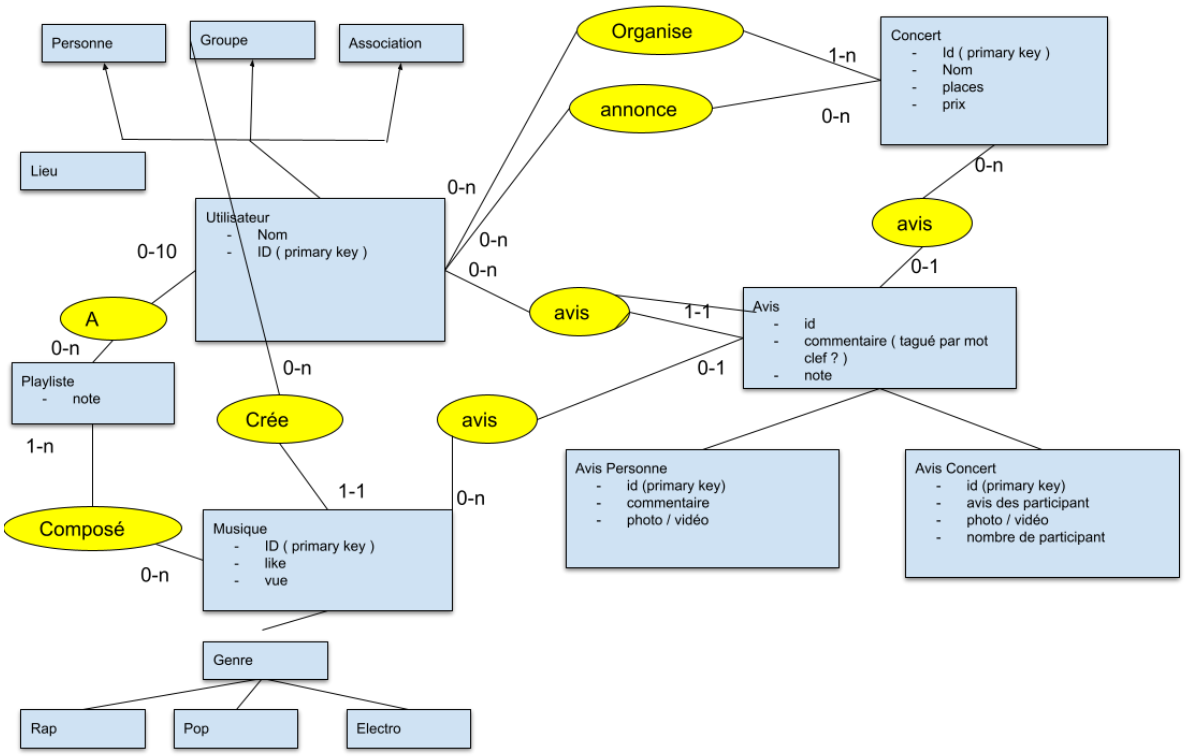
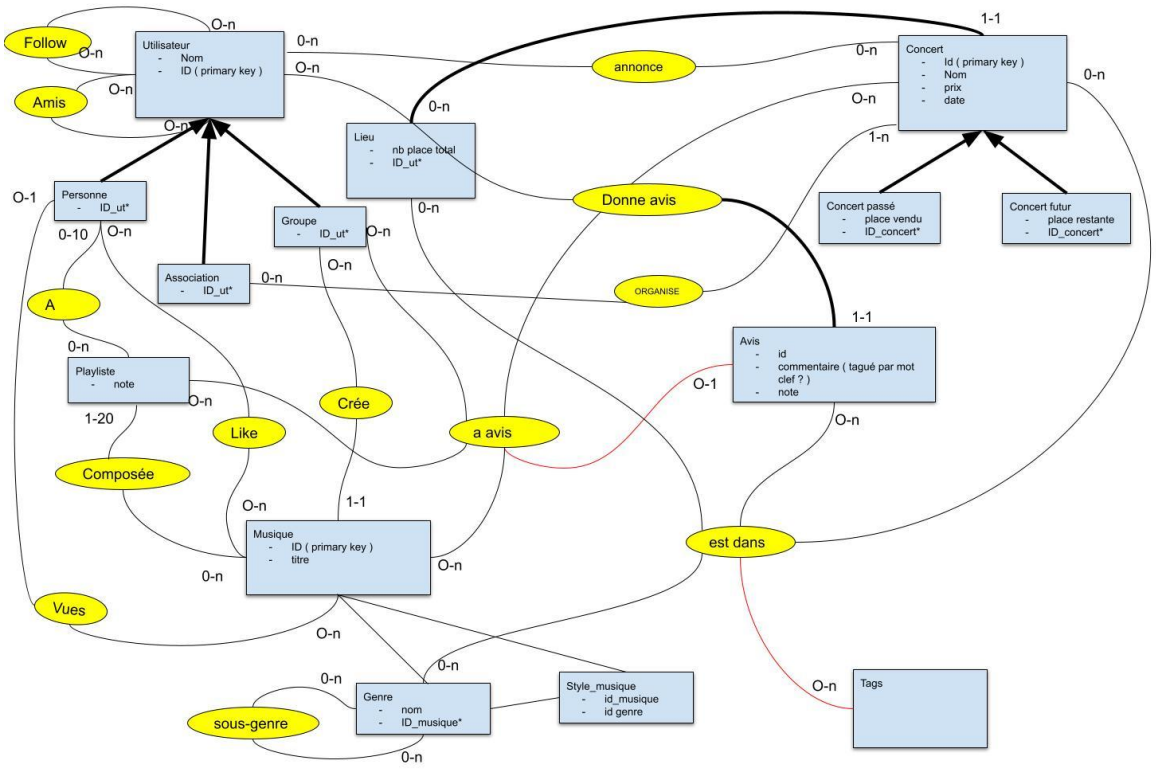
Projet NoteBook: Bases de données

