Compte Rendu Technique - Livrable 2

DRAVET Timothée - FISA INFO 4A

Phase 2 : Migration MongoDB

Résumé

Ce rapport présente la migration de la base de données IMDB depuis une structure relationnelle SQLite vers une architecture NoSQL orientée documents avec MongoDB. L'objectif est d'analyser l'impact de la dénormalisation sur la performance et la complexité du développement.

1 Modèle de document structuré (Collection `movies_complete`)

Contrairement aux collections plates issues de la migration directe, la collection `movies_complete` regroupe toutes les informations liées à un film dans un document unique.

```
{
  "_id": "MID (ex: tt0111161)",
  "title": "String",
  "year": Number,
  "runtime": Number,
  "genre": "String",
  "rating": {
    "average": Number,
    "votes": Number
  },
  "directors": [
    { "id": "PID", "name": "String" }
  ],
  "writers": [
    { "id": "PID", "name": "String", "category": "String" }
  ],
  "cast": [
    {
      "id": "PID",
      "name": "String",
      "characters": ["String"],
      "ordering": Number
    }
  ],
  "titles": [
    { "title": "String", "region": "String" }
  ]
}
```

Listing 1 – Schéma JSON d'un document dénormalisé

2 Implémentation des 9 requêtes MongoDB

Voici les pipelines d'agrégation utilisés pour répondre aux besoins métiers sur la collection structurée. À retrouver dans le fichier `compare_performance.py`.

- **R1 : Filmographie d'un acteur**
  Filtre sur `cast.name` et projection du titre et de l'année.
- **R2 : Top N films par genre**
  Filtre combiné `$match` sur genre/année et `$sort` sur `rating.average`.
- **R3 : Acteurs multi-rôles**
  Utilise `$unwind: "$cast"` puis un groupement par `_id.film_id` et `_id.actor_name`.
- **R4 : Collaborations**
  Recherche des films d'un acteur, `$unwind` sur les réalisateurs et comptage.
- **R5 : Genres populaires**
  Groupement par genre avec `$avg` et `$sum`, suivi d'un `$match` (équivalent HAVING).
- **R6 : Évolution de carrière**
  Calcul de la décennie via `$divide` et `$floor` dans un bloc `$addFields`.
- **R7 : Classement par genre**
  Utilisation de `$setWindowFields` avec l'opérateur `$rank`.
- **R8 : Carrière propulsée (Breakout)**
  Pipeline complexe utilisant `$sortArray` et `$filter` pour identifier le premier succès.
- **R9 : Requête libre (Réalisateur-Acteur)**
  Comparaison de champs internes (`directors.name` présent dans `cast.name`).

```
def query_1_filmography_by_actor(db: pymongo.database.Database,
   actor_name: str, time_results: Dict) -> List[Dict]:
    """
    1. Filmographie d'un acteur : Dans quels films a jou'e un
   acteur donn'e ?
    (Recherche simple sur le tableau 'cast')
    """
    pipeline = [
        {
            "$match": {
                "cast.name": actor_name
            }
        },
        {
            "$project": {
                "_id": 0,
                "title": 1,
                "year": 1
            }
        },
        {
            "$sort": { "year": -1 }
        }
    ]
    results = list(db[TARGET_COLLECTION].aggregate(pipeline))
    return results
```

Listing 2 – Requête 1 : Filmographie d'un acteur

3 Analyse comparative : SQL vs MongoDB

3.1 Tableau des performances (Temps d'exécution)

Requête	SQL (ms)	Mongo (ms)	Gain/Perte (%)
Q1 : Filmographie	18.00	0.556	96.50%
Q2 : Top N Films	0.346	0.313	13.42%
Q3 : Multi-rôles	20.78	9.858	52.95%
Q4 : Collaborations	33.14	0.402	98.83%
Q5 : Genres populaires	1.541	0.512	62.14%
Q6 : Évolution carrière	15.81	0.363	97.50%
Q7 : Ranking par genre	0.926	0.256	72.35%
Q8 : Carrière propulsée	42.09	7.564	79.69%
Q9 : Requête libre	7.227	0.388	94.56%

TABLE 1 – Comparaison des temps de réponse (Dataset : Medium)

3.2 Complexité du code et maintenance

— **SQL :** Syntaxe déclarative concise pour les jointures, mais devient complexe avec les fonctions de fenêtre.
— **MongoDB :** Pipeline d'agrégation puissant mais verbeux. La dénormalisation simplifie les lectures mais complexifie les mises à jour.

3.3 Avantages et inconvénients

Critère	SQL (SQLite)	NoSQL (MongoDB)
Schéma	Rigide, nécessite des jointures coûteuses.	Flexible, favorise l'imbrication des données.
Performance	Excellente pour les filtres simples indexés.	Très rapide sur les documents complets (pas de JOIN).
Évolutivité	Limitée (monolithique).	Naturelle (sharding, réplication).

4 Analyse Documents Plats vs Structurés

L'analyse de la collection `movies_complete` montre que :

1. **Temps de récupération :** Passer de N requêtes (ou jointures) à 1 seule requête par `_id` réduit la latence de 74.24% en moyenne.
2. **Stockage :** La dénormalisation augmente la taille de la base d'environ 70.88% due à la duplication des noms de personnes.
3. **Complexité :** La structure imbriquée élimine le besoin de `$lookup` lors de l'affichage des détails d'un film.

5 Conclusion

La migration vers MongoDB offre un gain de performance significatif pour les accès fréquents aux détails des films, au prix d'une consommation de stockage accrue. Cette architecture est idéale pour la phase d'affichage (Phase 4).