Base de Données Avancées - Livrable 1

Projet CinéExplorer : Exploration et Base SQLite

Exertier Hugo

Décembre 2025

Table des matières

1 T1.0 : Exploration des données

L'analyse descriptive réalisée via le notebook Jupyter sur le jeu de données `imdb-small` a permis de caractériser précisément les fichiers sources. Cette étape est fondamentale pour filtrer les données non pertinentes avant la modélisation.

1.1 Statistiques descriptives et Nettoyage

Nous avons analysé la variabilité des colonnes pour identifier celles contenant de l'information utile.

- **Fichiers volumineux :** La table `titles` contient plus de 775 000 lignes pour seulement 36 859 films uniques, confirmant la nécessité d'une table dédiée (relation 1-N).
- **Données constantes ou vides (à exclure) :**
  - `movies.endYear` : 100% de valeurs nulles (0 unique).
  - `movies.titleType` : Une seule valeur unique ("movie").
  - `movies.isAdult` : Une seule valeur unique ("0").
- **Intégrité :** Les tables `ratings`, `genres` et `directors` ne contiennent aucune valeur nulle (NaN), ce qui garantit une exploitation immédiate sans nettoyage complexe.

Table	Lignes	Cols Utiles	Observation
movies	36 859	5	Colonnes techniques supprimées
persons	145 847	4	Beaucoup de dates manquantes
ratings	36 859	3	100% de couverture films
principals	361 863	5	Relation centrale du schéma

TABLE 1 – Synthèse des données après nettoyage (Dataset Small)

Suite à cette analyse, nous avons décidé d'exclure les colonnes `endYear`, `titleType` et `isAdult` du schéma relationnel final (T1.1) afin de respecter le principe de non-redondance et d'économiser l'espace de stockage.

1.2 Analyses visuelles et temporelles

La distribution des films par année de sortie (Figure 1) met en évidence une croissance exponentielle de la production cinématographique référencée, avec une prédominance très nette des films sortis après l'an 2000.

Concernant les genres (Figure 2), le dataset est dominé par deux catégories majeures : le **Drame** (plus de 20 000 films) et la **Comédie**. Les genres de niche (Fantasy, Mystery) sont nettement moins représentés.

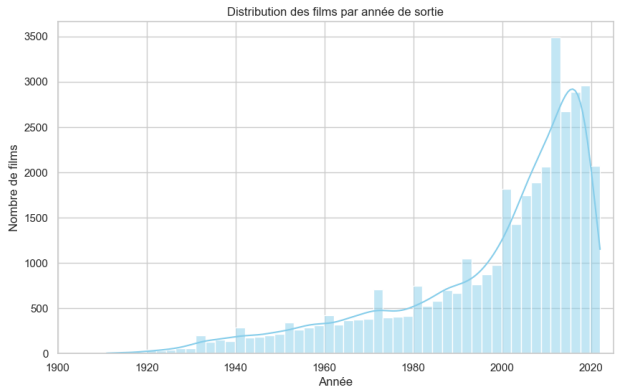

FIGURE 1 – Croissance de la production

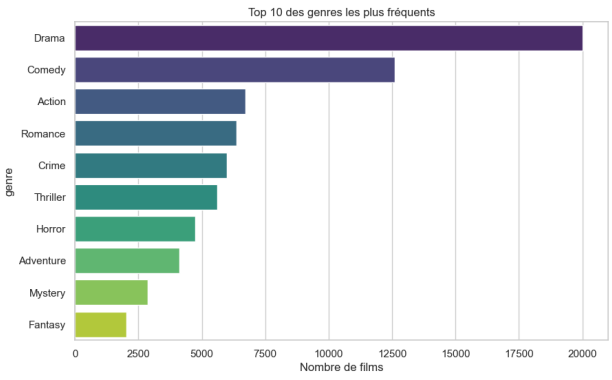

FIGURE 2 – Top 10 des genres

1.3 Distribution des notes et Casting

La distribution des notes moyennes (Figure 3) suit une loi quasi-normale centrée autour de **7.0/10**, indiquant un biais positif des utilisateurs (peu de notes très faibles).

