LIVRABLE 1 - BDD

Max Bouillanne

December 14, 2025

1 Introduction

La phase 1 de ce projet permet de prendre en main les données fournies et d'effectuer un schéma relationnel normalisé 3NF, permettant de garantir un fonctionnement correct de notre base de données d'un point de vue relationnel et enfin d'optimiser l'implémentation de nos requêtes complexes via des index.

2 Déploiement de la base normalisée

Après une première partie de travail sur Jupyter, nous avons réalisé un schéma relationnel en forme normale 3NF, permettant d'éviter toute forme de redondance dans nos données à travers les multiples tables de la base. Cela permet de garantir l'intégrité des données, principalement issues des 2 tables principales que sont Movies et Persons.

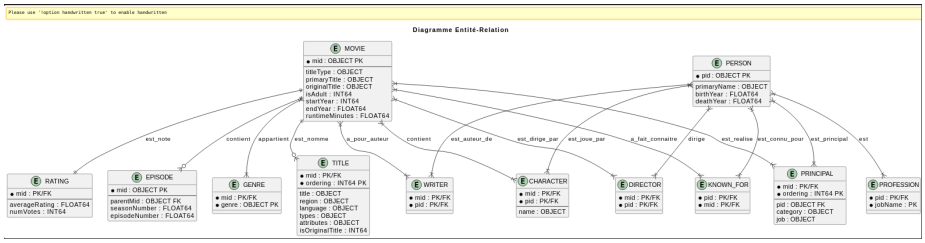

Figure 1: Schéma relationnel normalisé 3NF

L'implémentation du schéma à été réalisé à l'aide de plantUML, dont le script se trouve dans create_schema.puml.

3 Création de la base

La création de la base de données se fait exclusivement dans le script import_data.py. On y retrouve dans un premier temps une fonction nommé "create_base" qui permet de créer chacunes des tables, puis une autre fonction

nommé "import_data" qui permet de remplir ces tables.

L'insertion des données se fait en premier par les tables "MOVIE" et "PERSON" pour s'assurer que les dépendances sur les clés étrangères "mid" et "pid" soient bien respectés.
Ensuite, chaque table est à son tour ajoutée.

4 Implémentation de requêtes

Dans le script "queries.py", nous retrouvons l'implémentation des requêtes spécifiés à l'aide de fonctions SQL telle que "GROUP BY" et "HAVING".

Voici un exemple de requête qui à pour objectif de renvoyer les genres de films ayant plus de 50 films avec une note moyenne supérieure à 7 triés par leur note.

```
def Genre_Populaire():
    query = """
    SELECT
        g.genre as Genre,
        ROUND(AVG(r.averageRating), 2) as Note_Moyenne,
        COUNT(m.mid) as Nombre_de_Films
    FROM GENRE g
    JOIN MOVIE m ON g.mid = m.mid
    JOIN RATING r ON m.mid = r.mid
    WHERE m.titleType = 'movie'
    GROUP BY g.genre
    HAVING Note_Moyenne > 7.0
       AND Nombre_de_Films > 50
    ORDER BY Note_Moyenne DESC;
    """

    df = pd.read_sql_query(query, conn)
    print(df.head(20))
```

5 Indexation

IL est important de savoir quoi et pourquoi créer des index dans une base de données.
L'indexation permet de grandement accélerer la vitesse de recherche de données sur certaines colonnes. Ces index sont généralement créé sur des colonnes utilisées dans de conditions comme "WHERE" ou "JOIN" afin de les utiliser au maximums.

voici les index que j'ai créer :

```
conn.execute("CREATE INDEX IF NOT EXISTS idx_person_name ON
    PERSON(primaryName);")
conn.execute("CREATE INDEX IF NOT EXISTS idx_char_pid ON
    CHARACTER(pid);")
conn.execute("CREATE INDEX IF NOT EXISTS idx_char_mid ON
    CHARACTER(mid);")
conn.execute("CREATE INDEX IF NOT EXISTS idx_principals_mid
    ON PRINCIPAL(mid);")
conn.execute("CREATE INDEX IF NOT EXISTS idx_movie_year ON
    MOVIE(startYear);")
conn.execute("CREATE INDEX IF NOT EXISTS idx_rating_score ON
     RATING(averageRating);")
conn.execute("CREATE INDEX IF NOT EXISTS idx_genre_name ON
    GENRE(genre);")
```

Ces index sont créé selon les colonnes que j'utilise régulièrement dans mes conditions comme par exemple la colonnes "primaryName" de ma table "PERSON" qui se retrouve dans mes requêtes "Filmographie", "Collaboration" et "Evolution_Carriere".

6 Comparaison des résultats

Voici le tableau contenant le temps d'exécution de mes requêtes :

Requêtes	Sans index (ms)	Avec index (ms)	Gain (%)
Filmographie	2590.8	2.02	99.92 %
Top_Film	72.46	74.91	-3.38 %
Multi_Role	4741.55	4515.13	4.78 %
Collaboration	2521.92	2.15	99.91 %
Genre_Populaire	175.12	758.49	-333.13 %
Evolution_Carriere	2475.06	1.08	99.96 %
Meilleur_Film_Par_Genre	97.15	769.92	-692.51 %
Carriere_Propulse	2962.57	2981.73	-0.65 %
Derniere	0.39	0.38	2.56 %

Figure 2: Tableau des temps d'exécutions des requêtes (en ms)

On retrouve un gain de temps en % conséquent, preuve de l'utilité des index, mais aussi un gain négatif (ralentissement) pour certaines d'entres elles. L'utilisation d'un index permet d'accélerer la recherche d'information mais nécessite de faire des "sauts" entre l'index et la table. Ces sauts peuvent finir par être plus lents que de lire la table d'un seul coup, ce qui explique les gains négatifs.

Un impact direct de la création d'index est le coup en mêmoire qu'ils génèrent.

En effet la taille de la base de données avant la création d'index est de 718.51 Mo, après l'indexation, elle est de 845.39 Mo, ce qui représente une augmentation de 126.889 Mo.

7 Conclusion

Cette première phase a mis en évidence l'importance de la structuration et de l'optimisation d'une base de données et la robustesse d'une forme 3NF.
Couplée à la création d'index, les requêtes peuvent SQL permettent à cette base de répondre quasiment instantanément.

Cependant, cette étude met également en évidence le compromis entre vitesse d'exécution et espace disque. L'augmentation de la taille de la base montre qu'il est nécessaire de choisir les index créés afin qu'ils soient le plus efficaces possible.