Projet CinéExplorer

Livrable 1 : Exploration et Base SQLite

OUDELET Clément
4A Info - Polytech Marseille

3 décembre 2025

Table des matières

1 Exploration des Données (T1.0)

1.1 Observations Clés

L'analyse exploratoire réalisée avec Pandas a révélé plusieurs caractéristiques importantes du dataset :

- **Volumétrie :** Le dataset contient plusieurs milliers de films et d'acteurs. La table de liaison `principals` est la plus dense.
- **Qualité des données :** Les fichiers utilisent le caractère `\N` pour désigner les valeurs nulles, traité lors de l'import.
- **Cohérence :** Des tests d'intégrité ont validé la correspondance entre les clés étrangères (`mid`, `pid`) à travers les fichiers.

1.2 Visualisations

Voici la distribution temporelle des films et les genres les plus représentés.

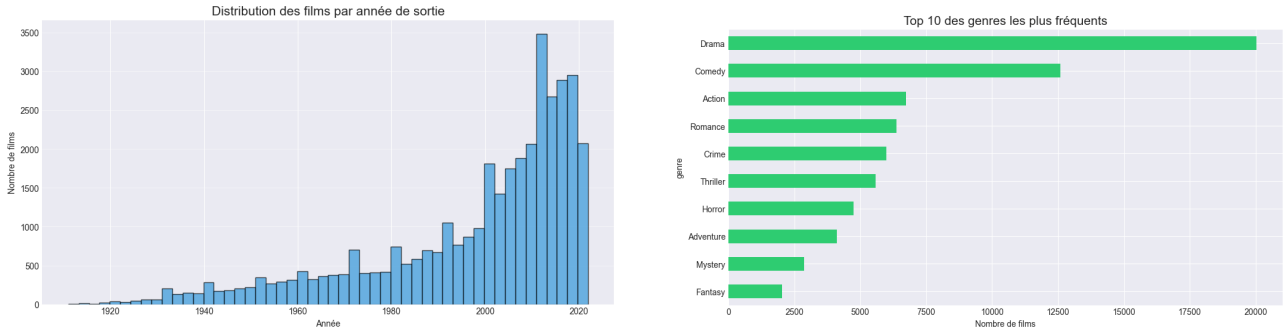

FIGURE 1 – Distribution des films par année et Top 10 des genres

2 Conception du Schéma Relationnel (T1.1)

Le schéma a été conçu en **3ème Forme Normale (3NF)**. Les listes séparées par des virgules (Genres, Titres alternatifs) ont été atomisées dans des tables dédiées.

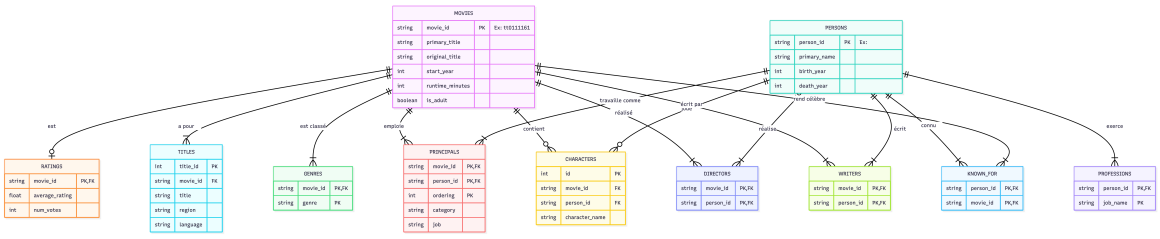

FIGURE 2 – Diagramme Entité-Relation (ER) du projet

3 Requêtes SQL (T1.3)

Nous avons implémenté les 9 requêtes demandées, couvrant des concepts SQL avancés.

3.1 Q1 : Filmographie d'un acteur

Objectif : Récupérer films, rôles et notes pour un acteur (recherche souple).

