# Rapport NoiseBook

**VYSHKA Tedi**
**RODRIGUEZ Lucas**

Ce rapport présente la conception de la base de données pour NoiseBook, un nouveau réseau social dédié à la musique. Dans ce rapport, nous expliquerons la manière dont la base de données a été conçue pour prendre en charge les fonctionnalités du réseau social. Nous discuterons également des requêtes que nous avons choisi d'implémenter ainsi que les choix que nous avons fait pour le calcul de l'indice de recommandation.

## Modélisation

La conception initiale d'un réseau social repose sur l'implémentation des utilisateurs, et celle de NoiseBook est unique.
Tous les utilisateurs devraient être en capacité d'écouter, découvrir, critiquer ou même discuter de n'importe quelle musique ou artiste, peu importe leur existence en tant qu'utilisateur du réseau social. La solution que nous avons décidé d'implémenter est de créer deux entités *musician* et *music_group* qui contiennent les données relatives aux musiciens et groupes de musiques existants, ainsi qu'une table *track* contenant leurs musiques respectives.
Toute personne souhaitant s'inscrire sur le réseau social sera considérée comme un utilisateur, mais fera partie en réalité de l'un de ces trois groupes:

- *People*: un utilisateur humain
- *Groups*: un utilisateur représentant un groupe de musique
- *Organizers*: un utilisateur représentant une entité officielle en capacité d'organiser un concert (association, salle de concert, …)

Pour réaliser cette implémentation nous avons fait un héritage disjoint total, permettant le moins de redondance possible.
L'entité *people* représente donc un utilisateur humain qui, s'il s'agit de la même personne, pourrait être relié par une relation à l'entité *musician*. Cette conception permet donc de faire un lien entre un utilisateur de NoiseBook et les informations d'un artiste (ses musiques, concerts, appartenance à un groupe…) sans le désavantage de ne pas avoir de table *musician* dédiée.
L'entité *group* quant à elle représente seulement les groupes de musiques et est donc directement reliée à l'entité *music_group* s'ils sont amenés à créer un compte sur le réseau social.
L'entité *organizers* constitue tout le reste des utilisateurs pouvant s'inscrire sur NoiseBook, plus précisément des entités officielles pouvant organiser des concerts.

Chaque utilisateur a donc la possibilité d'avoir 10 playlists contenant au maximum 20 références à l'entité *track*.
Les entités *track* et *concert* peuvent avoir un genre et/ou un sous-genre.
Dans l'organisation de ces genres musicaux, nous avons adopté la structure proposée par « Discogs ». Cette structure envisage l'existence de genres musicaux, qui à leur tour peuvent engendrer des sous-genres. Dans notre base de données, ces sous-genres sont désignés par l'entité *sub_genre*. Ce choix stratégique s'est imposé pour réguler la multiplicité des styles.

Afin que les utilisateurs aient des interactions sur le réseau social nous avons implémenté la possibilité de faire un *post* et d'y associer un ou plusieurs *media*, qui correspond à un lien vers le stockage de l'image sur un serveur.
En plus de ça, par l'entité *review* les utilisateurs sont en capacité de laisser un avis unique sur les entités *track*, *group*, *place* et *finished_concert*. Ces avis seront accompagnés d'un commentaire et d'une note.

Une façon supplémentaire de créer des liens entre les utilisateurs se base sur le mécanisme de *follows*. Un utilisateur a la possibilité de suivre n'importe quel autre utilisateur du réseau social, mais ce mécanisme n'est pas forcément réciproque. Un mécanisme alternatif est celui des *friendship* qui nécessite à l'un des deux utilisateurs d'accepter la demande afin qu'ils obtiennent le statut d'amis.

Notre modélisation nous permet aussi de relier de multiples informations par l'utilisation de *tag*. Chaque relation entre l'entité *tag* aux entités *post*, *review*, *place*, *genre*, *sub_genre*, *music_group*, *future_concert* et *finished_concert* nous permet de classifier et relier les informations. Ces tags pourront notamment être utilisés par la suite dans l'algorithme de recommandation d'évènements.

