Projet : Plateforme Web de Découverte de Films

SQL & NoSQL avec Django

Module : 4A-BDA – Bases de Données Avancées

Année universitaire :	2025-2026
Encadrants :	AL-KHARAZ M. HADDOU BEN DERBAL H.
Type :	Projet pratique
Prérequis :	Python
Équipes :	Groupes de 1 étudiant

Technologies utilisées

SQLite • MongoDB • Django • Python

Aix-Marseille Université – Polytech Marseille

Département Informatique

Table des matières

1 Introduction et vue d'ensemble

1.1 Contexte du projet

Scénario professionnel

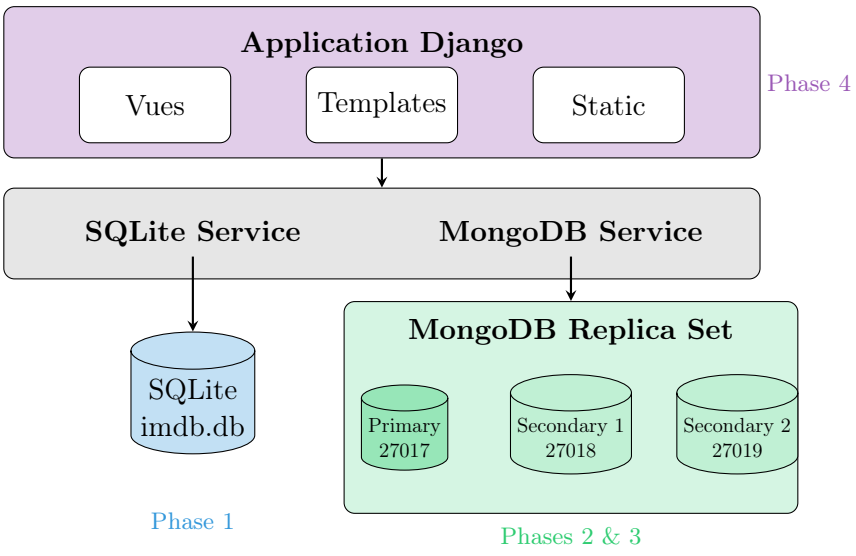
Vous êtes recrutés par **CinéExplorer**, une startup spécialisée dans l'information cinématographique. L'entreprise dispose d'une base de données IMDB qu'elle souhaite exploiter pour créer une plateforme web moderne.
Votre mission : Concevoir et développer une application web complète permettant aux utilisateurs d'explorer des films, avec une architecture évolutive et tolérante aux pannes.

Le projet vous amènera à :

1. **Explorer et comprendre** les données IMDB (analyse exploratoire)
2. **Maîtriser** une base relationnelle SQLite avec des requêtes SQL avancées
3. **Migrer** vers MongoDB pour bénéficier de la flexibilité NoSQL
4. **Distribuer** les données avec un Replica Set tolérant aux pannes
5. **Développer** une interface web professionnelle avec Django

1.2 Architecture globale du système

Le schéma ci-dessous présente l'architecture cible que vous allez construire progressivement :



1.3 Objectifs pédagogiques par phase

Phase	Thème	Compétences acquises
1	SQLite	Modélisation relationnelle, SQL avancé (jointures, agrégations, CTE, window functions), indexation
2	MongoDB	Modélisation documents, migration, pipeline d'agrégation, comparaison SQL/NoSQL
3	Distribution	Replica Set, tolérance aux pannes, élection de leader, haute disponibilité
4	Django	Architecture MVC, intégration multi-bases, design responsive, UX

2 Données IMDB

2.1 Description du dataset

Les données proviennent d'IMDB et sont fournies sous forme de fichiers CSV :

Fichier	Description
`movies.csv`	Films (identifiant, titre, année, durée)
`persons.csv`	Personnes (acteurs, réalisateurs, scénaristes)
`characters.csv`	Personnages joués par les acteurs
`directors.csv`	Relations film ↔ réalisateur
`genres.csv`	Genres associés aux films
`principals.csv`	Acteurs/réalisateurs principaux d'un film
`ratings.csv`	Notes moyennes et nombre de votes
`titles.csv`	Titres dans différentes langues/régions
`writers.csv`	Scénaristes des films

2.2 Tailles disponibles

Dataset	Films	Personnes	Usage recommandé
`imdb-tiny.zip`	∼100	∼500	Tests rapides
`imdb-small.zip`	∼10 000	∼50 000	**Développement (recommandé)**
`imdb-medium.zip`	∼100 000	∼500 000	Tests de performance

Conseil

Commencez avec `imdb-small.zip`. Passez à `imdb-medium.zip` uniquement si tout fonctionne et que vos ressources le permettent.

