

Base de Données Avancées - Livrable 2

Phase 2 : Migration et Optimisation NoSQL (MongoDB)

Exertier Hugo

Décembre 2025

Table des matières

1	Introduction	2
2	T2.2 : Migration "Flat" (Collections Plates)	2
2.1	Stratégie d'indexation initiale	2
3	T2.3 : Requêtes équivalentes (SQL vs MongoDB)	2
3.1	Comparatif de complexité	2
3.2	Tableau comparatif des performances	2
4	T2.4 : Documents Structurés (Optimisation)	2
4.1	Modèle de document (Schéma JSON)	3
4.2	Analyse : Documents Plats vs Structurés	3
5	Conclusion	3
A	Annexe : Les 9 Requêtes MongoDB (Code)	4

1 Introduction

Ce livrable présente la migration de la base de données relationnelle (SQLite) vers une base NoSQL orientée documents (MongoDB). L'objectif est de comparer les paradigmes, d'analyser les coûts de migration et de démontrer les gains de performance offerts par la dénormalisation des données pour une application web de type "CinéExplorer".

2 T2.2 : Migration "Flat" (Collections Plates)

Dans un premier temps, nous avons migré les données "telles quelles" (1 table SQL = 1 collection MongoDB) pour établir une base de comparaison.

2.1 Stratégie d'indexation initiale

Contrairement au SQL où les clés étrangères sont gérées implicitement, MongoDB nécessite des index explicites pour être performant lors des jointures (\$lookup). Pour éviter des temps d'exécution rédhibitoires (Full Collection Scan), nous avons créé des index sur les champs `movie_id` et `person_id` dès l'importation des données.

3 T2.3 : Requêtes équivalentes (SQL vs MongoDB)

Nous avons traduit les 9 requêtes SQL en Pipelines d'Agrégation MongoDB (Voir Annexe A pour le code complet).

3.1 Comparatif de complexité

L'absence de jointures natives simples en NoSQL rend certaines requêtes beaucoup plus verbeuses.

- **SQL** : Une jointure s'écrit en une ligne (`JOIN table ON ...`).
- **MongoDB** : Une jointure nécessite un stage `$lookup` suivi souvent d'un `$unwind`.

3.2 Tableau comparatif des performances

Les mesures comparent SQLite (avec index, phase 1) et MongoDB (avec la structure optimisée T2.4).

Requête	SQL (ms)	Mongo (ms)	Observation
Q1 - Filmographie	1465.8	271.6	SQL plus rapide (Index plus direct)
Q2 - Top N films	25.3	56.4	Comparable (Filtres simples)
Q3 - Multi-rôles	1478.3	632.1	Gain x2.3 (Agrégation locale)
Q4 - Collaborations	1089.9	240.0	Gain x4.5 (Plus de jointure complexe)
Q5 - Genres Pop.	245.2	99.5	Gain x2.5 (Tableaux pré-construits)
Q6 - Carrière	3086.6	167.7	Explosion perf. (x18)
Q7 - Classement	640.7	3095.1	SQL imbattable sur Window Func.
Q8 - Percée	3261.1	34.8	Explosion perf. (x93)
Q9 - Longévité	1675.2	7.7	Imbattable (x217)

TABLE 1 – Benchmark SQL vs NoSQL Structuré (Moyenne)

Analyse : MongoDB surclasse SQL sur les requêtes analytiques lourdes (Q6, Q8, Q9) car toutes les données sont regroupées dans un seul document, évitant les jointures coûteuses.

4 T2.4 : Documents Structurés (Optimisation)

Pour exploiter la puissance du NoSQL, nous avons dénormalisé les données dans une collection unique `movies_complete`.

4.1 Modèle de document (Schéma JSON)

Chaque film contient désormais ses acteurs, notes et équipes techniques sous forme embarquée.

```
{
  "_id": "tt0111161",
  "title": "The Shawshank Redemption", "year": 1994,
  "genres": ["Drama", "Crime"],
  "rating": { "average": 9.3, "votes": 2500000 },
  "directors": [ { "person_id": "nm0001104", "name": "Frank Darabont" } ],
  "cast": [
    { "person_id": "nm0000209", "name": "Tim Robbins",
      "characters": ["Andy Dufresne"], "ordering": 1 }
  ]
}
```

Listing 1 – Structure d'un document movies_complete

4.2 Analyse : Documents Plats vs Structurés

La comparaison de performance pour récupérer la fiche complète d'un film valide la stratégie.