I) Modélisation Conceptuelle

Pour la modélisation conceptuelle, dans notre premier schéma nous avons commencé par placer les principales tables (Utilisateur / Musique / Avis / Concert) et à partir de là nous avons créer les “sous-tables” qui vont avec (Personne / Groupe / Genre) qu’on a relié avec des relations (Playlist est composé de musique , etc).



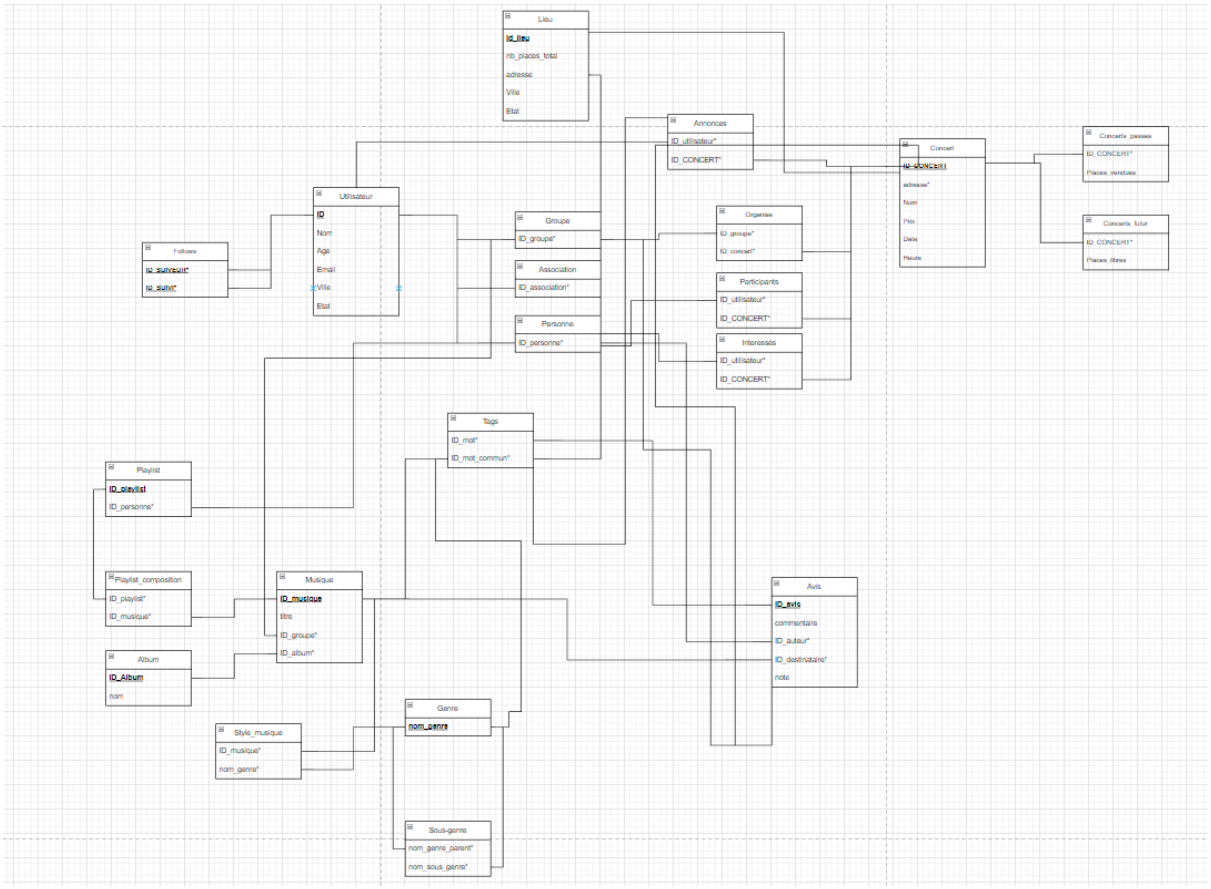
Après la première soutenance (présentation schéma), nous avons eu pas mal de retours et de changements à faire sur notre schéma. Nous avons ajouté des tables “concert passé” et “concert futur” qui vont se créer en fonction de la date. Retirer les tables “avis concert” et “avis personne” pour les remplacer par des relations entre avis et concert ou personne. De même pour Genre et sous genre nous avons retiré les sous-genre pop / rap / électro pour les remplacer par une relation sous-genre.
Pour expliquer quelque unes des relations que nous avons choisis, on a :
“Annonce” qui permet à un utilisateur d’annoncer un concert.
“Organise” qui permet à un utilisateur d’organiser un concert.
“a avis” qui montre qu’une musique / un groupe / une playlist / un lieu / un concert peut avoir un avis de la table “Avis”
“est dans” sera pour dire que le mot dans “tags” (qui est lui-même récupéré dans le commentaire de l’avis) peut se trouver dans le genre / le lieu / le concert.



