# Livrable 1 - Exploration et Base SQLite

**Projet :** Cine_Explorer (Medium)
**Auteur :** Antoine CHIAUSA

---

# 1. Résumé de l'Exploration

## 1.1 Méthodologie d'Exploration

Le notebook exploration.ipynb effectue une exploration complète de la base de données IMDB à travers plusieurs étapes :

### Chargement des données

D'abord, j'ai chargé les 11 fichiers CSV dans des DataFrames pandas :

- characters.csv, directors.csv, genres.csv, knownformovies.csv
- movies.csv, persons.csv, principals.csv, professions.csv
- ratings.csv, titles.csv, writers.csv

### Analyse statistique descriptive

Pour chaque table, j'ai examiné :

- Dimensions : nombre de lignes et colonnes
- Types de données : vérifier que les types sont correctement interprétés
- Valeurs manquantes (NaN) : identifier les colonnes incomplètes
- Valeurs uniques : comprendre la cardinalité de chaque colonne

### Analyse exploratoire (EDA)

J'ai créé 4 visualisations et analyses :

Distribution des films par année (histogramme) :

- Montre la croissance exponentielle des films depuis 1980
- Pic de production entre 2010-2020
- Couverture historique de 130 ans (1890-2020)

Top 10 des genres les plus fréquents :

- Les genres dominants sont Drama, Comedy, Documentary, Action, Romance
- Drama est largement surreprésenté

Distribution des notes (ratings) (histogramme) :

- Notes comprises entre 1.0 et 10.0
- Tendance vers les bonnes notes (distribution centrée sur 7-8)

Nombre moyen d'acteurs par film :

- Calcul de la moyenne pour comprendre la densité des castings
- Permet de valider la cardinalité de la table characters

### Vérification de l'intégrité référentielle

J'ai effectué deux vérifications critiques :

Recherche d'orphelins (vers movie) :

- Pour chaque table ayant une colonne mid, vérifier qu'aucun mid n'existe sans son film associé
- Résultat : 0 orphelins détectés

Couverture complète :

- Vérifier que tous les mid de la table movies sont représentés dans les autres tables
- Cela confirme la cohérence des clés étrangères

## 1.2 Qualité des Données

Les résultats de l'exploration montrent :

- 0 orphelins : Toutes les clés étrangères sont valides
- Intégrité référentielle respectée : Aucune anomalie détectée
- Valeurs manquantes gérées : Conversions NaN vers NULL lors de l'import
- Doublons détectés et supprimés : 1 247 doublons supprimés lors du chargement
- Types de données cohérents : Pas de conversions aberrantes

---

# 2. Diagramme Entité-Relation (ER)

Voir capture d'écran de l'exécution de create_schema.py

Le schéma suit une normalisation 3NF avec :

- 2 entités principales : MOVIES et PERSONS
- 9 entités associatives : pour les relations N:N
- Intégrité référentielle : Clés étrangères avec ON DELETE CASCADE

Vous pouvez afficher les tables et les dépendances en exécutant le script `create_schema.py`

---

# 3. Requêtes SQL Implémentées

## Q1 - Filmographie d'un Acteur

Objectif : Récupérer tous les films d'un acteur, ordonnés chronologiquement

```
SELECT m.primaryTitle
FROM movies m
JOIN principals p ON m.mid = p.mid
JOIN persons pe ON p.pid = pe.pid
WHERE pe.primaryName LIKE ?
ORDER BY m.startYear ASC
```

Construction de la requête :

- Je commence par la table movies pour avoir accès aux titres
- Je dois joindre principals car c'est la table intermédiaire qui relie les acteurs aux films
- Je joins persons pour accéder au nom de l'acteur (primaryName)
- Le WHERE utilise LIKE avec des jokers pour une recherche flexible (ex: "Tom" trouve "Tom Hanks", "Tom Cruise", etc.)
- Je trie par startYear pour avoir la filmographie en ordre chronologique
- Résultat : Liste simple avec juste les titres

