Rapport : CineNet

Théo Raoul et Mathusan Selvakumar

27 Mai 2024

1 Introduction

Pour ce projet de base de données, nous avons opté pour l'utilisation de datasets réels afin de créer une base de données sur les films, acteurs, réalisateurs, événements et studios. Cette approche permet de concevoir une base de données plus authentique et fidèle à la réalité. Cependant, certaines interactions entre utilisateurs, les projections lors des événements et d'autres éléments ont été générés dynamiquement à l'aide de Python.

Afin de garantir la qualité et la reproductibilité des données, nous avons utilisé une même graine de génération pour l'ensemble du processus.

2 Modèle Entité-Association

Le diagramme Entité-Association pour CineNet est fourni en un seul fichier PDF séparemment. Par ailleurs, les noms d'associations sont omis pour des raisons de lisibilité.

3 Schéma Relationnel

UserRoles (type, name, description)

Users (id, last_name, first_name, username, email, password_hashed, birth_date, role_type)
Users[role_type] ⊆ UserRoles[type]

Countries (country_code, name)

Cities (city_code, country_code, name)
Cities[country_code] ⊆ Countries[country_code]

UserLocation (user_id, city_code)
UserLocation[user_id] ⊆ Users[id], UserLocation[city_code] ⊆ Cities[city_code]

Friendship (initiator_id, recipient_id, date)
Friendship[initiator_id] ⊆ Users[id], Friendship[recipient_id] ⊆ Users[id]

Following (follower_id, followed_id, date)
Following[follower_id] ⊆ Users[id], Following[followed_id] ⊆ Users[id]

Categories (id, name, description)

Posts (id, user_id, date, content, parent_post_id, category_id)
Posts[user_id] ⊆ Users[id], Posts[parent_post_id] ⊆ Posts[id], Posts[category_id] ⊆ Categories[id]

Tags (id, name)

PostTags (tag_id, post_id)
PostTags[tag_id] ⊆ Tags[id], PostTags[post_id] ⊆ Posts[id]

Reactions (user_id, post_id, emoji_unicode)
Reactions[user_id] ⊆ Users[id], Reactions[post_id] ⊆ Posts[id]

Events (id, name, date, city_code, organizer_id, capacity, ticket_price)
Events[city_code] ⊆ Cities[city_code], Events[organizer_id] ⊆ Users[id]

Participation (user_id, event_id, type_participation)
Participation[user_id] ⊆ Users[id], Participation[event_id] ⊆ Events[id]

Genres (id, name, parent_genre_id)
Genres[parent_genre_id] ⊆ Genres[id]

Studios (id, name)

Movies (id, title, duration, release_date)

MovieGenres (movie_id, genre_id)
MovieGenres[movie_id] ⊆ Movies[id], MovieGenres[genre_id] ⊆ Genres[id]

MovieStudios (studio_id, movie_id)
MovieStudios[studio_id] ⊆ Studios[id], MovieStudios[movie_id] ⊆ Movies[id]

People (id, last_name, first_name, birth_date)

PeopleRoles (id, name)

MovieCollaborators (people_id, movie_id, role_id)
MovieCollaborators[people_id] ⊆ People[id], MovieCollaborators[movie_id] ⊆ Movies[id], MovieCollaborators[role_id] ⊆ PeopleRoles[id]

Screenings (event_id, movie_id, screening_time)
Screenings[event_id] ⊆ Events[id], Screenings[movie_id] ⊆ Movies[id]

UserEventRatings (user_id, event_id, rating)
UserEventRatings[user_id] ⊆ Users[id], UserEventRatings[event_id] ⊆ Events[id]

UserMovieRatings (user_id, movie_id, rating)
UserMovieRatings[user_id] ⊆ Users[id], UserMovieRatings[movie_id] ⊆ Movies[id]

MovieRecommendation (user_id, movie_id, score_recommendation)
MovieRecommendation[user_id] ⊆ Users[id], MovieRecommendation[movie_id] ⊆ Movies[id]

CompletedEventRecommendation (user_id, event_id, score_recommendation)
CompletedEventRecommendation[user_id] ⊆ Users[id], CompletedEventRecommendation[event_id] ⊆ Events[id]

ScheduledEventRecommendation (user_id, event_id, score_recommendation)
ScheduledEventRecommendation[user_id] ⊆ Users[id], ScheduledEventRecommendation[event_id] ⊆ Events[id]

PostRecommendation (user_id, post_id, score_recommendation)
PostRecommendation[user_id] ⊆ Users[id], PostRecommendation[post_id] ⊆ Posts[id]

4 Les Choix et Limites de la Modélisation

Pour le projet CineNet, nous avons opté pour une base de données centrée exclusivement sur les films, excluant les séries télévisées. Cette décision vise à simplifier la modélisation et la gestion de la base de données. Les films sont des entités relativement simples à modéliser et suffisamment nombreux pour permettre une analyse pertinente. De plus, ils sont des objets culturels très populaires, souvent utilisés comme références dans les discussions sur le cinéma. Nous avons donc choisi de nous concentrer uniquement sur les films pour ce projet.

Table `UserRoles`

Bien que la table `UserRoles` puisse sembler peu intéressante à première vue, elle a été conçue pour permettre des permissions variées selon les différents types d'utilisateurs. Un utilisateur lambda a un rôle moins important qu'un studio de production, par exemple. L'attribut `permission` permettrait d'implémenter des fonctionnalités plus avancées selon le type d'utilisateur.

Tables `Cities`, `Countries` et `UserLocations`

Les utilisateurs peuvent indiquer leur ville de résidence, et les tables `Cities`, `Countries` et `UserLocations` facilitent la localisation des utilisateurs. Connaître la ville de résidence permet de créer des requêtes intéressantes, telles que la recherche d'événements ayant lieu dans une ville spécifique.

