CHIAUSA Antoine - INFO 4A FISA

Rapport Livrable 2 - MongoDB

---

1. Méthodologie de Migration

La migration de la base de données relationnelle vers MongoDB a été réalisée selon deux approches distinctes pour comparer les architectures et les performances de chacune :

1.1 Approche Flat

- **Stratégie :** Migration des tables SQL vers des collections MongoDB indépendantes (`movies` , `persons` , `ratings` , etc.).
- **Relations :** Conservation des clés étrangères (`mid` , `pid`).
- **Requetes :** Utilisation intensive du pipeline d'agrégation et de l'opérateur `$lookup` pour simuler les jointures SQL.

1.2 Approche Structurée

- **Stratégie :** Création d'une collection unique `movies_complete` .
- **Embedding :** Les données liées (genres, réalisateurs, casting, notes) sont imbriquées directement dans le document du film sous forme de tableaux ou sous-documents.
- **Objectif :** Optimiser la lecture unitaire (scénario d'affichage d'une page film).

---

2. Analyse des Requêtes Complexes

Le défi principal a été la traduction des requêtes SQL analytiques en pipelines MongoDB.

Requêtes MongoDB :

```
def query_actor_filmography(db, actor_name: str) -> List[Dict]:

"""Retourne la filmographie d'un acteur."""

start = time.time()
actor = db.persons.find_one({"primaryName": {"$regex": actor_name}})
actor_pid = actor["pid"]
principals = list(db.principals.find({"pid": actor_pid}))
films = []
for principal in principals:
    movie = db.movies.find_one({"mid": principal["mid"]})
    if movie:
        films.append({
            "primaryTitle": movie["primaryTitle"],
            "startYear": movie.get("startYear") #get pour eviter KeyError
        })
print(f"query_actor_filmography: {time.time() - start:.4f}s")
return films
```

```
def top_n_films(db, genre: str, annee_debut: int, annee_fin: int, n: int) -> List[Dict]:

"""Retourne les n films les mieux notés pour un genre et une période."""

start = time.time()
pipeline = [
    {"$match": {"startYear": {"$gte": annee_debut, "$lte": annee_fin}}},
```

```
    {"$lookup": {"from": "genres", "localField": "mid", "foreignField": "mid", "as": "genres_data"}},
    {"$unwind": "$genres_data"},
    {"$match": {"genres_data.genre": genre}},
    {"$lookup": {"from": "ratings", "localField": "mid", "foreignField": "mid", "as": "rating_data"}},
    {"$unwind": "$rating_data"},
    {"$sort": {"rating_data.averageRating": -1}},
    {"$limit": n},
    {"$project": {"_id": 0, "primaryTitle": 1, "averageRating": "$rating_data.averageRating"}}
]
result = list(db.movies.aggregate(pipeline))
print(f"top_n_films: {time.time() - start:.4f}s")
return result
```

```
def acteur_multi_roles(db, actor_name: str) -> List[Dict]:

"""Retourne les films où un acteur a joué plusieurs personnages."""
start = time.time()

pipeline = [

{"$match": {"primaryName": {"$regex": actor_name, "$options": "i"}}},

{"$lookup": {"from": "characters", "localField": "pid", "foreignField": "pid", "as":
"characters_data"}},

{"$unwind": "$characters_data"},

{"$lookup": {"from": "movies", "localField": "characters_data.mid", "foreignField": "mid", "as":
"movie_data"}},

{"$unwind": "$movie_data"},

{"$group": {"_id": "$movie_data.mid", "primaryTitle": {"$first": "$movie_data.primaryTitle"},
"nombre_de_roles": {"$sum": 1}}},

{"$match": {"nombre_de_roles": {"$gt": 1}}},

{"$sort": {"nombre_de_roles": -1}},

{"$project": {"_id": 0, "primaryTitle": 1, "nombre_de_roles": 1}} #on renvoie juste le titre et le
nombre de rôles dans celui-ci

]

result = list(db.persons.aggregate(pipeline))

print(f"acteurs_multi_roles: {time.time() - start:.4f}s")

return result
```

```
def collaborations(db, actor_name: str) -> List[Dict]:

"""Retourne les réalisateurs ayant travaillé avec un acteur spécifique."""

start = time.time()

pipeline = [

{"$match": {"primaryName": {"$regex": actor_name, "$options": "i"}}},

{"$lookup": {"from": "principals", "localField": "pid", "foreignField": "pid", "as":
"principals_data"}},

{"$unwind": "$principals_data"},
```

```
{"$group": {"_id": "$principals_data.mid", "mid": {"$first": "$principals_data.mid"}}},

{"$lookup": {"from": "directors", "localField": "mid", "foreignField": "mid", "as":
"directors_data"}},

{"$unwind": "$directors_data"},

{"$lookup": {"from": "persons", "localField": "directors_data.pid", "foreignField": "pid", "as":
"director_persons"}},

{"$unwind": "$director_persons"},

{"$group": {"_id": "$director_persons.pid", "primaryName": {"$first":