Récupération des données via jointures (`JOIN`) entre `persons`, `principals` et `movies`. L'utilisation de `LEFT JOIN` préserve les films même si les données annexes (rôle, note) sont manquantes, et `LIKE` permet une recherche approximative.

```sql
SELECT m.primary_title, m.start_year, c.character_name, 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 m.movie_id = c.movie_id AND pe.person_id = c.person_id
LEFT JOIN ratings r ON m.movie_id = r.movie_id
WHERE pe.primary_name LIKE ?
ORDER BY m.start_year DESC
```

3.2 Q2 : Top N films d'un genre

Objectif : Filtrage multicritères (année, genre) et tri sur la note.

Filtrage simultané sur trois tables optimisé par les index sur `genres` et `start_year`. Le tri descendant sur la note permet d'extraire efficacement le Top N.

```sql
SELECT m.primary_title, m.start_year, r.average_rating
FROM movies m
JOIN genres g ON m.movie_id = g.movie_id
JOIN ratings r ON m.movie_id = r.movie_id
WHERE g.genre = ? AND m.start_year BETWEEN ? AND ?
ORDER BY r.average_rating DESC LIMIT ?
```

3.3 Q3 : Acteurs multi-rôles

Objectif : Identifier les performances multiples dans un même film (ex : Dr. Folamour).

Détection des doublons sémantiques par agrégation `GROUP BY (film, acteur)` et filtrage post-agrégation avec `HAVING count > 1`.

```sql
SELECT pe.primary_name, m.primary_title, COUNT(c.character_name) as nb
FROM characters c
JOIN persons pe ON c.person_id = pe.person_id
JOIN movies m ON c.movie_id = m.movie_id
GROUP BY c.movie_id, c.person_id
HAVING nb > 1
ORDER BY nb DESC
```

3.4 Q4 : Collaborations Réalisateur-Acteur

Objectif : Double jointure pour relier deux personnes via un film commun.

Identification des paires via jointure entre la filmographie de l'acteur (sous-requête) et la table `directors` sur le champ commun `movie_id`.

```sql
SELECT d_pe.primary_name, COUNT(*) as collab
FROM principals p_actor
JOIN directors d ON p_actor.movie_id = d.movie_id
JOIN persons d_pe ON d.person_id = d_pe.person_id
WHERE p_actor.person_id = (SELECT person_id FROM persons WHERE primary_name LIKE ?)
GROUP BY d.person_id
ORDER BY collab DESC
```

3.5 Q5 : Genres populaires

Objectif : Agrégation (Moyenne) avec condition sur le groupe (HAVING).

Calcul de statistiques agrégées par genre. Le filtre `HAVING` garantit la pertinence statistique en excluant les genres ayant un échantillon trop faible (< 50 films).

```sql
SELECT g.genre, AVG(r.average_rating) as avg_rat, COUNT(*) as nb
FROM genres g
JOIN ratings r ON g.movie_id = r.movie_id
GROUP BY g.genre
HAVING avg_rat > 7.0 AND nb > 50
ORDER BY avg_rat DESC
```

3.6 Q6 : Évolution de carrière (CTE)

Objectif : Utilisation d'une Common Table Expression pour grouper par décennie.

Utilisation d'une **CTE** pour pré-calculer la décennie de chaque film, simplifiant l'agrégation finale et améliorant la lisibilité du code.

```sql
WITH career AS (
    SELECT
        (m.start_year / 10) * 10 as decade,
        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 ratings r ON m.movie_id = r.movie_id
    WHERE pe.primary_name = ? AND m.start_year IS NOT NULL
)
SELECT decade, COUNT(*) as count, AVG(average_rating)
FROM career
GROUP BY decade
ORDER BY decade
```

3.7 Q7 : Classement par genre (Window Function)

Objectif : Classement local (par genre) sans sous-requête corrélée.