Table `Friendships`

Un utilisateur peut être ami avec un autre, mais une seule direction est créée dans la table `Friendships`. Cela peut sembler être une limitation, car il faut vérifier les deux sens pour confirmer une amitié. Toutefois, cette approche permet de différencier celui qui a initié la demande et la date de la demande.

Table `Following`

Un utilisateur peut suivre un autre utilisateur sans que ce dernier ne le suive en retour, permettant ainsi une asymétrie dans les relations de suivi.

Table `Categories`

Les catégories de forums sont limitées au premier niveau, à l'instar des subreddits sur Reddit. Il n'est pas possible d'avoir un sous-forum dans un autre, ce qui simplifie la structure et l'utilisation.

Table `Posts`

Une publication est signée par un utilisateur (l'auteur) qui possède un nom d'utilisateur et un mot de passe crypté pour une connexion sécurisée. Une publication peut ou non être une réponse à une autre publication et peut ou non appartenir à une catégorie. Ne pas avoir de catégorie signifie que la publication est générale.

Système de Mots-Clés

Grâce aux tables `Tags` et `PostTags`, il est possible de faire des recherches rapides en utilisant des titres de films, des noms d'acteurs ou des genres cinématographiques comme mots-clés. Les recherches plus avancées, telles que l'affichage des sous-genres pour un genre donné, doivent être effectuées au niveau de l'application en utilisant les tags et en comparant avec la table des genres, puis en récupérant récursivement les sous-genres.

Réactions aux Publications

Les utilisateurs peuvent réagir aux publications avec des emojis représentant différentes appréciations.

Table `Events`

Les utilisateurs peuvent indiquer leur intérêt ou leur participation effective à un événement. Les événements peuvent avoir trois états : `Scheduled` (Prévu), `Completed` (Fini), ou `Cancelled` (Annulé).

5 Requêtes

5.1 Requête SQL pour la Récupération des Utilisateurs Participant à un Événement Planifié

Cette requête SQL implique au moins trois tables pour récupérer des événements. Elle sélectionne le nom d'utilisateur des utilisateurs participant à un événement planifié dans une ville et un pays spécifiques où ils sont situés. La requête reçoit deux paramètres : le nom de l'événement et le nom de la ville.

```
SELECT
    U.username,
    E.date
FROM
    Users U
    JOIN UserLocations UL ON U.id = UL.user_id
    JOIN Cities C ON UL.city_code = C.city_code
    JOIN Countries CO ON C.country_code = CO.country_code
    JOIN Participation P ON U.id = P.user_id
    JOIN Events E ON P.event_id = E.id
WHERE
```

```
    E.status = 'Scheduled'
    AND P.type_participation = 'Participating'
    AND E.name = :eventname
    AND CO.name = :countryname
;
```

5.2 Requête SQL pour la Récupération des Utilisateurs Suivant un Utilisateur Spécifique

Cette requête SQL implique une auto-jointure de la table `Users` pour récupérer les utilisateurs qui suivent un autre utilisateur. Elle sélectionne le nom d'utilisateur des utilisateurs qui suivent un utilisateur spécifique. La requête reçoit un paramètre : le nom d'utilisateur.

```
SELECT
    A.username AS Follower,
    B.username AS Followed
FROM
    Users A
    JOIN FOLLOWING F ON A.id = F.follower_id
    JOIN Users B ON F.followed_id = B.id
WHERE
    B.username = :username;
```

5.3 Requête SQL pour la Récupération des Utilisateurs Suivant un Nombre Minimum d'Utilisateurs

Cette requête SQL implique une sous-requête corrélée pour récupérer les utilisateurs qui suivent un autre utilisateur. Elle sélectionne le nom d'utilisateur des utilisateurs qui suivent un nombre minimum d'autres utilisateurs. La requête reçoit un paramètre : le nombre minimum d'utilisateurs suivis.

```
SELECT
    username,
(
        SELECT
            COUNT(*)
        FROM
            FOLLOWING F
        WHERE
            F.follower_id = U.id) AS Following_Count
FROM
    Users U
WHERE (
    SELECT
        COUNT(*)
    FROM
        FOLLOWING F
    WHERE
        F.follower_id = U.id) >= :minfollowingcount;
```

5.4 Requête SQL pour le Calcul de la Popularité Moyenne des Utilisateurs Suivis par Pays

Cette requête SQL implique une sous-requête dans le FROM. Elle sélectionne le nom du pays et la popularité moyenne des utilisateurs suivis par pays.

```
SELECT
    CO.name AS Country_Name,
    AVG(PopularityScores.Popularity) AS Average_Popularity
FROM
    Countries AS CO
    JOIN (
```

```
        SELECT
            COUNT(*) AS Popularity,
            UL.city_code
        FROM
            FOLLOWING AS F
            JOIN Users AS U ON F.followed_id = U.id
            JOIN UserLocations AS UL ON U.id = UL.user_id
        GROUP BY
            UL.city_code) AS PopularityScores ON CO.country_code =(
        SELECT
            country_code
        FROM
            Cities
        WHERE
            city_code = PopularityScores.city_code)
GROUP BY
    CO.name;
```

5.5 Requête SQL pour la Récupération des Utilisateurs Postant Après une Date Spécifique

Cette requête SQL implique une sous-requête dans le WHERE. Elle sélectionne les nom d'utilisateurs qui ont posté après une date spécifique.

```
SELECT U.username
FROM Users U
WHERE U.id IN (SELECT user_id FROM Posts WHERE date > :date)
```

6 Conclusion