"$director_persons.primaryName"}, "nombre_de_films": {"$sum": 1}}},

{"$sort": {"nombre_de_films": -1}},

{"$project": {"_id": 0, "primaryName": 1, "nombre_de_films": 1}}

]

result = list(db.persons.aggregate(pipeline))

print(f"collaborations: {time.time() - start:.4f}s")

return result
```

```
def genres_populaires(db) -> List[Dict]:

"""Retourne les genres ayant une note moyenne > 7.0 et +50 films."""

start = time.time()

pipeline = [

{"$lookup": {"from": "movies", "localField": "mid", "foreignField": "mid", "as": "movie_data"}},

{"$unwind": "$movie_data"},

{"$lookup": {"from": "ratings", "localField": "mid", "foreignField": "mid", "as": "rating_data"}},

{"$unwind": "$rating_data"},

{"$group": {"_id": "$genre", "note_moyenne": {"$avg": "$rating_data.averageRating"},
"nombre_de_films": {"$sum": 1}}},

{"$match": {"note_moyenne": {"$gt": 7.0}, "nombre_de_films": {"$gt": 50}}},

{"$sort": {"note_moyenne": -1}},

{"$project": {"_id": 0, "genre": "$_id", "note_moyenne": 1, "nombre_de_films": 1}}

]

result = list(db.genres.aggregate(pipeline))

print(f"genres_populaires: {time.time() - start:.4f}s")

return result
```

```
def classement_par_genre(db) -> List[Dict]:

"""Retourne les 3 meilleurs films pour chaque genre."""
```

```
start = time.time()

pipeline = [

{"$lookup": {"from": "movies", "localField": "mid", "foreignField": "mid", "as": "movie_data"}},

{"$unwind": "$movie_data"},

{"$lookup": {"from": "ratings", "localField": "mid", "foreignField": "mid", "as": "rating_data"}},

{"$unwind": "$rating_data"},

{"$sort": {"genre": 1, "rating_data.averageRating": -1}},

{"$group": {"_id": "$genre", "films": {"$push": {"primaryTitle": "$movie_data.primaryTitle",
"averageRating": "$rating_data.averageRating"}}}},

{"$project": {"_id": 0, "genre": "$_id", "films": {"$slice": ["$films", 3]}}}

]

result = list(db.genres.aggregate(pipeline))

print(f"classement_par_genre: {time.time() - start:.4f}s")

return result
```

```
def carriere_propulsee(db) -> List[Dict]:

"""Retourne les personnes ayant percé grâce à un film."""

start = time.time()

pipeline = [

{"$lookup": {"from": "ratings", "localField": "mid", "foreignField": "mid", "as": "rating_data"}},

{"$unwind": "$rating_data"},

{"$lookup": {"from": "movies", "localField": "mid", "foreignField": "mid", "as": "movie_data"}},

{"$unwind": "$movie_data"},

{"$group": {

"_id": "$pid",

"films": {

"$push": {

"mid": "$mid",

"primaryTitle": "$movie_data.primaryTitle",

"startYear": "$movie_data.startYear",

"numVotes": "$rating_data.numVotes"

}

}

}},
```

```
{"$project": {

"films_faibles": {"$filter": {"input": "$films", "as": "film", "cond": {"$lt": ["$$film.numVotes",
200000]}}},

"films_forts": {"$filter": {"input": "$films", "as": "film", "cond": {"$gte": ["$$film.numVotes",
200000]}}}

}},

{"$match": {"films_faibles": {"$ne": []}, "films_forts": {"$ne": []}}},

{"$lookup": {"from": "persons", "localField": "_id", "foreignField": "pid", "as": "person_info"}},

{"$unwind": "$person_info"},

{"$project": {

"_id": 0,

"primaryName": "$person_info.primaryName",

"films_faibles": 1,

"films_forts": 1

}}

]

result = list(db.knownformovies.aggregate(pipeline))

print(f"carriere_propulsee: {time.time() - start:.4f}s")

return result
```

```
def films_par_realisateur_et_genre(db, director_name: str) -> List[Dict]:

"""Retourne tous les films d'un réalisateur par genre."""

start = time.time()

pipeline = [

{"$match": {"primaryName": {"$regex": director_name, "$options": "i"}}},

{"$lookup": {"from": "directors", "localField": "pid", "foreignField": "pid", "as":
"directors_data"}},

{"$unwind": "$directors_data"},

{"$lookup": {"from": "movies", "localField": "directors_data.mid", "foreignField": "mid", "as":
"movie_data"}},

{"$unwind": "$movie_data"},

{"$lookup": {"from": "genres", "localField": "movie_data.mid", "foreignField": "mid", "as":
"genres_data"}},

{"$unwind": "$genres_data"},

{"$lookup": {"from": "ratings", "localField": "movie_data.mid", "foreignField": "mid", "as":
"rating_data"}},
```

```
{"$unwind": "$rating_data"},

{"$sort": {"genres_data.genre": 1, "rating_data.averageRating": -1}},

{"$project": {"_id": 0, "genre": "$genres_data.genre", "primaryTitle": "$movie_data.primaryTitle",
"averageRating": "$rating_data.averageRating", "numVotes": "$rating_data.numVotes", "startYear":
"$movie_data.startYear"}}

]

result = list(db.persons.aggregate(pipeline))

print(f"films_par_realisateur_et_genre: {time.time() - start:.4f}s")

return result
```