## Schéma relationnel

### Users

users(user_id, username, password, email, bio)
people(person_id, user_id*, first_name, last_name, birth_date, sex)
groups(group_id, user_id*, music_group_id*, group_name)
organizers(organizer_id, user_id*, organizer_name, location)

### Relationships

follows(follower_id*, followed_id*)
friendship(user1_id*, user2_id*)

### Music

track(track_id, title)
music_group(music_group_id, group_name)
musician(musician_id, artist_name)
link_people_musician(person_id* , musician_id*)
link_musician_music_group(musician_id*, music_group_id*)
musician_plays_track(musician_id*, track_id*)
music_group_plays_track(music_group_id*, track_id*)

### Playlists

playlist(playlist_id, playlist_name, description, user_id*)
playlist_track(playlist_id*, track_id*)

### Reviews

review(review_id, review_timestamp, review_grade, review_comment)
group_review(review_id*, user_id*, group_id*)
track_review(review_id*, user_id*, track_id*)
place_review(review_id*, user_id*, place_id*)
concert_review(review_id*, user_id*, concert_id*)

### Posts

post(post_id, post_timestamp, post_content, user_id*)
media(media_id, media_link, post_id*)

**Concerts**

place(place_id, place_name, address, city, country, exterior, max_capacity)
future_concert(concert_id,concert_name,concert_date,start_time,ticket_price,children_allowed,place_id*)
finished_concert(concert_id, concert_name, concert_date, start_time, ticket_price, children_allowed, attendance, place_id*)
future_concert_musicians_lineup(musician_id*, concert_id*)
future_concert_music_group_lineup(music_group_id*, concert_id*)
finished_concert_musicians_lineup(musician_id*, concert_id*)
finished_concert_music_group_lineup(music_group_id*, concert_id*)
organizers_announce_concert(organizer_id*, concert_id*)
user_attends_concert(user_id*, concert_id*)

**Tags**

tag(tag_id, tag_content)
post_tag(tag_id*, post_id*)
review_tag(tag_id*, review_id*)
place_tag(tag_id*, place_id*)
genre_tag(tag_id*, genre_id*)
sub_genre_tag(tag_id*, sub_genre_id*)
music_group_tag(tag_id*, group_id*)
future_concert_tag(tag_id*, concert_id*)
finished_concert_tag(tag_id*, concert_id*)

**Genres**

genre(genre_id, genre_title, genre_description)
sub_genre(sub_genre_id, sub_genre_title, sub_genre_description, parent_genre*)
future_concert_sub_genre(concert_id*, sub_genre_id*)
finished_concert_genre(concert_id*,genre_id*)
finished_concert_sub_genre(concert_id*,sub_genre_id*)
future_concert_genre(concert_id*, genre_id*)
track_genre(track_id*, genre_id*)
track_sub_genre(track_id*, sub_genre_id*)

## Clés étrangères

- user_id, follower_id, followed_id, user1_id, user2_id dans users (user_id)
- group_id dans groups (group_id)
- music_group_id dans music_group (music_group_id)
- parent_genre, genre_id dans genre (genre_id)
- sub_genre_id dans sub_genre (sub_genre_id)
- track_id dans track (track_id)
- person_id dans people (person_id)
- musician_id dans musician (musician_id)
- playlist_id dans playlist (playlist_id)
- place_id dans place (place_id)
- concert_id dans future_concert (concert_id)
- organizer_id dans organizers (organizer_id)
- review_id dans review (review_id)
- post_id dans post (post_id)
- tag_id dans tag (tag_id)
- music_group_tag -> group_id dans music_group (music_group_id)
- finished_concert_tag -> concert_id dans finished_concert (concert_id)

## Contraintes externes

- deux utilisateurs ne peuvent pas avoir le même *username*
- les emails doivent être bien formatés (@ et .)
- un utilisateur doit avoir au moins 13 ans
- le sexe d'un utilisateur doit appartenir à l'ensemble ('M','F,'O')
- un utilisateur ne peut pas se follow lui même
- un utilisateur ne peut pas être amis avec lui-même
- la capacité maximale d'un lieu doit être supérieure à 0
- le prix d'un ticket d'un concert doit être supérieur ou égal à 0
- le nombre de participants d'un concert terminé doit être supérieur ou égal à 0
- les notes d'un avis doivent être comprises entre 0 et 10
- un utilisateur peut créer au plus 10 playlists
- une playlist contient au maximum 20 musiques
- un avis concerne soit un groupe, une musique, un lieu ou un concert
- un utilisateur ne peut pas déposer deux avis sur la même entité
- le nombre de participants d'un concert terminé doit être inférieur à la capacité maximale du lieu

Certaines contraintes externes ont été implémentées à l'aide des triggers de PostgreSQL.