Pour finir à partir de ce schéma nous avons recréé un énième schéma nous aidant à créer les tables qu'on va faire et se partager. Ce schéma sera le récapitulatif des tables que nous avons dans notre base de données avec des liens (clé primaire, clé étrangère). Nous avons créé ce schéma à partir du schéma 2, et y avons ajoutés quelque fonctionnalités et modifications comme :

- un groupe organise un concert
- une association annonce le concert
- une personne peut participer ou être intéressé par un concert
- un utilisateur u1 qui follows un utilisateur u2 sont considéré comme ami
- lieu qui a maintenant un nombre de place
- un album qui est composé de musique

Par souci de simplicité, nous avons décidé de négliger toute la partie du projet qui consiste à concevoir des tags et leurs utilisations dans chacune des tables.



(si le schéma est flou voici un lien pour directement le voir : https://drive.google.com/file/d/1-ex4Ig6Pw3Sl32ei7DrAT7H63_GfohGW/view?usp=sharing)

2) Requêtes implémentées

Dans le cadre de notre projet, nous avons réalisé une vingtaine de requêtes afin de permettre de manipuler nos tables et faire vivre notre réseau social. Nous allons donc vous présenter chacune de ces requêtes et leur utilisation:

Une requête permettant de récupérer le nom du lieu et le nom du groupe organisateur pour chaque concert, pour faire cela nous faisons des jointures SQL sur plusieurs tables notamment Lieu,Organise,Groupes et Utilisateurs et Concerts. La commande pour lancer cette requête est recherche_concerts.

Une requête permettant de rechercher les amis d'un utilisateur. Cette requête utilise une jointure réflexive sur la table follows on détermine les personnes qui suivent l'utilisateur et que l'utilisateur suit également. La commande de cette requête est recherche_amis.

Une requête permettant de connaître les concerts qui ont plus de n participants où n est un nombre. Cette requête avec sous-requête corrélée

détermine les concerts en comptant et comparant le nombre de participants de chaque concert dans la table Participants. La commande de cette requête est recherche_nb_concerts_participants(n).