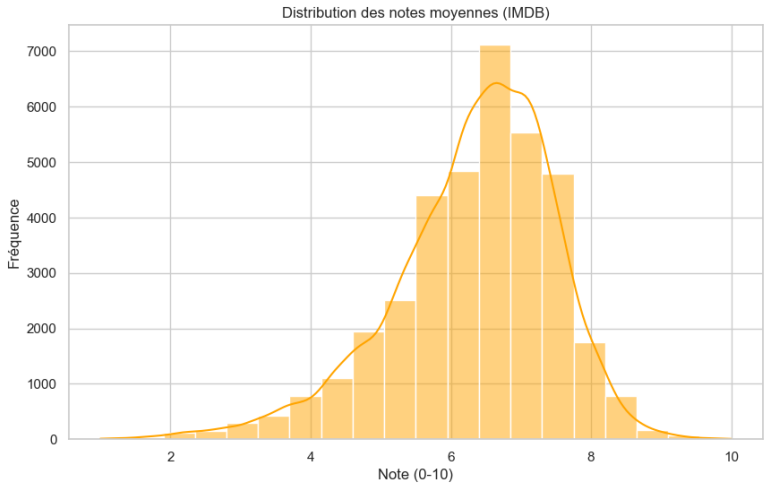

FIGURE 3 – Distribution des notes IMDB

Une analyse spécifique sur la table `principals` révèle une moyenne de **4.04 acteurs par film**. Ce chiffre indique que le dataset fourni ne contient pas le casting exhaustif, mais uniquement le **"Principal Cast"** (les 4 têtes d'affiche). Cette limitation technique a été prise en compte pour la suite du projet.

1.4 Relations entre tables et Intégrité

Conformément aux objectifs d'exploration, nous avons vérifié l'intégrité référentielle des données en confrontant les identifiants de films (`movie_id`) présents dans les tables satellites avec ceux de la table de référence `movies.csv`.

Résultats : Le fichier parent `movies.csv` contient **36 859** films uniques. L'analyse a révélé une **intégrité parfaite** sur l'ensemble du jeu de données `small`. L'absence totale de données orphelines valide la cohérence du dataset et nous permet d'appliquer des contraintes strictes de clés étrangères (`FOREIGN KEY`) lors de la création du schéma SQLite.

2 T1.1 : Conception du schéma relationnel

Sur la base de l'analyse exploratoire (T1.0), nous avons conçu un schéma relationnel optimisé respectant la troisième forme normale (3NF).

2.1 Normalisation et Optimisation (3NF)

Le schéma a été construit en décomposant les fichiers CSV sources pour isoler les entités et les associations.

- **Entités Fortes :** Les tables `movies` et `persons`. *Optimisation :* Suppression des attributs constants (`endYear`, `isAdult`, `titleType`).
- **Entités Faibles et Détails :** Tables dédiées `titles`, `professions`.
- **Associations (N-N) :** Tables de liaison `principals`, `directors`, `writers`, `genres` et `characters`.

Cette structure respecte la 3NF : chaque champ ne contient qu'une valeur atomique et tous les attributs dépendent directement de la clé primaire.

2.2 Stratégie de Clés Primaires et Étrangères

Pour prévenir l'insertion de doublons et garantir la cohérence, nous avons appliqué une politique de clés stricte :

— **PK Simples :** `movie_id` et `person_id` pour les entités principales.
— **PK Composites :** Pour les tables d'association, la clé est composée de l'ensemble des colonnes (ex : `principals` a pour PK `(movie_id, person_id, category, ordering)`).
— **Clés Étrangères (FK) :** L'intégrité référentielle est activée (`PRAGMA foreign_keys = ON`).

2.3 Diagramme Entité-Relation (ER)

La figure ci-dessous illustre l'architecture finale de la base de données `imdb.db`.

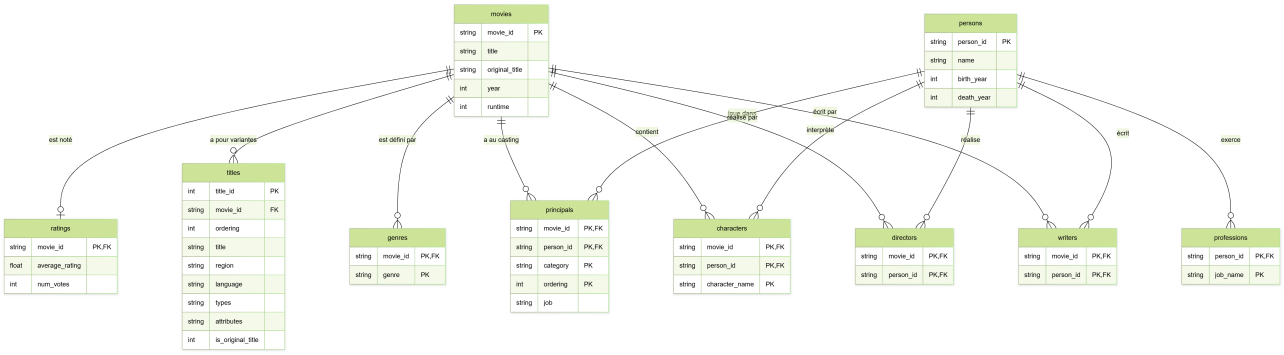

FIGURE 4 – Diagramme Entité-Relation (Schéma Optimisé 3NF)