## Description des requêtes

Pour notre base de données nous avons implémenté les requêtes:

- qui renvoie les 5 meilleurs titres avec la note moyenne la plus élevée, avec au moins 5 commentaires, et leur note maximale.
- qui renvoie les utilisateurs qui se suivent mutuellements
- qui renvoie les 3 villes qui accueillent le plus grand nombre de concerts
- qui renvoie les groupes de musique qui ont un nombre de followers supérieur à la moyenne
- qui renvoie les lieux qui n'ont pas encore accueilli de concert
- qui renvoie les groupes de musique qui se sont produits dans plus de trois villes différentes
- qui renvoie le groupe de musique dont le nombre moyen de spectateurs par concert est le plus élevé
- qui renvoie tous les groupes de musique et les lieux où ils se sont produits y compris les groupes qui ne se sont pas encore produits
- qui renvoie les 3 groupes de musique les plus joués dans les concerts avec la capacité maximale chaque mois en 2023
- qui renvoie le nombre de concerts organisés par chaque organisateur
- qui renvoie tous les groupes de musique qui ont organisé plus de 10 concerts
- qui renvoie la série de follows depuis un utilisateur
- qui renvoie les utilisateurs qui ont donné plus de 5 commentaires sur des concerts terminés, avec une note moyenne supérieure à 4
- qui renvoie le nombre de membres pour chaque groupe
- qui renvoie tous les concerts à venir avec les genres et sous-genres correspondants
- qui renvoie le concert dont le prix du billet est le plus bas
- qui renvoie les musiques classées selon la note moyenne la plus élevée, leur genre
- qui renvoie les musiques et les trie en fonction de leur note moyenne d’avis
- qui renvoie le nombre de tags communs entre les intérêts de l'utilisateur et les tags associés à un concert à venir
- qui identifie les concerts à venir dont le genre correspond aux genres des chansons que l'utilisateur a ajoutées à ses playlists

## Un algorithme de recommandation d’événements

Dans le contexte de la recommandation d'événements, la technique du filtrage collaboratif présente des avantages significatifs. Cette technique repose sur l'idée que si deux utilisateurs ont des comportements similaires dans le passé, ils auront probablement des préférences similaires dans le futur.

Pour améliorer l'efficacité des systèmes de recommandation, il est nécessaire de prendre en compte des données supplémentaires telles que les tags, les reviews et les playlists créées par l'utilisateur. Ces données peuvent révéler des tendances et des préférences spécifiques, ce qui permet de proposer des recommandations plus précises.

