Livrable 2 - Migration MongoDB

Boungo Battiste FISA 4A

Projet: Bases de Données Distribuées et Avancées
Dataset: IMDB-Medium (290k films, 630k personnes)

---

1. Introduction et Objectifs

Ce livrable compare les performances entre SQLite (relationnel) et MongoDB (NoSQL) sur le dataset IMDB-Medium, en explorant deux approches MongoDB : collections plates (normalisées) et documents structurés (dénormalisés).

Objectifs:
- Migrer SQLite → MongoDB (T2.1-T2.2)
- Implémenter les 9 requêtes équivalentes (T2.3)
- Créer une collection dénormalisée `movies_complete` (T2.4)
- Comparer performances et complexité

---

2. Architecture des Données

2.1 Schéma Relationnel (SQLite)

6 tables normalisées (3NF) avec contraintes FK:

> movies, persons, genres, ratings, directors, writers

2.2 Collections Plates MongoDB

Réplication 1:1 du schéma relationnel - 6 collections séparées nécessitant `$lookup` pour les jointures.

2.3 Documents Structurés (T2.4)

Collection `movies_complete` avec données imbriquées:

```
{
  _id: "tt0111161",
  primaryTitle: "The Shawshank Redemption",
  startYear: 1994,
  genres: ["Drama", "Crime"],
  rating: {averageRating: 9.3, numVotes: 2500000},
  directors: [{pid: "nm0001104", name: "Frank Darabont"}],
  writers: [{pid: "nm0000175", name: "Stephen King"}]
}
```

Création: Pipeline d'agrégation avec 4 `$lookup` + enrichissement (voir migrate_structured.py)

Avantages: 1 requête au lieu de 8, pas de `$lookup`, code simplifié
Inconvénient: Redondance des données, mises à jour plus complexes

---

3. Migration et Indexation

3.1 Migration (T2.2)

Script: migrate_flat.py**Résultat:** 3,188,991 documents migrés en ~5 minutes

Collection	Documents
movies	289,061
persons	630,986
genres	696,594
ratings	289,061
directors	648,578
writers	634,711

3.2 Indexation (T2.3)

Script: create_indexes.py

MongoDB: 18 index (12 simples + 6 composites) créés en 31s
Augmentation stockage: +20-25%

Index critiques:

- `movies.startYear`, `persons.primaryName`

- `genres.mid` , `ratings.mid`
- Composites: `{genre: 1, mid: 1}` , `{averageRating: -1, numVotes: -1}`

---

4. Les 9 Requêtes MongoDB (T2.3)

Script: queries_mongo.py

#	Requête	Technique	Performance	Ligne
Q1	Filmographie personne	3 `$lookup` + `$regex`	>30 min	45-69
Q2	Top N par genre	2 `$lookup`	>30 min	72-94
Q3	Acteurs multi-rôles	Non implémentée (données manquantes)	-	-
Q4	Collaborations	4 `$lookup`	>30 min	128-160
Q5	Genres populaires	1 `$lookup` + `$group`	>30 min	163-181
Q6	Évolution carrière	6 `$lookup` (`$facet`)	>30 min	184-224
Q7	Top 3 par genre	2 `$lookup` + `$push`	>30 min	227-249
Q8	Carrières propulsées	2 `$lookup` + agrégations	>30 min	252-281
Q9	Dir-Writer combo	2 `$lookup` (sans `$unwind`)	~quelques secondes	284-304

Observation: Toutes les requêtes avec `$lookup` sont inadaptées pour ce volume de données, sauf Q9 qui évite le `$unwind` massif.

---

5. Comparaison des Performances

5.1 Benchmarks SQLite

Script: quick_benchmark.py

Requête	Temps
Q2 - Top 10 Drama	663 ms

Requête	Temps
Q5 - Genres populaires	4,276 ms
Q7 - Top 3 par genre	13,109 ms

Moyenne: ~4,500 ms - Les JOIN sont rapides avec index B-Tree

5.2 Benchmarks MongoDB - Avec $lookup

Script: quick_benchmark_mongo.py

Résultat: Toutes les requêtes avec `$lookup` dépassent 30 minutes → TIMEOUT

Cause: `$lookup` scanne intégralement les collections cibles (O(n²))

