est 20, ce qui correspond déjà à l'id d'un autre film existant dans la collection. Si l'on recherche un film par son id il sera impossible de savoir quel est le bon film, car deux films ont le même identifiant. Cela peut également poser problème en cas de suppression. Pour corriger cette erreur, il faut modifier l'id du quatrième film inséré dans la collection. La requête de modification de cette erreur est la suivante (cf figure 7):

```

> db.CentreFilm.updateOne({ "titre": "Joker" }, { $set: { "id": "40" } })
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }

```

Figure 7: Correction de l'erreur d'id du film Joker

Nous pouvons verifier si la modification a ete réalisé (cf figure 8):

```

> db.CentreFilm.find({"titre":"Joker"})
{ "_id" : ObjectId("6626aede3d2448e4354df2e6"), "id" : "40", "titre" : "Joker", "
até, bascule peu à peu dans la folie et devient le criminel psychotique connu sou
000 "réalisateur" : { "nom" : "Phillips", "prénom" : "Todd", "âge" : 40, "nombre

```

Figure 8: Modification prise en compte pour le film Joker

La modification a été réalisée avec succès.

3.3 Requêtes simple

Nous pouvons effectuer des requêtes simples pour rechercher un film, comptez le nombre de films réalisant une condition particulière, etc.

Par exemple, il y a une requête qui compte le nombre de film ayant comme genre le drame (cf figure 9):

```
> db.CentreFilm.find({"genre":"drame"}).count();  
1
```

Figure 9: Utilisation de find et count

Nous recherchons dans toute la collection le nombre de documents ayant comme genre de film le drame. Nous utilisons pour cela '.count()' qui compte le nombre de films correspondant à ce critère.

Nous pouvons écrire une requête qui affiche uniquement le titre du livre ayant comme id 10 (cf figure 10):

```
> db.CentreFilm.findOne({ id: 10 }, { titre: 1, _id: 0 })  
{ "titre" : "Monk" }
```

Figure 10: Utilisation de findOne

Cette requête recherche un document grâce à findOne() dans la collection qui a un champ id avec la valeur 10. Nous ne souhaitons pas afficher l'id dans le résultat, nous voulons seulement le titre donc nous avons spécifier dans la requête cette condition avec titre:1 et id:0.

Une autre requête consiste à rechercher et afficher le nom des réalisateurs ayant réalisés plus de 5 films (cf figure 11):

```
> db.CentreFilm.find({"réalisateur.nombreFilms": {$gt: 5}}, {"réalisateur.nom": 1, "_id": 0})  
{ "réalisateur" : { "nom" : "Andy" } }  
{ "réalisateur" : { "nom" : "Gilligan" } }  
{ "réalisateur" : { "nom" : "Coppola" } }  
{ "réalisateur" : { "nom" : "Phillips" } }
```

Figure 11: Utilisation de find et \$gt

Ci dessous une requête qui affiche les titres et récompenses des films ayant reçus des récompenses (cf figure 12):

```

> db.CentreFilm.find({"Récompenses": {$exists: true}}, {"titre": 1, "Récompenses": 1, "_id": 0}).pretty()
{
  "titre" : "Breaking Bad",
  "Récompenses" : [
    {
      "Nom" : "Primetime Emmy Awards",
      "Nombre" : 16,
      "Catégories" : [
        "Meilleure série dramatique"
      ]
    },
    {
      "Nom" : "Golden Globes Awards",
      "Nombre" : 8,
      "Catégories" : [
        "Meilleure série dramatique"
      ]
    }
  ]
}

```

Figure 12: Utilisation de \$exists

Dans cette requête l'opérateur \$exists est utilisé pour vérifier si le champ Récompenses existe dans chaque document. Nous ajoutons des conditions avec "titre": 1, "Récompenses": 1, "_id": 0 pour que le titre du film et les récompenses si elles existent, soient affichés et l'id ne soit pas affiché dans les résultats.

3.4 Requêtes avancées

Concernant les requêtes avancées, les requêtes d'agrégation permettent de regrouper, filtrer, trier et effectuer des calculs sur les données de manière plus avancée que les requêtes simples de recherche.

