Livrable 1: Exploration et SQLite

BOUNGO BATTISTE FISA 4A

1. Resume de l'exploration

1.1 Objectif

Explorer et analyser le dataset IMDB en construisant une base de donnees SQLite normalisee, puis developper des requetes SQL complexes pour l'analyse de donnees cinematographiques.

1.2 Observations cles

Donnees importees

L'exploration initiale des fichiers CSV a revele:

- **Films**: 2,107 titres
- **Personnes**: 9,461 individus (realisateurs, scenaristes)
- **Genres**: 5,619 associations film-genre
- **Notes**: 1,887 evaluations
- **Realisateurs**: 2,357 relations film-realisateur
- **Scenaristes**: 3,182 relations film-scenariste

Distribution temporelle

- **Periode couverte**: Films majoritairement entre 1900 et 2020
- **Concentration**: Pics de production dans les annees 2000-2015
- **Annees manquantes**: ~10% des films sans annee renseignee

Genres cinematographiques

Les genres les plus representes:
1. Drama (32% des films)
2. Documentary (18%)
3. Comedy (15%)

4. Short (12%)
5. Horror, Action, Romance (~5% chacun)

Notes et popularite

- **Note moyenne globale**: 6.8/10
- **Films tres bien notes (>8.0)**: ~8% du dataset
- **Films populaires (>100k votes)**: ~5% du dataset
- **Distribution**: Courbe normale centree autour de 6.5-7.5

Integrite referentielle

Avant nettoyage:
- **Orphelins genres**: 142 associations sans film correspondant
- **Orphelins ratings**: 87 notes sans film
- **Orphelins directors**: 234 relations invalides
- **Solution**: Filtrage a l'import pour garantir l'integrite

1.3 Graphiques cles

Les visualisations suivantes ont ete generees dans le notebook `data/exploration.ipynb` :

Graphique 1: Distribution des films par annee

> Histogramme montrant la production cinematographique par decennie.
> Tendance: Croissance exponentielle depuis 1970, pic dans les annees 200
> 0.

Graphique 2: Distribution des genres

> Diagramme en barres des 10 genres principaux.
> Observation: Drama largement dominant, suivi des documentaires.

Graphique 3: Distribution des notes

> Histogramme de la distribution des notes moyennes.
> Observation: Distribution normale, mediane a 6.8, ecart-type ~1.2.

Graphique 4: Relation votes vs note

Scatter plot montrant la correlation entre nombre de votes et note.
Observation: Films populaires (>50k votes) ont tendance a avoir des notes plus elevees (>7.0).

1.4 Qualite des donnees

Points forts:
- Identifiants uniques pour chaque entite
- Typage coherent des donnees numeriques
- Peu de valeurs aberrantes

Points d'amelioration:
- ~10% de valeurs manquantes pour startYear
- ~5% de valeurs manquantes pour birthYear/deathYear
- Quelques incohérences (endYear < startYear)

---

2. Diagramme ER du schema relationnel

2.1 Schema conceptuel (3NF)

```
MOVIES(mid PK, titleType, primaryTitle, originalTitle,
       isAdult, startYear, endYear, runtimeMinutes)
   │1
   │N
   ├── GENRES(mid FK, genre)              PK(mid, genre)
   └── RATINGS(mid PK/FK, averageRating, numVotes)


PERSONS(pid PK, primaryName, birthYear, deathYear)
   │1
   │N
   ├── DIRECTORS(mid FK, pid FK)          PK(mid, pid)
   └── WRITERS(mid FK, pid FK)            PK(mid, pid)
```

2.2 Contraintes d'integrite

Cles primaires (PK)

- `movies.mid` - Identifiant unique du film
- `persons.pid` - Identifiant unique de la personne
- `ratings.mid` - Identifiant unique (aussi FK vers movies)
- `(genres.mid, genres.genre)` - Cle composee
- `(directors.mid, directors.pid)` - Cle composee
- `(writers.mid, writers.pid)` - Cle composee

Cles etrangeres (FK)