Métriques	Approche Flat (Type SQL)	Approche Structurée (NoSQL)
Méthode	10 requêtes / lookups	1 requête (find_one)
Temps moyen	9.09 ms	0.51 ms
Gain	-	x 17.9
Stockage	111.01 MB	38.68 MB (-65%)

TABLE 2 – Impact de la dénormalisation sur l'accès unitaire et le stockage

Observation Stockage : Contrairement à la théorie usuelle où la dénormalisation augmente la taille (duplication), ici le modèle structuré est **65% plus léger**. Cela s'explique par la suppression de la redondance des clés étrangères (l'ID du film n'est plus répété 360 000 fois dans la table casting, mais stocké une seule fois par document).

5 Conclusion

La migration démontre la puissance du modèle NoSQL pour la lecture rapide (Gain x17.9) et même pour l'optimisation du stockage (-65%). L'approche hybride est validée : MongoDB servira les pages de détails instantanément, tandis que SQL reste pertinent pour les recherches multicritères complexes.

A Annexe : Les 9 Requêtes MongoDB (Code)

Voici les pipelines d'agrégation implémentés pour reproduire les requêtes SQL sur le modèle structuré.

```
[
  {"$match": {"cast.name": {"$regex": actor_name, "$options": "i"}}},
  {"$project": { "title": 1, "year": 1, "rating": "$rating.average" }}
]
```

Listing 2 – Q1: Filmographie (Pipeline)

```
[
  {"$match": {"genres": genre}},
  {"$sort": {"rating.average": -1}},
  {"$limit": limit}
]
```

Listing 3 – Q2: Top N Films (Pipeline)

```
[
  {"$unwind": "$cast"},
  {"$group": { "_id": {"mid": "$_id", "pid": "$cast.person_id"}, "count": {"$sum": 1} }},
  {"$match": {"count": {"$gt": 1}}}
]
```

Listing 4 – Q3: Acteurs Multi-rôles (Pipeline)

```
[
  {"$match": {"cast.name": {"$regex": actor_name, "$options": "i"}}},
  {"$unwind": "$directors"},
  {"$group": { "_id": "$directors.name", "count": {"$sum": 1} }},
  {"$sort": {"count": -1}}
]
```

Listing 5 – Q4: Collaborations (Pipeline)

```
[
  {"$unwind": "$genres"},
  {"$group": { "_id": "$genres", "avg": {"$avg": "$rating.average"}, "count": {"$sum": 1} }},
  {"$match": { "avg": {"$gt": 7.0}, "count": {"$gt": 50} }},
  {"$sort": {"avg": -1}}
]
```

Listing 6 – Q5: Genres Populaires (Pipeline)

```
[
  {"$match": {"cast.name": {"$regex": actor_name, "$options": "i"}}},
  {"$project": { "decade": { "$multiply": [{ "$floor": {"$divide": [{"$year", 10} ]}], 10} } }},
  {"$group": { "_id": "$decade", "count": {"$sum": 1}, "avg": {"$avg": "$rating.average" } } }
]
```

Listing 7 – Q6: Carrière par décennie (Pipeline)

```
[
  {"$match": {"rating.votes": {"$gt": 5000}}},
  {"$unwind": "$genres"},
  {"$setWindowFields": {
    "partitionBy": "$genres", "sortBy": {"rating.average": -1},
    "output": { "rank": {"$rank": {}} }
  }},
  {"$match": {"rank": {"$lte": 3}}}
]
```

Listing 8 – Q7: Classement par Genre (Window Function)

```
[
  {"$match": {"rating.votes": {"$gt": 200000}}},
  {"$unwind": "$cast"},
  {"$sort": {"year": 1}},
  {"$group": { "_id": "$cast.person_id", "first_hit": {"$first": "$title" } } }
]
```

]

Listing 9 – Q8: Carrière Propulsée (Pipeline)

```
[
  {"$match": {
    "runtime": {"$gt": 120},
    "rating.average": {"$gt": 8.0},
    "titles": {"$elemMatch": {"region": "FR"}}
  }}
]
```

Listing 10 – Q9: Requête Complexe (Pipeline)