Modèle E/R :

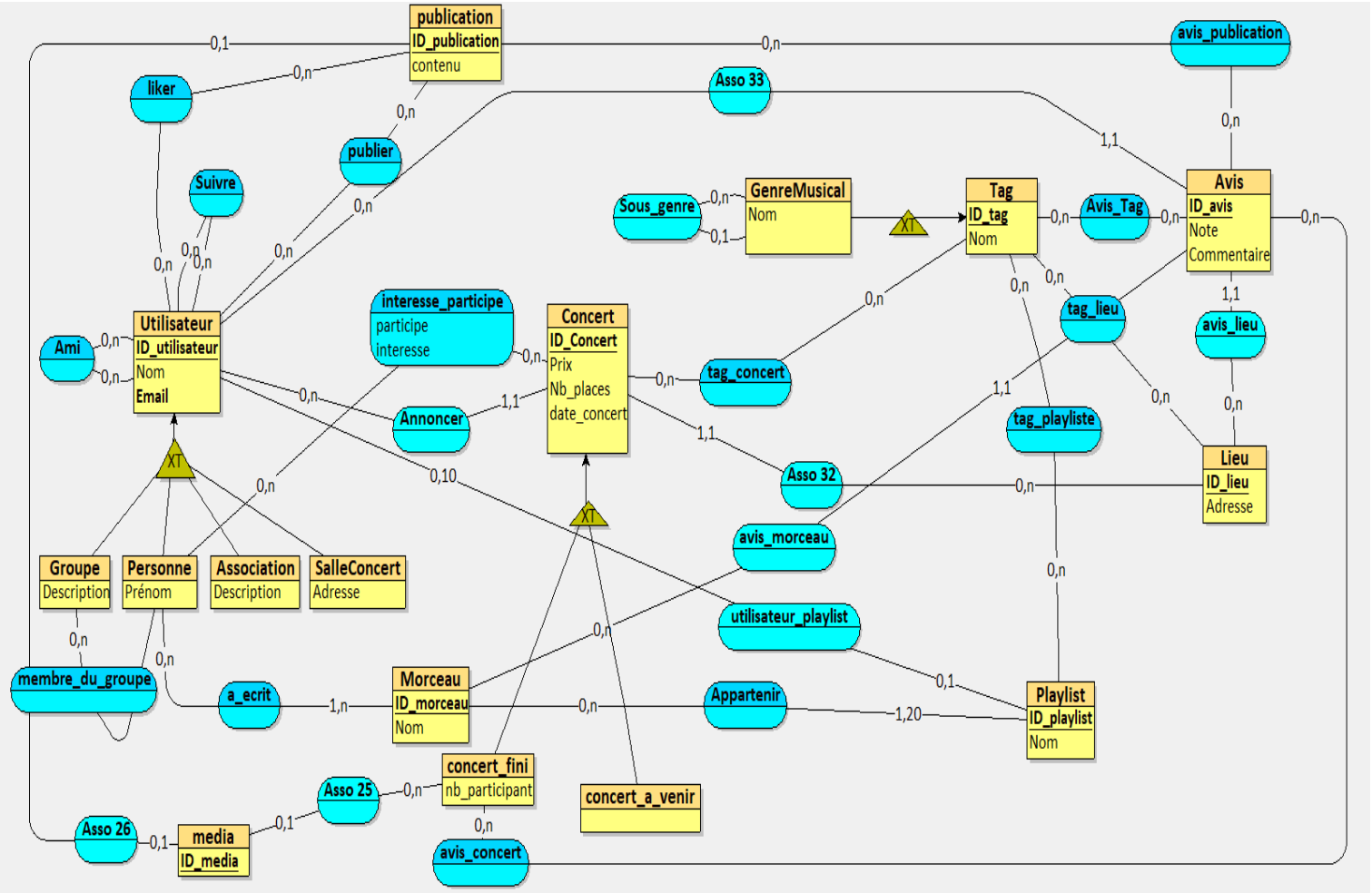

Schéma relationnel:

Utilisateur = (ID_utilisateur, Nom, Email);
Groupe = (#ID_utilisateur, Description);
Personne = (#ID_utilisateur, Prénom);
Association = (#ID_utilisateur, Description);
SalleConcert = (#ID_utilisateur, Adresse VARCHAR);
Lieu = (ID_lieu, Adresse);
Morceau = (ID_morceau, Nom);
Playlist = (ID_playlist, Nom, #ID_utilisateur);
Avis = (ID_avis, Note, Commentaire, #ID_morceau, #ID_utilisateur, #ID_lieu);
Tag = (ID_tag, Nom);
GenreMusical = (#ID_tag_1, Nom, #ID_tag);
Concert = (ID_ConcerT, Prix, Nb_places, date_concert, #ID_lieu, #ID_utilisateur);