Optimisation critique : Q7 (Carrière Propulsée)

- **Problème initial :** Au départ, mon approche partait de la collection `persons` entraînant un scan complet et des millions de `$lookup` inutiles.
-
- **Solution implémentée :** Inversion du flux de données.
  1. Départ de la table de liaison `knownformovies`.
  2. `$lookup` immédiat sur `ratings` pour filtrer.
  3. `$group` par acteur avec accumulation des films dans des tableaux.
  4. Filtrage des tableaux (films forts vs faibles).
  5. `$lookup` final (tardif) vers `persons` uniquement pour les résultats validés.
- **Résultat :** La requête est passée d'un temps d'exécution infini à ~45 secondes.

---

3. Comparaisons

Résultat de `compare_resultat.py`

Flat vs Structuré

```
================================================
TEMPS DE REQUÊTE (film complet)
================================================

1. FLAT (10000 itérations)
   Temps moyen: 0.93 ms
   Temps total: 9.34 s

2. STRUCTURÉ (10000 itérations)
   Temps moyen: 0.12 ms
   Temps total: 1.17 s

=> Structuré est 8.0x plus rapide

================================================
COMPLEXITÉ DU CODE
================================================

1. FLAT (8+ requêtes)
   movie = db.movies.find_one({'mid': id})
   genres = list(db.genres.find({'mid': id}))
   # ... 6 autres requêtes + jointures manuelles

2. STRUCTURÉ (1 requête)
   movie = db.movies_complete.find_one({'_id': id})
```

```
================================================
RÉSUMÉ
================================================

Critère                   FLAT            STRUCTURÉ
------------------------------------------------------------
Temps moyen/requête       0.93 ms        0.12 ms
Temps total (100000 req)  9.34 s         1.17 s
Taille stockage           935.90 MB       467.28 MB
```

SQL vs MongoDB (Flat)

```
================================================
COMPARATIF SQL vs MONGODB
================================================

Requête                   SQL (ms)     MongoDB (ms) Gain
------------------------------------------------------------
Q1                        207.26       264.21       0.8x
Q2                        65.90        431.93       0.2x
Q3                        202.09       258.38       0.8x
Q4                        530.24       300.13       1.8x
Q5                        914.78       19093.97     0.0x
Q6                        1704.83      22098.12     0.1x
Q7                        8524.95      45301.05     0.2x
Q8                        1476.81      259.45       5.7x
------------------------------------------------------------
TOTAL                     13626.86     88007.24     0.2x
```

C'est tout à fait normal que nous ayons une moins bonne performance avec MongoDB ici car nous utilisons la même approche qu'avec SQL. Toutes nos tables sont des collections différentes et chaque ligne de notre table SQL est un document dans la collection. Pour que MongoDB soit beaucoup plus rapide que SQL, il faudrait modifier les requêtes et interroger notre collection unique `movies_complete` plutôt que d'interroger une collection, puis de faire un lookup dessus, et encore un autre, etc...
C'est notamment visible sur `carriere_propulsee`, sur `classement_par_genre` et sur `films_par_realisateur_et_genre`.

---

4. Conclusion

1. **Lecture (Web App) :** MongoDB Structuré est **8x plus rapide** que l'approche relationnelle. En revanche l'approche "flat" peut être bien plus lente (20 fois +) que l'approche SQLite avec index.
2. **Stockage :** L'approche structurée permet un gain de **50%** d'espace disque.