5.3 Benchmarks MongoDB - Sans $lookup

Script: simple_benchmark_mongo.py

Requête	Temps
Top 20 films par votes	5 ms
Compter films par année	461 ms
Distribution genres	1,268 ms

Moyenne: 547 ms - MongoDB excellent pour agrégations simples (sans jointures)

5.4 Documents Structurés (T2.4)

Script: compare_queries.py

Comparaison sur 20 films aléatoires:

Méthode	Temps moyen	Requêtes/film	Gain
Collections plates (N requêtes)	90.91 ms	8.0	-
Documents structurés (1 requête)	**0.65 ms**	1	**140x**

Stockage:
- Collections plates: 269.5 MB
- Documents structurés: 140.8 MB
- **Économie: 47.8%** (élimination des foreign keys redondants)

Complexité du code:

```
# Collections plates: ~15 lignes
movie = db.movies.find_one({"_id": mid})
genres = list(db.genres.find({"mid": mid}))
rating = db.ratings.find_one({"mid": mid})
# ... + directors + writers avec boucles

# Documents structurés: 1 ligne
movie = db.movies_complete.find_one({"_id": mid})
```

Gain: 15x moins de code

---

6. Tableau Comparatif Final

Critère	SQLite	MongoDB (plates)	MongoDB (structurées)
Requêtes avec JOIN/lookup	663-13,109 ms	>30 min	-
Requêtes simples	115-4,276 ms	5-1,268 ms	-
Lecture film complet	~100 ms	90.91 ms	**0.65 ms**
Stockage (sans index)	180 MB	220 MB	**141 MB**
Nombre d'index	12	18	0

Ratios de performance:
- MongoDB plates avec `$lookup` : **100-1000x plus lent** que SQLite
- MongoDB plates sans `$lookup` : **8x plus rapide** que SQLite
- MongoDB structurées: **140x plus rapide que plates**, **154x plus rapide que SQLite**

---

7. Analyse et Conclusion

7.1 Pourquoi MongoDB est 100x plus lent avec $lookup?

Architecture:
- SQLite: Index B-Tree avec pointeurs directs (O(log n))
- MongoDB `$lookup` : Scan complet de la collection cible ($O(n^2)$)

Exemple: Pour Q2 avec 289k films × 696k genres → 201 milliards de comparaisons potentielles

7.2 Recommandations

Utiliser SQLite/PostgreSQL quand:
- Schéma normalisé avec nombreuses jointures
- Intégrité référentielle critique
- Transactions ACID requises

Utiliser MongoDB quand:
- Documents auto-suffisants (embedded)
- Schéma flexible requis
- Scalabilité horizontale nécessaire
- Agrégations complexes sur UNE collection

7.3 Conclusion Générale

Leçon principale: L'architecture des données dicte le choix de la base de données.

Avec un schéma normalisé (collections/tables séparées):
- SQLite est **100-1000x plus performant** que MongoDB pour les jointures
- MongoDB `$lookup` n'est qu'une solution de secours, pas une solution principale

Avec un schéma dénormalisé (embedded documents):
- MongoDB devient **140x plus rapide** que les collections plates
- **154x plus rapide** que SQLite pour la récupération de documents complets
- **47.8% d'économie de stockage** (élimination redondance FK)

MongoDB n'est PAS un remplacement drop-in de SQL - il nécessite une restructuration complète du schéma et une approche différente de la modélisation des données.

---

8. Fichiers du Livrable

```
Livrable_2_MongoDB/
├── test_connection.py          # T2.1: Test connexion MongoDB
├── migrate_flat.py             # T2.2: Migration SQLite → MongoDB
├── queries_mongo.py            # T2.3: 9 requêtes équivalentes
├── create_indexes.py           # T2.3: Création 18 index
├── migrate_structured.py       # T2.4: Collection movies_complete
├── compare_queries.py          # T2.4: Comparaison 1 vs N requêtes
├── simple_benchmark_mongo.py   # Benchmark sans $lookup
```

```
├── quick_benchmark_mongo.py     # Benchmark avec $lookup
└── RAPPORT.md                   # Ce rapport
```

Exécution:

```
python test_connection.py          # Vérifier connexion
python migrate_flat.py             # Migrer vers MongoDB
python create_indexes.py           # Créer les index
python migrate_structured.py       # Créer movies_complete (20 min)
python compare_queries.py          # Comparer performances
```