concert_fini = (#ID_Concert, nb_participant);
concert_a_venir = (#ID_Concert);
media = (ID_media, #ID_Concert);
publication = (ID_publication, contenu, #ID_media);
Suivre = (#ID_utilisateur, #ID_utilisateur_1);
Ami = (#ID_utilisateur, #ID_utilisateur_1);
Appartenir = (#ID_morceau, #ID_playlist);
Avis_Tag = (#ID_avis, #ID_tag);
interesse_participe = (#ID_utilisateur, #ID_Concert, participe, interesse);
tag_lieu = (#ID_lieu, #ID_tag);
tag_concert = (#ID_Concert, #ID_tag);
tag_playliste = (#ID_playlist, #ID_tag);
publier = (#ID_utilisateur, #ID_publication);
avis_concert = (#ID_avis, #ID_Concert);
avis_publication = (#ID_avis, #ID_publication);
liker = (#ID_utilisateur, #ID_publication);
a_ecrit = (#ID_utilisateur, #ID_morceau);
membre_du_groupe = (#ID_utilisateur, #ID_utilisateur_1);

Contraintes d'Intégrité

- Note dans la table Avis doit être comprise entre 0 et 5.
- Nb_places dans la table table concert doit être supérieur strictement à 0
- Un utilisateur ne peut pas être ami avec lui-même et il ne peut pas se suivre d'où l'ajout de la contrainte CHECK(ID_utilisateur <> ID_utilisateur_1)
- Les booléens participe et interesse ne peuvent pas être tous les deux vrais dans la table participe_interesse

Choix de modélisation :

Chaque utilisateur est identifié par un ID unique (ID_utilisateur) qui est utilisé comme clé primaire dans la table Utilisateur. L'attribut email est également unique pour chaque utilisateur, ce qui évite la duplication des comptes.

Le modèle divise les utilisateurs en différentes catégories : Personne, Groupe, Association et SalleConcert. Cela permet de différencier les types d'utilisateurs et de stocker des informations spécifiques à chaque type.

Les tables de relation "Ami", "Suivre", "interesse_participe", "membre_du_groupe" et "publier" permettent de modéliser les interactions entre utilisateurs,

tels que les relations d'amitié, le suivi d'autres utilisateurs, l'intérêt pour les concerts, l'appartenance à un groupe musical, et la publication de contenu.

Les avis peuvent être laissés sur les morceaux, les lieux et les concerts. Il existe une table pour chaque type de relation d'avis (Avis, avis_concert, avis_publication).

Les tables "Concert", "concert_fini", "concert_a_venir" modélisent les concerts, y compris ceux qui sont passés et ceux qui sont à venir.

Les tags sont utilisés pour catégoriser les morceaux.

Limitations :

Limite du nombre de playlists par utilisateur :
Dans le modèle actuel, il n'existe aucune contrainte qui limite le nombre de playlists qu'un utilisateur peut créer.

Répartition des ID des utilisateurs entre les différentes entités :
Les entités "Personne", "Groupe", "Association" et "SalleConcert" sont toutes des sous-ensembles de l'entité "Utilisateur", partageant le même ID_utilisateur. Actuellement, rien n'empêche les ID des utilisateurs de se chevaucher entre ces entités, ce qui peut causer des problèmes d'intégrité des données.

Vérification du nombre de participants à un concert fini :

Dans le modèle actuel, il n'y a pas de contrôle pour s'assurer que le nombre de participants à un concert fini (représenté par nb_participant dans la table "concert_fini") est inférieur ou égal au nombre de places disponibles pour ce concert (représenté par Nb_places dans la table "Concert").

Requêtes dans la base de données

Une requête qui porte sur au moins trois tables qui fait une jointure :

La requête suivante joint trois tables Utilisateurs, Concert et Lieu, pour fournir une liste des utilisateurs qui ont organisé des concerts, le prix de ces concerts et l'adresse ou ces concerts ont eu lieu :

```
SELECT u.Nom, c.Prix, l.Adresse
FROM Utilisateur u
JOIN Concert c ON u.ID_utilisateur = c.ID_utilisateur
JOIN Lieu l ON c.ID_lieu = l.ID_lieu;
```

Une requête avec une jointure externe LEFT JOIN :

La requête suivante retourne le nom de chaque utilisateur et les commentaires qu'ils ont faits dans leurs avis, en faisons un LEFT JOIN entre Avis et Utilisateurs sur l'attribut 'ID_utilisateur'.

```
SELECT u.Nom, a.Commentaire
FROM Utilisateur u
LEFT JOIN Avis a ON u.ID_utilisateur = a.ID_utilisateur;
```

Une requête utilisant du fenêtrage :