Une requête permettant de déterminer le nombre de concerts auxquels chaque utilisateur a participé.Pour obtenir cela, la requête stocke dans une table temporaire appelée “PC” les paires distinctes de participants et concerts et fait une jointure de cette table avec les tables “Personnes” et “Utilisateurs”. La table résultante est ensuite regroupée par le nom de l'utilisateur et le nombre de concerts auxquels il a participé est compté pour chaque groupe. Le résultat du comptage est ensuite trié par ordre décroissant, de sorte que les utilisateurs ayant participé au plus grand nombre de concerts apparaissent en haut de la table. La commande est recherche_nb_concerts.

Une requête permettant de rechercher les utilisateurs qui suivent au moins deux autres utilisateurs. Elle commence par sélectionner tous les utilisateurs de la table "Utilisateurs" et filtre ensuite ceux qui ont plus de deux followers. Pour cela, la requête utilise une sous-requête qui sélectionne les ID des followers sur la table "Follows", regroupe ces ID par le nombre de followers, et filtre ensuite ceux qui ont deux followers ou plus avec GROUP BY et HAVING.La commande est recherche_suiveurs.

Une requête permettant de rechercher les sous-genres de chaque genre. Pour obtenir cela, nous réalisons une jointure gauche sur la table Genre et sous-genre et renvoyons les lignes pour lesquelles le nom du genre est également celui du genre parent du sous-genre. La commande est recherche_sous_genres.

Une requête permettant de rechercher les utilisateurs qui ont assisté à plus de cinq concerts dont le prix moyen est supérieur à vingt. La requête commence par sélectionner les paires distinctes de participant et de concert à partir des tables "Participants" et "Concerts", respectivement, et les stocke dans une table temporaire. Ensuite, elle utilise une jointure pour relier cette table temporaire avec les tables "Personnes" et "Utilisateurs" pour obtenir les noms et les adresses e-mail des utilisateurs qui ont assisté à ces concerts.
La requête regroupe ensuite les résultats par nom d'utilisateur et adresse mail, et utilise la fonction COUNT pour compter le nombre de concerts auxquels chaque utilisateur a assisté, et la fonction AVG pour calculer le prix moyen de ces concerts. La requête filtre ensuite les résultats pour ne conserver que les utilisateurs qui ont assisté à plus de cinq concerts et dont le prix moyen des concerts est supérieur à vingt. La commande de cette requête est recherche_utilisateurs.

Une requête permettant de rechercher les groupes ayant une moyenne des notes supérieure à 3.5. Pour obtenir cela, La requête commence par sélectionner les colonnes correspondantes de la table "Avis_groupes", de la table "Avis" et de la

table "Groupes". Ensuite, elle utilise une jointure interne pour lier ces tables en utilisant les identifiants d'avis et de groupe. Cette jointure permet d'associer les avis des utilisateurs à chaque groupe musical. La requête regroupe ensuite les résultats par identifiant de groupe et utilise la fonction AVG pour calculer la note moyenne associée à chaque groupe. La requête filtre ensuite les résultats pour ne conserver que les groupes ayant une note moyenne supérieure à 3.5. Les résultats finaux contiennent l'identifiant de chaque groupe et sa note moyenne. La commande est recherche_groupes.

Une requête qui recherche la note moyenne maximale donnée par chaque utilisateur. Cette requête crée une table temporaire appelée "max_notes_utilisateur" qui contient l'identifiant de chaque utilisateur et sa note maximale qu'elle sélectionne à partir de la table "Avis",elle utilise ensuite la fonction MAX pour obtenir la note maximale de chaque utilisateur et la fonction GROUP BY pour grouper les résultats par identifiant d'utilisateur. La requête utilise ensuite cette table temporaire pour calculer la moyenne de toutes les notes maximales. La fonction ROUND est utilisée pour arrondir le résultat à deux décimales. La commande est recherche_moyenne_max_notes.

Une requête permettant de lister les groupes musicaux les plus populaires pour chaque mois de l'année 2022. Elle crée trois tables temporaires: "mois","interet_concert" et "top_groupes".
La table "mois" est créée en utilisant la fonction generate_series() pour générer une série de dates couvrant tous les mois de l'année 2022. Cette table contient les dates de début et de fin de chaque mois.
La table "interet_concert" est créée en utiilisant une jointure entre les tables "Participitans" et "Concerts". Elle contient l'identifiant de chaque concert, l'identifiant de chaque personne ayant montré de l'intérêt pour ce concert, et le mois correspondant à la date du concert.
La table "top_groupes" est créée en utilisant une jointure entre la table "Organise" et la table "interet_concert". Elle contient l'identifiant de chaque groupe musical, le mois correspondant à la date du concert, le nombre de personnes ayant montré de l'intérêt pour chaque concert associé à ce groupe, et un classement des groupes en fonction du nombre d'intérêts. Le classement est effectué pour chaque mois en utilisant la fonction RANK(). La requête finale joint les tables "mois" et "top_groupes" en utilisant la plage de dates de début et de fin de chaque mois. Elle ne conserve que les groupes qui se classent dans les dix premiers pour chaque mois. Les résultats sont triés par ordre croissant de la date de début du mois, de la date de fin du mois, et du nombre d'intérêts pour chaque groupe. La commande est recherche_concerts_populaires.