---

## Q2 - Top N Films par Genre et Période

Objectif : Trouver les n meilleurs films d'un genre sur une période donnée

```
SELECT m.primaryTitle, r.averageRating
FROM movies m
JOIN ratings r ON m.mid = r.mid
JOIN genres g ON m.mid = g.mid
WHERE g.genre = ? AND m.startYear BETWEEN ? AND ?
ORDER BY r.averageRating DESC
LIMIT ?
```

Construction de la requête :

- Je pars de movies pour les titres
- Je joins ratings pour accéder aux notes (averageRating)

- Je joins genres pour filtrer par genre spécifique
- Le WHERE filtre sur 3 critères :
  - Genre exact (pas de LIKE)
  - Année entre deux bornes avec BETWEEN
- Je trie en ordre décroissant par note (DESC)
- Le LIMIT retourne seulement les N premiers résultats
- Résultat : Titre + Note du film

---

## Q3 - Acteurs avec Rôles Multiples

Objectif : Trouver les films où un acteur a joué plusieurs personnages différents

```
SELECT m.primaryTitle, COUNT(c.name) as nombre_de_roles
FROM movies m
JOIN characters c ON m.mid = c.mid
JOIN persons p ON c.pid = p.pid
WHERE p.primaryName LIKE ?
GROUP BY m.mid
HAVING nombre_de_roles > 1
ORDER BY nombre_de_roles DESC
```

Construction de la requête :

- Je joins characters pour accéder aux rôles spécifiques
- Je joins persons pour filtrer par nom d'acteur
- Différence clé avec Q1 : j'utilise characters et non principals
  - characters contient le nom du personnage joué
  - principals ne contient que la catégorie générale (actor, actress, etc.)
- Le GROUP BY regroupe par film (m.mid)
- Le COUNT(c.name) compte le nombre de personnages différents dans chaque film
- Le HAVING filtre les groupes pour ne garder que ceux avec plus d'1 rôle
- Résultat : Titre du film + Nombre de rôles

---

## Q4 - Collaborations (Acteur-Réalisateurs)

Objectif : Trouver tous les réalisateurs ayant travaillé avec un acteur donné

```
SELECT pe.primaryName, COUNT(m.mid) as nombre_de_films
FROM persons pe
JOIN directors d ON pe.pid = d.pid
```

```
JOIN movies m ON d.mid = m.mid
WHERE m.mid IN (
    SELECT m2.mid
    FROM movies m2
    JOIN principals pr ON m2.mid = pr.mid
    JOIN persons pe ON pr.pid = pe.pid
    LEFT JOIN characters c ON c.mid = m2.mid AND c.pid = pe.pid
    WHERE pe.primaryName LIKE ?
)
GROUP BY pe.pid
ORDER BY nombre_de_films DESC
```

Construction de la requête :

- C'est une requête avec sous-requête (subquery)
- La sous-requête : trouve tous les films où joue l'acteur recherché
  - Joins principals et persons pour identifier les films de l'acteur
  - LEFT JOIN characters pour avoir le contexte (optionnel)
- La requête principale : parmi ces films, trouve les réalisateurs
  - Je joins directors à persons pour obtenir les noms des réalisateurs
  - COUNT(m.mid) compte le nombre de films ensemble
- Le GROUP BY pe.pid regroupe par réalisateur (évite les doublons)
- Je trie par nombre de films en commun (décroissant)
- Résultat : Nom du réalisateur + Nombre de films avec l'acteur

---

## Q5 - Genres Populaires

Objectif : Trouver les genres avec une bonne note moyenne (>7.0) et suffisamment de films (>50)

```
SELECT g.genre, AVG(r.averageRating) as note_moyenne,
       COUNT(m.mid) as nombre_de_films
FROM genres g
JOIN movies m ON g.mid = m.mid
JOIN ratings r ON m.mid = r.mid
GROUP BY g.genre
HAVING note_moyenne > 7.0 AND nombre_de_films > 50
ORDER BY note_moyenne DESC
```

Construction de la requête :

- Je joins movies et ratings à genres pour avoir tous les films d'un genre avec leurs notes
- Le GROUP BY g.genre regroupe tous les films par genre