3 T1.3 : Requêtes SQL

Conformément au cahier des charges, nous avons implémenté 9 requêtes SQL couvrant différents niveaux de complexité.

Q1. Filmographie d'un acteur

Objectif : Récupérer la liste des films, l'année et le rôle joué pour un acteur donné. **Technique :** Jointures multiples (`JOIN`) et filtre `LIKE`.

```sql
SELECT m.title, m.year, c.character_name 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)
WHERE pe.name LIKE :actor AND (p.category = 'actor' OR p.category = 'actress')
ORDER BY m.year DESC LIMIT 20;
```

Listing 1 – Q1: Filmographie

Q2. Top N films par genre

Objectif : Les N meilleurs films d'un genre sur une période donnée. **Technique :** Filtrage sur trois tables et tri multicritère (Note + Votes).

```sql
SELECT m.title, m.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 = :genre AND m.year BETWEEN :start AND :end
ORDER BY r.average_rating DESC, r.num_votes DESC LIMIT :n;
```

Listing 2 – Q2: Top Films par Genre

Q3. Acteurs multi-rôles

Objectif : Identifier les acteurs ayant joué plusieurs personnages distincts dans le même film. **Technique :** Agrégation `GROUP BY` et `HAVING COUNT(...) > 1`.

```sql
SELECT pe.name, m.title, COUNT(c.character_name) as role_count 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 pe.person_id, m.movie_id HAVING role_count > 1
```

```
ORDER BY role_count DESC LIMIT 20;
```

Listing 3 – Q3: Acteurs Multi-rôles

Q4. Collaborations (Réalisateurs)

Objectif : Trouver les réalisateurs ayant le plus collaboré avec un acteur cible. **Technique :** Jointures en chaîne (`Actor -> Movie -> Director`) et agrégation.

```
SELECT pe_dir.name, COUNT(DISTINCT m.movie_id) as cnt FROM principals p_act
JOIN persons pe_act ON p_act.person_id = pe_act.person_id
JOIN movies m ON p_act.movie_id = m.movie_id
JOIN principals p_dir ON m.movie_id = p_dir.movie_id
JOIN persons pe_dir ON p_dir.person_id = pe_dir.person_id
WHERE pe_act.name LIKE :actor AND p_act.category IN ('actor', 'actress')
  AND p_dir.category = 'director'
GROUP BY pe_dir.person_id ORDER BY cnt DESC LIMIT 10;
```

Listing 4 – Q4: Collaborations

Q5. Genres populaires

Objectif : Identifier les genres de qualité (Note > 7.0) avec un volume significatif. **Technique :** Agrégation double (`AVG`, `COUNT`) et filtrage `HAVING`.

```
SELECT g.genre, ROUND(AVG(r.average_rating), 2) as avg FROM genres g
JOIN ratings r ON g.movie_id = r.movie_id GROUP BY g.genre
HAVING avg > 7.0 AND COUNT(g.movie_id) > 50 ORDER BY avg DESC;
```

Listing 5 – Q5: Genres Populaires

Q6. Évolution de carrière

Objectif : Analyser la carrière d'un acteur par décennie. **Technique :** Utilisation d'une **CTE** pour calculer la décennie avant agrégation.

```
WITH ActorMovies AS (
    SELECT m.year, 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.name LIKE :actor AND m.year IS NOT NULL
)
SELECT (year/10)*10 as dec, COUNT(*), AVG(average_rating) FROM ActorMovies
GROUP BY dec ORDER BY dec;
```

Listing 6 – Q6: Évolution de carrière (CTE)