3 Structure du projet Django

Dès le début du projet, organisez votre code selon cette structure :

```
cineexplorer/
|-- manage.py
|-- requirements.txt
|-- README.md
|-- .gitignore
|
|-- config/                     # Configuration Django
|   |-- __init__.py
|   |-- settings.py
|   |-- urls.py
|   +-- wsgi.py
|
|-- data/                       # Donnees
|   |-- csv/                    # Fichiers CSV IMDB
|   |   |-- movies.csv
|   |   |-- persons.csv
|   |   +-- ...
|   |-- imdb.db                 # Base SQLite generee
|   |-- exploration.ipynb       # Notebook d'exploration
```

```
|   +-- mongo/                    # Donnees MongoDB (a ajouter au .gitignore)
|       |-- standalone/           # Phase 2 : instance simple
|       |-- db-1/                 # Phase 3 : Replica Set node 1
|       |-- db-2/                 # Phase 3 : Replica Set node 2
|       +-- db-3/                 # Phase 3 : Replica Set node 3
|
|-- scripts/                      # Scripts par phase
|   |-- phase1_sqlite/
|   |   |-- create_schema.py
|   |   |-- import_data.py
|   |   |-- queries.py
|   |   +-- benchmark.py
|   |-- phase2_mongodb/
|   |   |-- migrate_flat.py
|   |   |-- migrate_structured.py
|   |   |-- queries_mongo.py
|   |   +-- compare_performance.py
|   +-- phase3_replica/
|       |-- setup_replica.sh
|       +-- test_failover.py
|
|-- movies/                       # Application Django
|   |-- __init__.py
|   |-- models.py                 # Modeles SQLite
|   |-- views.py
|   |-- urls.py
|   |-- services/
|   |   |-- __init__.py
|   |   |-- sqlite_service.py     # Requetes SQLite
|   |   +-- mongo_service.py      # Requetes MongoDB
|   +-- templates/
|       +-- movies/
|           |-- base.html
|           |-- home.html
|           |-- list.html
|           |-- detail.html
|           |-- search.html
|           +-- stats.html
|
|-- static/
|   |-- css/
|   |-- js/
|   +-- img/
|
+-- reports/                      # Rapports PDF
    |-- livrable1/
    |-- livrable2/
    |-- livrable3/
    +-- final/
```

Conseil

Créez cette structure dès la Phase 1. Utilisez Git dès le début avec des commits réguliers.

4 Phase 1 : Exploration et base SQLite

4.1 Objectifs

— Comprendre la structure et la qualité des données IMDB
— Concevoir un schéma relationnel normalisé
— Maîtriser des requêtes SQL de complexité variée
— Optimiser les performances avec des index

4.2 T1.0 : Exploration des données

Avant de créer le schéma, explorez les données pour les comprendre.
Créez un notebook Jupyter `data/exploration.ipynb` contenant :

1. **Chargement des CSV** avec pandas
2. **Statistiques descriptives** pour chaque fichier :
   — Nombre de lignes et colonnes
   — Types de données
   — Valeurs manquantes (NaN, chaînes vides)
   — Valeurs uniques par colonne
3. **Analyses exploratoires** :
   — Distribution des films par année (histogramme)
   — Top 10 des genres les plus fréquents
   — Distribution des notes (ratings)
   — Nombre moyen d'acteurs par film
4. **Relations entre tables** :
   — Vérifier les clés étrangères (tous les movie_id existent-ils ?)
   — Identifier les données orphelines

Livrable : Le notebook avec les visualisations et vos observations.