- `genres.mid → movies.mid` (ON DELETE CASCADE)
- `ratings.mid → movies.mid` (ON DELETE CASCADE)
- `directors.mid → movies.mid` (ON DELETE CASCADE)
- `directors.pid → persons.pid` (ON DELETE CASCADE)
- `writers.mid → movies.mid` (ON DELETE CASCADE)
- `writers.pid → persons.pid` (ON DELETE CASCADE)

Contraintes de domaine

- `movies.primaryTitle NOT NULL`
- `persons.primaryName NOT NULL`
- `movies.isAdult` $\in \{0, 1\}$
- `ratings.averageRating` $\in [0.0, 10.0]$

2.3 Normalisation (3NF)

Le schema respecte les trois formes normales:

1NF (Premiere Forme Normale):
- Toutes les colonnes contiennent des valeurs atomiques
- Pas d'attributs multi-values
- Chaque table a une cle primaire

2NF (Deuxieme Forme Normale):
- Respect de la 1NF

- Pas de dependances partielles
- Tous les attributs non-cles dependent de la totalite de la cle primaire

3NF (Troisieme Forme Normale):
- Respect de la 2NF
- Pas de dependances transitives
- Les attributs non-cles dependent uniquement de la cle primaire

---

3.1 Les 9 requêtes SQL complexes

Les neuf requêtes illustrent une exploitation avancée de SQL pour analyser des données cinématographiques sous différents angles (carrières, genres, collaborations, qualité).

1. **Filmographie d'un artiste**

   Récupère tous les films d'une personne (réalisateur ou scénariste) en combinant plusieurs rôles, avec intégration des notes et tri chronologique.

2. **Top N films par genre**

   Sélectionne les meilleurs films d'un genre et d'une période donnée grâce à des jointures multiples, un tri par qualité et popularité, et une limitation des résultats.

3. **Acteurs multi-rôles (démo)**

   Identifie les acteurs ayant interprété plusieurs personnages dans un même film via agrégation et filtrage, à titre illustratif.

4. **Collaborations entre créateurs**

   Analyse les collaborations en identifiant les réalisateurs ayant travaillé avec un scénariste donné, et en comptant leurs projets communs.

5. **Genres populaires et qualitatifs**

   Met en évidence les genres combinant un volume significatif de films et une note moyenne élevée, garantissant une analyse statistiquement pertinente.

6. **Évolution de carrière par décennie**

   Étudie la productivité et la qualité d'un artiste au fil du temps en regroupant ses films par décennies.

7. **Classement intra-genre**

Établit le top 3 des films pour chaque genre grâce aux fonctions de fenêtrage et au classement par partition.

8. **Carrières propulsées par un succès**

   Identifie les artistes dont la carrière a décollé après un film à fort succès, en comparant la production avant et après ce tournant.

9. **Auteurs complets prolifiques**

   Repère les créateurs à la fois réalisateurs et scénaristes, analyse leur productivité, leur qualité moyenne et leurs genres de prédilection.

---

4. Benchmark des performances

4.1 Methodologie

Le benchmark compare les temps d'execution des 9 requetes avec et sans index:

1. **Phase 1:** Suppression de tous les index, mesure des temps d'execution
2. **Phase 2:** Creation de 12 index optimises
3. **Phase 3:** Mesure des temps d'execution avec index
4. **Phase 4:** Calcul des gains de performance

Outil: Module Python `time` avec precision en millisecondes
Environnement: SQLite 3.x, Python 3.x
Dataset: ~2,100 films, ~9,400 personnes

4.2 Resultats du benchmark

Requete	Sans index (ms)	Avec index (ms)	Gain (%)
Q1 - Filmographie	45.32	12.18	73.1
Q2 - Top N films	38.76	8.94	76.9
Q3 - Multi-roles	0.05	0.04	20.0
Q4 - Collaborations	52.41	15.63	70.2
Q5 - Genres populaires	28.15	9.87	64.9
Q6 - Evolution carriere	41.29	13.52	67.3
Q7 - Top 3 par genre	89.64	31.28	65.1
Q8 - Carriere propulsee	125.38	48.76	61.1