- Les fonctions d'agrégation :
  - AVG(r.averageRating) : moyenne des notes
  - COUNT(m.mid) : nombre de films distincts
- Le HAVING filtre les genres (différent du WHERE qui filtre les lignes individuelles)
  - note_moyenne > 7.0 : exclure les genres mal notés
  - nombre_de_films > 50 : avoir un échantillon significatif
- Je trie par note moyenne descendante pour voir les meilleurs genres
- Résultat : Genre + Note moyenne + Nombre de films

---

## Q6 - Classement par Genre (Top-3)

Objectif : Pour chaque genre, obtenir les 3 meilleurs films

```
WITH films_classes AS (
    SELECT g.genre, m.primaryTitle, r.averageRating,
           RANK() OVER (PARTITION BY g.genre ORDER BY r.averageRating
DESC) as rang
    FROM movies m
    JOIN genres g ON m.mid = g.mid
    JOIN ratings r ON m.mid = r.mid
)
SELECT genre, primaryTitle, averageRating, rang
FROM films_classes
WHERE rang <= 3
ORDER BY genre, rang
```

Construction de la requête :

- J'utilise une CTE (Common Table Expression) avec WITH ... AS
- Dans la CTE, je joins genres et ratings à movies
- Utilisation d'une window function RANK() OVER (PARTITION BY ... ORDER BY ...)
  - PARTITION BY g.genre : créer des groupes par genre
  - ORDER BY r.averageRating DESC : trier par note décroissante dans chaque groupe
  - RANK() attribue un rang (1, 2, 3, ...) à chaque film dans son genre
- Dans la requête principale, je filtre WHERE rang <= 3
- Je trie par genre puis par rang pour une présentation claire
- Résultat : Genre + Titre + Note + Rang (1, 2, ou 3)

---

## Q7 - Carrière Propulsée

Objectif : Trouver les personnes dont la carrière a décollé grâce à un film à succès (>200k votes)

```
WITH film_de_percee AS (
    SELECT DISTINCT
        kfm.pid,
        m.mid,
        m.primaryTitle,
        m.startYear,
        r.numVotes
    FROM knownformovies kfm
    JOIN movies m ON kfm.mid = m.mid
    JOIN ratings r ON m.mid = r.mid
    WHERE r.numVotes > 200000
),
avant_perce AS (
    SELECT kfm.pid, m.primaryTitle, m.startYear
    FROM knownformovies kfm
    JOIN movies m ON kfm.mid = m.mid
    JOIN ratings r ON m.mid = r.mid
    WHERE r.numVotes < 200000
    GROUP BY kfm.pid
    HAVING COUNT(DISTINCT m.mid) > 0
)
SELECT DISTINCT
    per.primaryName,
    fp.primaryTitle,
    fp.startYear,
    r.numVotes
FROM avant_perce avp
JOIN persons per ON avp.pid = per.pid
JOIN film_de_percee fp ON avp.pid = fp.pid
JOIN ratings r ON fp.mid = r.mid
GROUP BY per.primaryName, avp.primaryTitle
ORDER BY avp.startYear ASC
```

Construction de la requête :

- C'est une requête complexe avec 2 CTEs et logique de comparaison
- CTE 1 (film_de_percee) : films à succès (>200k votes) d'une personne
  - DISTINCT pour éviter les doublons
- CTE 2 (avant_perce) : personnes ayant eu des films moins célèbres (<200k votes)
  - GROUP BY et HAVING pour vérifier qu'ils ont au moins 1 film
- Requête principale : jointure entre les deux CTEs
  - Je cherche les personnes de "avant_perce" qui sont aussi dans "film_de_percee"
  - Cela indique une progression de popularité

- Résultat : Nom personne + Film percée + Année + Nombre de votes

---

## Q8 - Films par Réalisateur et Genre

Objectif : Tous les films d'un réalisateur, organisés par genre et note

```
SELECT
    g.genre,
    m.primaryTitle,
    r.averageRating,
    r.numVotes,
    m.startYear
FROM persons pe
JOIN directors d ON pe.pid = d.pid
JOIN movies m ON d.mid = m.mid
JOIN genres g ON m.mid = g.mid
JOIN ratings r ON m.mid = r.mid
WHERE pe.primaryName LIKE ?
ORDER BY g.genre, r.averageRating DESC
```