4.3 T1.1 : Conception du schéma relationnel

1. Concevoir un schéma relationnel normalisé (3NF)
2. Définir les clés primaires et étrangères
3. Créer un diagramme Entité-Relation (ER)
4. Implémenter le script `create_schema.py`

4.4 T1.2 : Import des données

Implémenter `import_data.py` :

1. Respecter l'ordre d'insertion (tables parentes avant enfants)
2. Utiliser des transactions pour la performance
3. Gérer les erreurs et données invalides
4. Afficher les statistiques d'import

4.5 T1.3 : Requêtes SQL

Implémenter les requêtes suivantes dans `queries.py`. Chaque requête doit être une fonction Python documentée.

#	Description
1	**Filmographie d'un acteur** : Dans quels films a joué un acteur donné ? (paramètre : nom)
2	**Top N films** : Les N meilleurs films d'un genre sur une période selon la note moyenne (paramètres : genre, année_début, année_fin, N)
3	**Acteurs multi-rôles** : Acteurs ayant joué plusieurs personnages dans un même film, triés par nombre de rôles
4	**Collaborations** : Réalisateurs ayant travaillé avec un acteur spécifique, avec le nombre de films ensemble (utiliser une sous-requête)
5	**Genres populaires** : Genres ayant une note moyenne > 7.0 et plus de 50 films, triés par note (utiliser GROUP BY + HAVING)
6	**Évolution de carrière** : Pour un acteur donné, nombre de films par décennie avec note moyenne (utiliser CTE - WITH)
7	**Classement par genre** : Pour chaque genre, les 3 meilleurs films avec leur rang (utiliser RANK() ou ROW_NUMBER())
8	**Carrière propulsée** : Personnes ayant percé grâce à un film (avant : films $<$ 200k votes, après : films $>$ 200k votes)
9	**Requête libre** : Inventez une requête utilisant au moins 3 jointures et répondant à une question pertinente

Format attendu pour chaque requête :

```
def query_actor_filmography(conn, actor_name: str) -> list:
    """
    Retourne la filmographie d'un acteur.

    Args:
        conn: Connexion SQLite
        actor_name: Nom de l'acteur (ex: "Tom Hanks")

    Returns:
        Liste de tuples (titre, année, personnage, note)

    SQL utilisé:
        SELECT ... FROM ... JOIN ... WHERE ...
    """
    sql = """
        SELECT m.title, m.year, c.character, r.average_rating
        FROM movies m
        JOIN principals p ON m.movie_id = p.movie_id
        JOIN persons pe ON p.person_id = pe.person_id
        LEFT JOIN characters c ON ...
        LEFT JOIN ratings r ON ...
```

```
    WHERE pe.name LIKE ?
    ORDER BY m.year DESC
"""
return conn.execute(sql, (f'%{actor_name}%',)).fetchall()
```

4.6 T1.4 : Indexation et benchmark

1. Mesurer le temps d'exécution de chaque requête (sans index)
2. Analyser les plans d'exécution avec `EXPLAIN QUERY PLAN`
3. Créer des index pertinents
4. Mesurer à nouveau et calculer le gain
5. Comparer la taille de la base avant/après

Tableau de benchmark attendu :

Requête	Sans index (ms)	Avec index (ms)	Gain (%)
Q1 - Filmographie Q2 - Top N films ...			

5 Phase 2 : Migration MongoDB

5.1 Objectifs

— Comprendre les différences SQL vs NoSQL
— Maîtriser la migration de données
— Utiliser le pipeline d'agrégation MongoDB
— Comparer les performances des deux approches

5.2 T2.1 : Installation et configuration

1. Installer MongoDB Community Edition (5.x ou 6.x)
2. Lancer le serveur : `mongod -dbpath ./data/mongo/standalone`
3. Installer PyMongo : `pip install pymongo`
4. Tester la connexion depuis Python

5.3 T2.2 : Migration des collections plates

Créer `migrate_flat.py` :

1. Créer une collection MongoDB pour chaque table SQLite
2. Extraire les données de SQLite (pas des CSV !)
3. Insérer dans MongoDB avec `insert_many()`
4. Vérifier les comptages