Classement performant en une seule passe grâce à la fonction de fenêtrage `RANK()` partitionnée par genre, permettant de filtrer directement le Top 3.

```sql
WITH ranked AS (
    SELECT
        g.genre,
        m.primary_title,
        r.average_rating,
        RANK() OVER (PARTITION BY g.genre ORDER BY r.average_rating DESC) as rk
    FROM movies m
    JOIN genres g ON m.movie_id = g.movie_id
    JOIN ratings r ON m.movie_id = r.movie_id
    -- WHERE r.num_votes > 5000
)
SELECT genre, primary_title, average_rating, rk
FROM ranked
WHERE rk <= 3
```

3.8 Q8 : Carrière propulsée

Objectif : Identification de films à fort impact (>200k votes).

Isolation du premier succès chronologique (>200k votes) pour chaque acteur via `ROW_NUMBER()` ordonné par date, identifiant ainsi le rôle charnière.

```sql
WITH FirstHit AS (
    SELECT
        pe.primary_name,
        m.primary_title,
        m.start_year,
        r.num_votes,
        -- On classe par annee SEULEMENT les films > 200k pour chaque acteur
        ROW_NUMBER() OVER (
            PARTITION BY pe.person_id
            ORDER BY m.start_year ASC
        ) as hit_rank
    FROM persons pe
    JOIN principals p ON pe.person_id = p.person_id
    JOIN movies m ON p.movie_id = m.movie_id
    JOIN ratings r ON m.movie_id = r.movie_id
    WHERE r.num_votes > 200000
      AND p.category IN ('actor', 'actress') -- On cible les acteurs
)
SELECT primary_name, primary_title, start_year, num_votes
FROM FirstHit
WHERE hit_rank = 1 -- On prend uniquement leur PREMIER gros succes
ORDER BY num_votes DESC
LIMIT 20
```

3.9 Q9 : Requête Libre (Complexe)

Objectif : Films longs (>2h), très bien notés (>8.0) et en langue française.

Intersection stricte de trois critères (technique, critique, régional) réalisée via des jointures internes (`INNER JOIN`) sur les tables correspondantes.

```sql
SELECT m.primary_title, m.runtime_minutes, r.average_rating
FROM movies m
JOIN ratings r ON m.movie_id = r.movie_id
JOIN titles t ON m.movie_id = t.movie_id
WHERE m.runtime_minutes > 120
  AND r.average_rating > 8.0
  AND t.language = 'fr'
LIMIT 20
```

4 Indexation et Benchmark (T1.4)

4.1 Stratégie d'Indexation

Pour optimiser les performances, nous avons créé des index sur les colonnes de filtrage (`primary_name`, `genre`, `start_year`) et sur toutes les clés étrangères pour accélérer les jointures.

4.2 Résultats du Benchmark

Les tests ont été effectués sur un MacBook Air (Puce M3 16go RAM). Les résultats montrent un gain massif sur les requêtes lourdes (Q4, Q6), tandis que l'optimiseur de requêtes SQLite sur les petits ensembles de données (Q3, Q8) montre parfois un léger surcoût administratif de l'index.

Requête	**Sans Index (ms)**	**Avec Index (ms)**	**Gain (%)**
Q1 Filmographie	1059.1	402.1	62.0%
Q2 Top Films	23.2	9.2	60.3%
Q3 Multi-rôles	219.1	235.3	-7.4%
Q4 Collaborations	344.5	0.3	99.9%
Q5 Genres Pop	35.3	43.3	-22.4%
Q6 Carrière	628.1	0.4	99.9%
Q7 Classement	80.0	89.0	-11.2%
Q8 Propulsés	3.4	3.9	-13.2%
Q9 Complexe	7.0	0.7	90.0%

Table 1 – Comparatif des temps d'exécution (Moyenne)

5 Conclusion Phase 1

Cette première phase a permis de valider la structure relationnelle des données. L'importation optimisée (transactions) et l'ajout d'index stratégiques ont rendu la base performante et prête pour la migration vers MongoDB (Phase 2).