La requête suivante l'ID de chaque utilisateur et le nombre total de concerts auxquels ils ont exprimé un intérêt sans pour autant y participer, et pour cela on utilise la clause 'OVER(PARTITION BY ID_utilisateur)' qui divise l'ensemble des résultats en partitions distinctes pour chaque valeurs de 'ID_utilisateur', ainsi count(*) sera appliquée individuellement pour chaque partition de l'ensemble résultats.

```
SELECT ID_utilisateur, COUNT(*) OVER(PARTITION BY ID_utilisateur) as Total_Interesse
FROM interesse_participe
WHERE participe = FALSE AND interesse = TRUE;
```

Une requête impliquant le calcul de deux agrégats :

La requête suivante calcul la moyenne du maximum des prix des concerts pour chaque groupe de ID_utlilsateur dans la table Concert.

```
SELECT AVG(MaxPrice) as AverageOfMaxPrice
FROM (
   SELECT MAX(Prix) as MaxPrice
   FROM Concert
   GROUP BY ID_utilisateur
);
```

Une requête dans le WHERE :

La requête suivante récupère le noms des utilisateurs qui ont laissé un avis pour tous les concerts auxquels ils ont assisté, la clause WHERE NOT EXSIST vérifie sil ne y'a pas de concert pour lesqules il n'y a pas d'avis, la clause AND NOT EXSIST vérifie si in n'existe pas d'avis lié a l'utilisateur actuel et au concert actuel.

```
SELECT u.Nom
FROM Utilisateur u
WHERE NOT EXISTS (
   SELECT 1
   FROM Concert c
   WHERE c.ID_utilisateur = u.ID_utilisateur
   AND NOT EXISTS (
      SELECT 1
      FROM Avis a
```

```
        WHERE a.ID_utilisateur = u.ID_utilisateur
        AND a.ID_morceau = c.ID_Concert
    )
```

Une requête avec GROUP BY :

La requete suivant retourne le nom et nombre de followers de tous les utilisateurs, en faisons une jointure avec la table Suivre pour avoir les follow et ensuite un GROUP BY, pour regrouper les utilisateurs selon leurs ID.

```
SELECT u.Nom, COUNT(*) as Number_of_Followers
FROM Utilisateur u
JOIN Suivre s ON u.ID_utilisateur = s.ID_utilisateur_1
GROUP BY u.Nom;
```

Une requête RECURSIVE :

La requête suivante la chaine des utilisateurs qui se suivent en partant de l'utilisateur avec l'ID 1, le premier SELECT récupère l'utilisateur avec l'ID 1, et ensuite le deuxième SELECT récupère les utilisateurs tels que i follow i+1 de manière récursive.

```
WITH RECURSIVE user_chain(ID_utilisateur, Nom, Email, depth, path) AS (
  SELECT u.ID_utilisateur, u.Nom, u.Email, 1, ARRAY[u.ID_utilisateur]
  FROM Utilisateur u
  WHERE u.ID_utilisateur = 1

  UNION ALL

  SELECT u.ID_utilisateur, u.Nom, u.Email, uc.depth + 1, path || u.ID_utilisateur
  FROM Utilisateur u
  JOIN Suivre s ON u.ID_utilisateur = s.ID_utilisateur_1
  JOIN user_chain uc ON s.ID_utilisateur = uc.ID_utilisateur
  WHERE NOT u.ID_utilisateur = ANY(uc.path)
)
SELECT * FROM user_chain ORDER BY depth;
```

Une sous requête dans le FROM :

La requête suivante récupère le nombre de publications de chaque utilisateur.

```
SELECT u.Nom, u.Email, (SELECT COUNT(*) FROM publication p WHERE
p.ID_publication = pub.ID_publication) AS Nombre_publications
FROM Utilisateur u,
    (SELECT ID_utilisateur, ID_publication FROM publier) AS pub
WHERE u.ID_utilisateur = pub.ID_utilisateur;
```

Une sous requête dans le WHERE :

La requête suivante récupère le nom et le courriel de chaque utilisateur tels que l'utilisateur a écrit un morceau, pour chaque utilisateur on vérifie s'il existe dans la sous requête sui retourne les IDs des utilisateurs qui ont écrit un morceau.

Une sous requête dans le where

```
SELECT u.Nom, u.Email
FROM Utilisateur u
WHERE u.ID_utilisateur IN (
   SELECT a.ID_utilisateur
   FROM a_ecrit a
);
```

Algorithme de recommandation d'évènement

Faute de temps l'algorithme n'a pas été implémenté, mais une idée est de récupérer les concert qui sont fini, ensuite pour chaque utilisateur voir la note qu'il donne au concert, ensuite pour chaque concert on va vérifier si des utilisateurs qui ont les mêmes goûts que notre utilisateur ont donné un bon avis ou pas et en fonction de cela on pourra attribuer une note au concert.