Une requête donnant le groupe le plus populaire , c'est à dire celui qui a organisé le plus de concert, la requête affichera le groupe et le nombre de concert organisé grâce à des INNER JOIN dans les tables Groupes et Organise avec un COUNT du

nombre d’organisation que nous trions dans l’ordre décroissant et qu’on limite à 1 pour avoir seulement le plus grand.

Une requête nous donnant la moyenne des âges des utilisateurs grâce à la fonction AVG() de sql.

Une requête nous donnant le Concert le mieux noté, c'est-à-dire celui qui a la plus grosse moyenne dans les avis. On a utilisé AVG() pour obtenir la moyenne et des INNER JOIN pour relier les tables entre elles, puis comme plus haut trié et limité par 1 pour avoir la plus grande.

Une requête nous donnant le concert le plus célèbre, c’est-à-dire celui qui a le plus de participants. Nous avons compté dans Participation celui où il y avait le plus de personne et à partir de celui là nous avons cherché le nom du concert avec un INNER JOIN Concert.

Une requête nous donnant l’utilisateur le plus suivi, même chose que plus haut en remplaçant Participation par Follows.

Une requête nous donnant le plus grand lieu , c’est-à- dire celui qui a le plus grand nombre de place. Pour ça nous avons juste utilisé un WHERE du plus grand nombre de place dans LIEU.

Une requête nous donnant le Concert qui a rapporté le plus, pour cela nous avons calculé le nombre de personnes participant à chaque concert multiplié par le prix du concert pour avoir le bénéfice de toute les concert, avec ca on a trié puis limité à 1 pour n’avoir que celui qui rapporte le plus.

Un requête qui donne le prochain groupe qui organise un concert en fonction de la date actuel, la requête prend classé par du plus proche au plus loin les prochain concert qui arrive (elle ne prend que ceux qui ne sont pas encore passé “>= CURRENT_DATE”) qu’on limite à 1 pour avoir le plus proche puis à partir de cette id_concert va chercher dans Organise le groupe qui l’organise et depuis la on a id_groupe pour trouver le nom du groupe dans Utilisateur avec l’id.

3) Vers un algorithme de recommandation d’événements

Nous avons décidé de créer un indice de recommandation de concerts qui permet aux utilisateurs de se faire recommander des concerts en fonction de la proximité géographique et la popularité des concerts et groupes. Pour faire cela, nous créons une vue qui permet de stocker un indice de recommandation pour chacun des concerts. Pour créer cet indice, nous le calculons pour un concert donné à l’aide de coefficients en fonction de la moyenne des notes obtenues dans les avis sur le

groupe jouant au concert, le nombre de d'intéressés du concert ainsi que son nombre de participants. Les colonnes renvoyées par la vue incluant l'identifiant du concert, son nom, la ville et l'état où il a lieu, la date du concert, et l'indice de recommandation calculé. On filtre les données en fonction de la date qui doit correspondre à une date incluse dans une période de six mois à compter de la date courante. Enfin, la requête utilise une clause GROUP BY pour regrouper les résultats par concert, nom, ville, état, et date, afin que les résultats soient agrégés pour chaque concert unique.

On peut tester cette requête à l'aide de la commande recommandation_concerts_utilisateurs qui demande en paramètre le nom et le mail de l'utilisateur ainsi qu'un nombre puis utilise la vue ci-dessus pour récupérer les concerts les plus recommandés dans la même ville ou même état que l'utilisateur.

Nous avons aussi une seconde requête pour recommander des musiques à l'utilisateur en fonction du genre le plus écouté de sa playlist. Cette requête utilise une sous requête pour trouver le genre le plus écouté par l'utilisateur, puis avec ce genre on récupère toutes les musiques de ce genre grâce à la table Style_musique.