Construction de la requête :

- 5 jointures successives :
  1. persons vers directors : filtrer le réalisateur
  2. directors vers movies : obtenir ses films
  3. movies vers genres : genres de chaque film
  4. movies vers ratings : notes et votes
- Le WHERE filtre par nom du réalisateur avec LIKE
- Je sélectionne multiple colonnes : genre, titre, note, votes, année
- Tri par genre (alphabétique) puis par note décroissante
- Résultat : Genre + Titre + Note + Votes + Année

---

# 4. Tableau de Benchmark Complet

| Requête | Sans Index (ms) | Avec Index (ms) | Gain (%) |
|---|---|---|---|
| Q1 - Filmographie | 5913.52 | 159.20 | +97.3% |
| Q2 - Top N films | 136.39 | 51.46 | +62.3% |
| Q3 - Multi-rôles | 2989.53 | 152.33 | +94.9% |
| Q4 - Collaborations | 6248.69 | 401.26 | +93.6% |

| Requête | Sans Index (ms) | Avec Index (ms) | Gain (%) |
|---|---|---|---|
| Q5 - Genres populaires | 693.30 | 713.29 | -2.9% |
| Q6 - Classement genre | 1263.54 | 1328.52 | -5.1% |
| Q7 - Carrière propulsée | 7386.87 | 6356.35 | +14.0% |
| Q8 - Films réalisateur | 1012.74 | 1122.44 | -10.8% |
| TOTAL | 25644.59 | 10284.85 | +59.9% |

### Impact sur la Base de Données

```
Taille initiale        : 428.63 MB
Taille après index     : 674.89 MB
Augmentation           : +246.26 MB (+57.4%)
Temps économisé global : 15359.75 ms
```

---

## 5. Index Créés et Justifications

### Index Implémentés (13 au total)

| Index | Table | Colonnes | Justification |
|---|---|---|---|
| idx_persons_name | persons | primaryName | Recherche par nom (Q1, Q3, Q4, Q8 utilisent WHERE LIKE) |
| idx_principals_mid | principals | mid | Jointure principals vers movies (45M lignes) |
| idx_principals_pid | principals | pid | Jointure principals vers persons |
| idx_directors_mid | directors | mid | Jointure directors vers movies (Q4, Q8) |
| idx_directors_pid | directors | pid | Jointure directors vers persons (Q4, Q8) |
| idx_genres_mid | genres | mid | Jointure genres vers movies (Q2, Q5, Q6, Q8) |
| idx_genres_genre | genres | genre | Filtrage WHERE g.genre = ? (Q2, Q5) |
| idx_ratings_mid | ratings | mid | Jointure ratings vers movies |
| idx_ratings_numVotes | ratings | numVotes | Filtrage sur votes (Q7) |

| Index | Table | Colonnes | Justification |
|---|---|---|---|
| idx_characters_mid | characters | mid | Jointure characters vers movies (Q3) |
| idx_characters_pid | characters | pid | Jointure characters vers persons (Q3) |
| idx_knownformovies_pid | knownformovies | pid | Jointure knownformovies vers persons (Q7) |
| idx_knownformovies_mid | knownformovies | mid | Jointure knownformovies vers movies (Q7) |

### Stratégie d'Indexation

Principes appliqués :

1. Index sur toutes les clés étrangères : Essentiels pour les jointures
2. Index sur les colonnes de filtrage : primaryName, genre, numVotes
3. Index sur les colonnes ORDER BY : Pour optimiser le tri
4. Pas d'index composé : Complexité de maintenance vs bénéfice limité

Observations :

- Q1, Q3, Q4 : Gains massifs (93-97%) grâce aux index sur principals et persons
- Q2 : Gain de 62% grâce à index sur genres et ratings
- Q5, Q6, Q8 : Performance dégradée car la requête agrégée/triée affecte trop de lignes

---

## Conclusion

L'ajout de 13 index diminue de 59.9% du temps total des requêtes. Les jointures multi-tables bénéficient énormément des index sur clés étrangères, tandis que les requêtes d'agrégation complexes voient peu d'amélioration voire une dégradation due au coût de maintenance des index.