Requete	Sans index (ms)	Avec index (ms)	Gain (%)
Q9 - Dir-Writer combo	67.82	22.45	66.9
GAIN MOYEN	**54.31**	**18.07**	**66.7%**

4.3 Analyse des resultats

Observations principales:

1. **Gain significatif:** Reduction moyenne de 66.7% du temps d'execution
2. **Meilleure amelioration:** Q2 (Top N films) avec 76.9% de gain
3. **Requetes les plus lentes:** Q8 (Carriere propulsee) avec CTEs complexes
4. **Impact minimal:** Q3 (Multi-roles) car table inexistante dans le dataset

Impact sur la taille:
- Taille avant indexation: 2.48 MB
- Taille apres indexation: 2.68 MB
- Augmentation: 0.20 MB (8%)

Conclusion: L'overhead de stockage (8%) est largement compense par le gain de performance (67%). Les index sont essentiels pour les requetes analytiques.

4.4 Analyse des plans d'execution

Exemple: Requete 2 (Top N films) SANS index

```
SCAN movies AS m
SCAN genres AS g
SEARCH ratings AS r USING INTEGER PRIMARY KEY (rowid=?)
USE TEMP B-TREE FOR ORDER BY
```

Exemple: Requete 2 (Top N films) AVEC index

```
SEARCH movies AS m USING INDEX idx_movies_year (startYear>? AND sta
rtYear<?)
SEARCH genres AS g USING INDEX idx_genres_mid (mid=?)
SEARCH ratings AS r USING INTEGER PRIMARY KEY (rowid=?)
USE TEMP B-TREE FOR ORDER BY
```

Interpretation:
- Passage de SCAN (parcours complet) a SEARCH (recherche ciblee)
- Utilisation des index sur startYear et mid pour filtrage rapide
- Reduction drastique du nombre de lignes examinees

---

5. Index crees et justifications

5.1 Liste des index

Index	Table	Colonne(s)	Type
idx_movies_year	movies	startYear	B-Tree
idx_movies_title	movies	primaryTitle	B-Tree
idx_persons_name	persons	primaryName	B-Tree
idx_genres_mid	genres	mid	B-Tree
idx_genres_genre	genres	genre	B-Tree
idx_ratings_mid	ratings	mid	B-Tree
idx_ratings_rating	ratings	averageRating	B-Tree
idx_ratings_votes	ratings	numVotes	B-Tree
idx_directors_mid	directors	mid	B-Tree
idx_directors_pid	directors	pid	B-Tree
idx_writers_mid	writers	mid	B-Tree
idx_writers_pid	writers	pid	B-Tree

5.2 Justifications detaillees

Une stratégie d’indexation ciblée a été mise en place pour optimiser les performances des requêtes SQL les plus fréquentes.

- **Movies** :

  Index sur l’année pour accélérer les filtres temporels, et sur le titre pour d’éventuelles recherches textuelles futures.

- **Persons** :

  Index sur le nom afin d’optimiser les recherches par artiste, très utilisées dans les requêtes.

- **Genres** :

Index sur l'identifiant du film pour accélérer les jointures, et sur le genre pour améliorer les filtres par catégorie.

- **Ratings** :

  Index sur l'identifiant du film pour les jointures, ainsi que sur la note moyenne et le nombre de votes pour optimiser les tris et filtres liés à la qualité et à la popularité.

- **Directors / Writers** :

  Index sur les identifiants de films et de personnes afin d'optimiser les requêtes croisant films et créateurs.

Principes appliqués

- Indexation systématique des clés étrangères
- Index sur les colonnes utilisées dans `WHERE` et `ORDER BY`
- Priorité aux colonnes à bonne sélectivité
- Compromis maîtrisé entre performances et coût (12 index pour ~8 % d'overhead)

Annexes

A. Commandes d'execution

```
install -r requirements.txt
python create_schema.py
python import_data.py
python queries.py
python benchmark.py
jupyter notebook data/exploration.ipynb
```

B. Dependances

```
pandas>=1.3.0
jupyter>=1.0.0
matplotlib>=3.4.0
sqlite3 (inclus avec Python)
```