La requete ci dessous calcule le budget moyen des films qui ont un budget supérieur à 600 000 euros (cf figure 13):

```

> db.CentreFilm.aggregate([{$match: {"budget": {$gt: 600000}}}, {$group: {_id: null, moyenne: {$avg: "$budget"}}})
{ "_id" : null, "moyenne" : 21316666.666666668 }

```

Figure 13: Utilisation de aggregate, \$gt, \$group et \$avg

La méthode \$match dans le cadre de l'agrégation permet de sélectionner les documents qui correspondent à certaines conditions, ici la condition est que le champ budget soit supérieur à 600 000. Ce morceau de requête : \$group: _id: null, moyenne: \$avg: "\$budget" regroupe les films correspondants en un seul groupe grâce à \$group, _id: null spécifie que tous les documents sont regroupés ensemble, et l'opération \$avg calcule la moyenne du champ "budget" pour les documents regroupés.

Une autre requête consiste à afficher le nombre total d'acteur par sexe (cf figure 14):

```

> db.CentreFilm.aggregate([{$unwind: "$acteurs"}, {$group: {_id: "$acteurs.sexe", count: {$sum: 1}}}, {$project: {sexe: "$_id", count: 1, _id: 0}}])
{ "count" : 6, "sexe" : "feminin" }
{ "count" : 7, "sexe" : "masculin" }

```

Figure 14: Utilisation de aggregate

Dans cette requête nous utilisons l'opérateur d'agrégation `$unwind` pour décomposer le champ "acteurs" qui sont un tableau dans chaque document. Nous utilisons également l'opérateur `$group` qui regroupe les films d'acteurs décomposés en fonction du champ "acteurs.sexe", qui représente le sexe de chaque acteur. Pour chaque groupe, la requête calcule le nombre total d'acteurs en utilisant `$sum`. Enfin l'opérateur `$project` permet de restructurer les documents dans le résultat pour présenter les informations souhaitées.

Cette requête ci dessous permet d'afficher les noms des acteurs ayant joué dans plus de 2 films et calculer leur age moyen (cf figure 15):

```
> db.CentreFilm.aggregate([
...   {$unwind: "$acteurs"},
...   {$match: {"acteurs.nbFilmRealises": {$gt:2}}},
...   {$group:{
...     _id: null,
...     count: {$sum: 1},
...     ageTotal: {$sum: "$acteurs.age"}
...   }},
...   {$project:{
...     _id: 0,
...     nombreActeurs: "$count",
...     ageMoyenActeurs: {$divide:["$ageTotal", "$count"]}
...   }}
... ])
{ "nombreActeurs" : 13, "ageMoyenActeurs" : 60.76923076923077 }
>
```

Figure 15: Utilisation de aggregate, \$unwind, \$match,\$group, count et \$project

Dans cette requête nous utilisons `$unwind` avec le tableau acteurs ce qui permet de dérouler le tableau pour pouvoir traiter chaque acteur individuellement. Nous utilisons `$match` pour ne conserver que les acteurs ayant réalisé plus de 2 films. L'opérateur `$group` permet de filtrer les documents avec les conditions précédentes. Nous utilisons `count` pour calculer le nombre d'acteurs dans chaque groupe et `$sum` pour calculer l'age moyen de ces acteurs.

3.5 Utilisation de Map-Reduce

Nous avons réalisé un programme en map reduce qui affiche la somme totale des entrées pour tous les films.

```
var map = function() {
    emit("total", this.nbEntree);
};

var reduce = function(key, values) {
    return Array.sum(values);
};
```

```

db.CentreFilm.mapReduce(
    map,
    reduce,
    {
        out: { inline: 1 }
    }
);

```

Dans ce programme, on définit la fonction map avec `:var map = function()` Dans la fonction, nous utilisons `emit("total", this.nbEntree)` pour émettre une paire clé-valeur. La clé "total" est utilisée pour regrouper toutes les valeurs ensemble. Puis nous définissons la fonction reduce avec `var reduce = function(key, values)` La fonction prend une clé et une liste de valeurs en entrée. Elle retourne la somme de toutes les valeurs associées à la même clé. Enfin, `db.CentreFilm.mapReduce(...)` exécute l'opération MapReduce sur la collection CentreFilm. Les résultats du programme sont stockés dans la sortie spécifiée par `out: inline: 1`.