5.4 T2.3 : Requêtes MongoDB équivalentes

Reproduire les 9 requêtes SQL en MongoDB dans `queries_mongo.py` :

— Utiliser `find()`, `aggregate()`, `$lookup`, `$group`, `$match`
— Chronométrer chaque requête
— Comparer avec les temps SQLite

5.5 T2.4 : Documents structurés

Créer une collection `movies_complete` avec des documents dénormalisés :

```
{
    "_id": "tt0111161",
    "title": "The Shawshank Redemption",
    "year": 1994,
    "runtime": 142,
    "genres": ["Drama"],
    "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
        },
        {
            "person_id": "nm0000151",
            "name": "Morgan Freeman",
            "characters": ["Red"],
            "ordering": 2
        }
    ],
    "writers": [
        {"person_id": "nm0175840", "name": "Stephen King", "category": "novel"},
        {"person_id": "nm0001104", "name": "Frank Darabont", "category": "screenplay"}
    ],
    "titles": [
        {"region": "US", "title": "The Shawshank Redemption"},
        {"region": "FR", "title": "Les Évadés"}
    ]
}
```

Méthode : Utiliser le pipeline d'agrégation avec `$lookup` multiples.
Comparaison à effectuer :

— Temps pour récupérer un film complet (1 requête vs N requêtes)

— Taille de stockage (collections plates vs structurées)

— Complexité du code

6 Phase 3 : Distribution et Replica Set

6.1 Objectifs

— Comprendre la réplication et la haute disponibilité

— Configurer un Replica Set MongoDB
— Tester la tolérance aux pannes
— Préparer l'intégration Django

6.2 T3.1 : Configuration du Replica Set

1. Créer 3 répertoires dans `data/mongo/` : `db-1`, `db-2`, `db-3`
2. Lancer 3 instances MongoDB (depuis la racine du projet) :

```
# Terminal 1 - Primary
mongod --replSet rs0 --port 27017 --dbpath ./data/mongo/db-1 --bind_ip localhost

# Terminal 2 - Secondary 1
mongod --replSet rs0 --port 27018 --dbpath ./data/mongo/db-2 --bind_ip localhost

# Terminal 3 - Secondary 2
mongod --replSet rs0 --port 27019 --dbpath ./data/mongo/db-3 --bind_ip localhost
```

3. Initialiser le Replica Set :

```
// Dans mongosh connecté au port 27017
rs.initiate({
    _id: "rs0",
    members: [
        { _id: 0, host: "localhost:27017" },
        { _id: 1, host: "localhost:27018" },
        { _id: 2, host: "localhost:27019" }
    ]
})
```

4. Vérifier avec `rs.status()`
5. Importer les données sur le primaire

6.3 T3.2 : Tests de tolérance aux pannes

Effectuer et documenter les tests suivants :

Test	**Actions et observations**
1. État initial	Capturer `rs.status()`, identifier Primary/Secondary
2. Écriture	Insérer des documents, vérifier la réplication
3. Panne Primary	Arrêter le Primary (Ctrl+C), observer l'élection
4. Nouveau Primary	Mesurer le temps d'élection, vérifier les données
5. Lecture	Confirmer que les données sont accessibles
6. Reconnexion	Relancer le nœud arrêté, observer la resync
7. Double panne	Que se passe-t-il si 2 nœuds tombent ?

6.4 T3.3 : Préparation Django

1. Créer le projet Django : `django-admin startproject config .`
2. Créer l'application : `python manage.py startapp movies`
3. Configurer `settings.py` (SQLite + connexion MongoDB)

4. Créer les services de base :
   — `sqlite_service.py` : fonctions d'accès SQLite
   — `mongo_service.py` : fonctions d'accès MongoDB
5. Créer une vue de test affichant des statistiques

7 Phase 4 : Interface web Django

7.1 Objectifs

— Créer une interface utilisateur professionnelle
— Intégrer intelligemment les deux bases de données
— Implémenter une UX intuitive et responsive