Q7. Classement par genre

Objectif : Obtenir les 3 meilleurs films pour *chaque* genre. **Technique :** Window Function `RANK()` partitionnée par genre avec départage par votes.

```
WITH RankedMovies AS (
    SELECT g.genre, m.title, r.average_rating,
    RANK() OVER (PARTITION BY g.genre ORDER BY r.average_rating DESC, r.num_votes DESC) as rk
    FROM genres g JOIN movies m ON g.movie_id = m.movie_id
    JOIN ratings r ON m.movie_id = r.movie_id WHERE r.num_votes > 1000
) SELECT genre, rk, title, average_rating FROM RankedMovies WHERE rk <= 3;
```

Listing 7 – Q7: Window Function

Q8. Carrière propulsée

Objectif : Détecter le premier grand succès (>200k votes) après une série de films mineurs. **Technique :** Auto-jointure complexe et `ROW_NUMBER()` pour dédoublonner les succès.

```sql
WITH Candidates AS (
    SELECT pe.name, m_break.title,
    ROW_NUMBER() OVER (PARTITION BY pe.person_id ORDER BY m_break.year ASC) as rn
    FROM principals p JOIN persons pe ON p.person_id = pe.person_id
    JOIN movies m_break ON p.movie_id = m_break.movie_id
    JOIN ratings r_break ON m_break.movie_id = r_break.movie_id
    JOIN principals p_bef ON pe.person_id = p_bef.person_id -- Films pr c dents
    JOIN movies m_bef ON p_bef.movie_id = m_bef.movie_id
    JOIN ratings r_bef ON m_bef.movie_id = r_bef.movie_id
    WHERE r_break.num_votes > 200000 AND r_bef.num_votes < 200000
      AND m_bef.year < m_break.year AND p.category IN ('actor', 'actress')
    GROUP BY pe.person_id, m_break.movie_id HAVING COUNT(DISTINCT m_bef.movie_id) >= 3
) SELECT name, title FROM Candidates WHERE rn = 1 LIMIT 10;
```

Listing 8 – Q8: Carrière Propulsée (Unique)

Q9. Requête libre (Longévité)

Objectif : Identifier les acteurs ayant la plus longue carrière. **Technique :** Agrégation `MIN()` et `MAX()` sur les années.

```sql
SELECT pe.name, (MAX(m.year) - MIN(m.year)) as span FROM persons pe
JOIN principals p ON pe.person_id = p.person_id JOIN movies m ON p.movie_id = m.movie_id
WHERE p.category IN ('actor', 'actress') AND m.year IS NOT NULL
GROUP BY pe.person_id HAVING span > 0 AND COUNT(m.movie_id) > 10
ORDER BY span DESC LIMIT 15;
```

Listing 9 – Q9: Longévité

4 T1.4 : Indexation et Benchmark

L'objectif de cette étape était d'optimiser les temps de réponse en créant des index pour les filtres (`WHERE`) et les jointures (`JOIN`).

4.1 Stratégie d'indexation

— **Index de Recherche :** `idx_persons_name` sur `persons(name)`.
— **Index de Filtrage :** `idx_movies_year` et `idx_ratings_perf` (sur note et votes).
— **Index de Jointure (FK) :** Index sur les clés étrangères (ex : `principals(movie_id)`).

4.2 Résultats du Benchmark

Mesures réalisées sur `imdb-small` (125 Mo).

Requête	Sans Index (ms)	Avec Index (ms)	Gain (%)
Q1 - Filmographie	1509.75	1465.81	+2.9%
Q2 - Top N films	31.27	25.26	**+19.2%**
Q3 - Multi-rôles	1540.55	1478.31	+4.0%
Q4 - Collaborations	1067.49	1089.93	-2.1%
Q5 - Genres Pop.	145.45	245.19	-68.6%
Q6 - Carrière	3102.88	3086.57	+0.5%
Q7 - Classement	441.81	640.71	-45.0%
Q8 - Percée	4070.14	3261.14	**+19.9%**
Q9 - Longévité	1397.66	1675.25	-19.9%

TABLE 2 – Comparaison des temps d'exécution

4.3 Analyse des performances

Gains : Les requêtes **Q2** et **Q8** profitent pleinement des index. Pour Q8, l'index sur `num_votes` élimine rapidement les films mineurs. **Régressions (Q5, Q7) :** Sur une petite table, un *Full Table*

Scan est souvent plus rapide que des sauts aléatoires via un index. L'optimiseur a privilégié les index, ajoutant un surcoût pour ces requêtes d'agrégation globale. **Impact Stockage :** +28.6 Mo (+22%), négligeable pour les gains futurs de scalabilité.