Nous avons donc ce résultat (cf figure 16):

```

... );
{
    "results" : [
        {
            "_id" : "total",
            "value" : 29669589
        }
    ],
    "ok" : 1
}

```

Figure 16: Utilisation de MapReduce

4 3ème partie : Python et pyMongo

Cette partie se concentre sur l'utilisation du langage Python et du driver PyMongo pour accéder à la base de données MongoDB et réaliser des fonctionnalités avancées.

4.1 Scripts Python

Nous avons un premier programme qui affiche la moyenne des notes des films (cf figure 17).

```

ah975865@mongo2:~/Documents/M1/S2/SGD/Projets$ python3
La moyenne des notes des films est de: 8.775
ah975865@mongo2:~/Documents/M1/S2/SGD/Projets$

```

Figure 17: Resultat du code fichier Script1.py

Le programme (cf Script1.py) récupère tous les documents de la collection CentreFilm en utilisant la méthode find() de l'objet de collection. Il parcourt chaque film dans la collection puis ajoute la note du film à la variable sommenotes en utilisant la méthode get() pour accéder à la valeur notation. Il incrémente la variable nbfilms a chaque itération puis calcul la moyenne des notes en divisant la somme des notes par le nombre total de films.

Puis nous avons un programme qui affiche la moyenne des commentaires des films.

```

ah975865@mongo2:~/Documents/M1/S2/SGD/Projets$ python3 s
Le total des notes est de : 108.0
Le nombre de commentaires avec des notes est de : 12
La moyenne des notes des commentaires est de : 9.0
ah975865@mongo2:~/Documents/M1/S2/SGD/Projets$

```

Figure 18: Resultat du code fichier Script2.py

Le programme(cf Script2.py) récupère tous les films de la collection CentreFilm, puis parcourt chaque film pour trouver les commentaires avec des notes. Ensuite, il calcule la somme des notes et le nombre total de commentaires avec des notes. Enfin, il divise la somme des notes par le nombre de commentaires pour obtenir la moyenne. Si aucun commentaire avec une note n'est trouvé, il affiche un message indiquant qu'aucun commentaire n'a été trouvé.

```

Entrez l'ID du film (10, 20, 30 ou 40) : 20
Le film "Breaking Bad" raconte l'histoire "Walter White, 50 ans, est professeur de chimie dans un lycée de nouveau Mexique. Pour reunir
de l'argent afin de subvenir aux besoins de sa familles, Walter met ses connaissances en chimie a profit pour fabriquer et vendre du cry
stal meth.".
Le prix du billet pour ce film est de 10.0 euros.
Il dure 1.58 heures.
Le genre est "Drame".
L'heure de diffusion est "18h30".
Les salles de diffusion sont :
- Salle 5
- Salle 2
- Salle 3
- Salle 12
- Salle 7
Souhaitez-vous obtenir des informations sur un autre film ? (oui/non): non
Au revoir !
ah975865@mongo2:~/Documents/M1/S2/SGD/Projets$

```

Figure 19: Resultat du code fichier Script6.py

Le programme(cf Script6.py) permet à l'utilisateur d'obtenir des informations sur un film en fonction de son ID. Il demande à l'utilisateur d'entrer l'ID d'un film parmi 10, 20, 30 ou 40. Ensuite, il recherche ce film dans la collection CentreFilm et affiche ses informations s'il est trouvé : titre, description, prix du billet, durée, genre, heure de diffusion et salles de diffusion. Si l'utilisateur souhaite obtenir des informations sur un autre film, il lui demande et répète le processus. Si l'ID entré n'est pas valide, il indique à l'utilisateur de réessayer avec un ID valide.

Attention l'id 40 n'existe pas dans la collection original, il faudra inserer le 4 eme document

present dans le fichier insertion et effectuer la requete de modification present dans le fichier misAJour pour modifier l'id du dernier document de 20 a 40. En effet le 4eme document a un id:20 cependant cet id est l'id du deuxieme film donc c'est une erreur.