7.2 T4.1 : Pages à implémenter

1. **Page d'accueil** (`/`)
   — Statistiques : nombre de films, acteurs, réalisateurs
   — Top 10 des films les mieux notés
   — Formulaire de recherche rapide
   — Films récemment ajoutés ou aléatoires
2. **Liste des films** (`/movies/`)
   — Pagination (20 films par page)
   — Filtres : genre, année (range), note minimale
   — Tri : titre, année, note (ASC/DESC)
   — Affichage en grille ou liste
3. **Détail d'un film** (`/movies/<id>/`)
   — Toutes les informations (depuis MongoDB)
   — Casting complet avec personnages
   — Réalisateurs et scénaristes
   — Titres alternatifs par région
   — Films similaires (même genre/réalisateur)
4. **Recherche** (`/search/?q=...`)
   — Recherche par titre de film
   — Recherche par nom de personne
   — Résultats groupés par type
5. **Statistiques** (`/stats/`)
   — Films par genre (bar chart)
   — Films par décennie (line chart)
   — Distribution des notes (histogram)
   — Top 10 acteurs prolifiques

7.3 T4.2 : Stratégie d'intégration multi-bases

Fonctionnalité	**Base utilisée**	**Justification**
Liste des films + filtres	SQLite	Requêtes relationnelles efficaces
Détail complet d'un film	MongoDB	Document pré-agrégé, 1 requête
Statistiques agrégées	SQLite ou MongoDB	Selon la complexité
Recherche textuelle	SQLite (LIKE)	Simple et suffisant

7.4 T4.3 : Design et UX

- Utiliser Bootstrap 5 ou Tailwind CSS
- Design responsive (mobile-first)
- Navigation claire (navbar, breadcrumbs)
- Feedback utilisateur (loading, messages)
- Graphiques avec Chart.js

8 Livrables et évaluation

8.1 Livrable 1 : Exploration et SQLite – 25%

À rendre :

- **Code :** Notebook d'exploration + scripts Phase 1
- **Rapport PDF (4-5 pages)** :
  - Résumé de l'exploration (observations clés, graphiques)
  - Diagramme ER du schéma relationnel
  - Les 9 requêtes SQL avec explications
  - Tableau de benchmark (avec/sans index)
  - Index créés et justifications

8.2 Livrable 2 : MongoDB – 25%

À rendre :

- **Code :** Scripts de migration et requêtes MongoDB
- **Rapport PDF (4-5 pages)** :
  - Modèle de document structuré (schéma JSON)
  - Les 9 requêtes MongoDB équivalentes
  - Tableau comparatif SQL vs MongoDB :
    - Temps d'exécution
    - Complexité du code
    - Avantages/inconvénients
  - Analyse documents plats vs structurés

8.3 Livrable 3 : Replica Set – 25%

À rendre :

- **Code :** Scripts de configuration et tests
- **Rapport PDF (3-4 pages)** :
  - Schéma de l'architecture (3 nœuds)
  - Procédure de configuration
  - Résultats des tests de panne (captures d'écran)
  - Temps de failover mesuré
  - Analyse : comportement avec 1 nœud, 2 nœuds down

8.4 Livrable 4 : Projet final – 25%

À rendre :

- **Repository Git** (GitHub/GitLab) avec :
  - Structure organisée (voir Section 3)
  - README.md complet
  - requirements.txt
  - .gitignore
  - Historique de commits propre
- **Application Django** fonctionnelle avec les 5 pages
- **Rapport final PDF (8-10 pages)** :
  - Architecture globale (schéma)
  - Choix technologiques justifiés
  - Description des fonctionnalités
  - Stratégie multi-bases
  - Benchmarks de performance
  - Difficultés et solutions

9 Technologies et ressources

9.1 Stack technologique

Obligatoire :

- Python 3.10+
- SQLite 3 (inclus avec Python)
- MongoDB 5.x ou 6.x (Community Edition)
- Django 4.x ou 5.x
- PyMongo

Recommandé :

- pandas, matplotlib (exploration)
- Jupyter Notebook
- Bootstrap 5
- Chart.js
- Git

9.2 Documentation

- Django : `https://docs.djangoproject.com/`
- PyMongo : `https://pymongo.readthedocs.io/`
- MongoDB : `https://docs.mongodb.com/`
- Bootstrap : `https://getbootstrap.com/docs/`
- Chart.js : `https://www.chartjs.org/docs/`