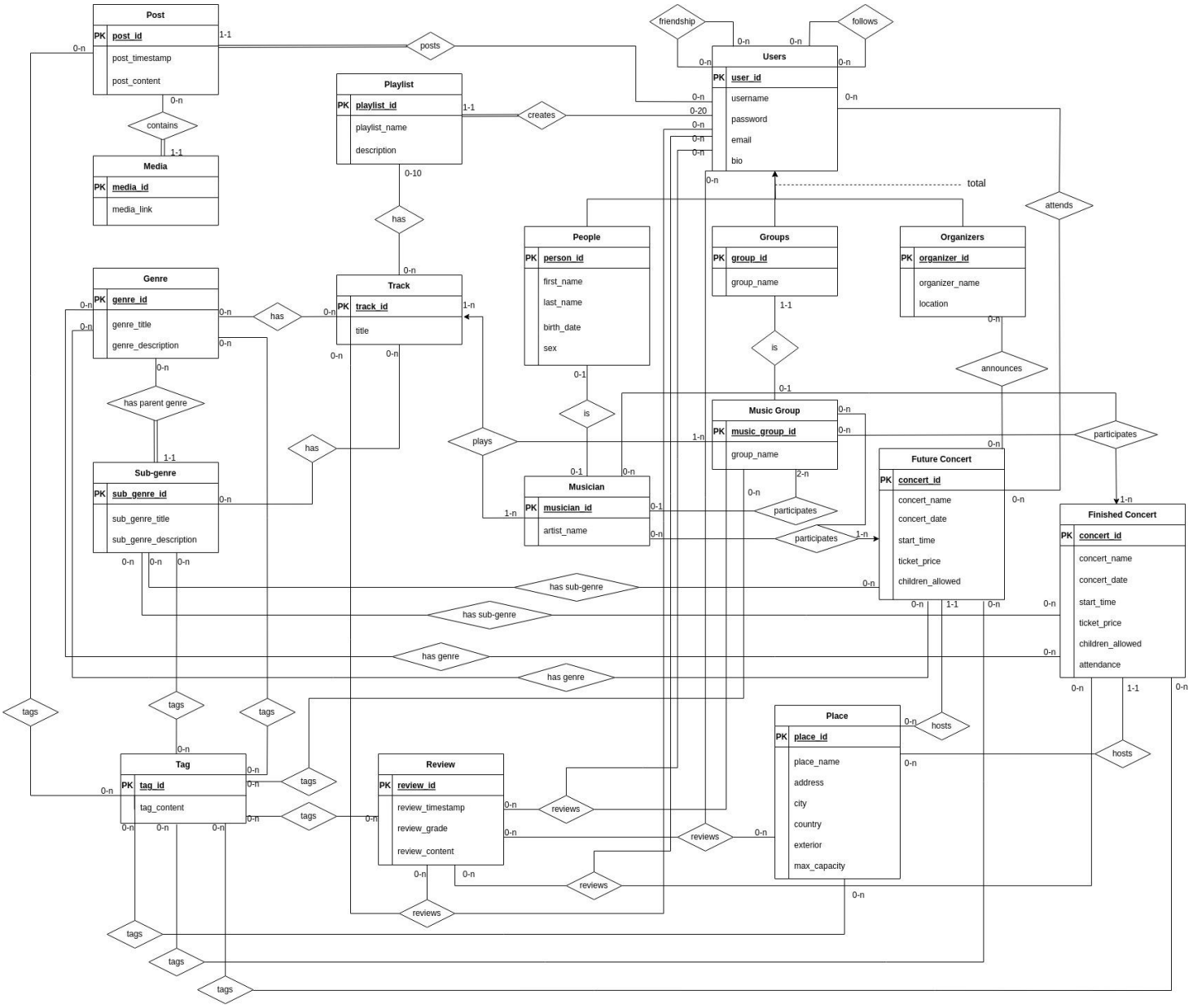
Ainsi, notre proposition de requête serait d'abord basée sur l'analyse des tags des reviews et des posts contenant des médias. Les tags liés à des genres spécifiques, des artistes, des groupes ou des morceaux pourraient être utilisés pour rechercher des événements avec des caractéristiques similaires. Par exemple, si un utilisateur a fréquemment tagué ou reviewé des concerts de rock, l'algorithme pourrait recommander des événements à venir avec des groupes de rock.

L'indice de recommandation pourrait alors être calculé en fonction du nombre de correspondances entre les tags d'un utilisateur et ceux associés à un événement. Plus le nombre de correspondances est élevé, plus l'indice de recommandation serait élevé.

Cette méthode a plusieurs points forts. Tout d'abord, elle permet une personnalisation fine des recommandations en fonction des préférences spécifiques de chaque utilisateur. De plus, elle peut s'adapter aux changements dans les goûts musicaux de l'utilisateur.

Cependant, cette méthode présente également des limites. Tout d'abord, elle dépend beaucoup de la qualité et de la précision des tags et des reviews.

Pour améliorer ce système, il serait intéressant d'intégrer un système de feedback de l'utilisateur, afin d'affiner les recommandations au fil du temps. De plus, il serait intéressant d'intégrer d'autres facteurs tels que la proximité géographique et les préférences des amis/followers de l'utilisateur.




**VYSHKA Tedi 22011242**

**RODRIGUEZ Lucas 22002335**