Monrousseau Matthieu 22103129
Montre Maxime 22107299

Rapport de projet : Base de Donnée

Partie 1 : Modélisation Conceptuelle :

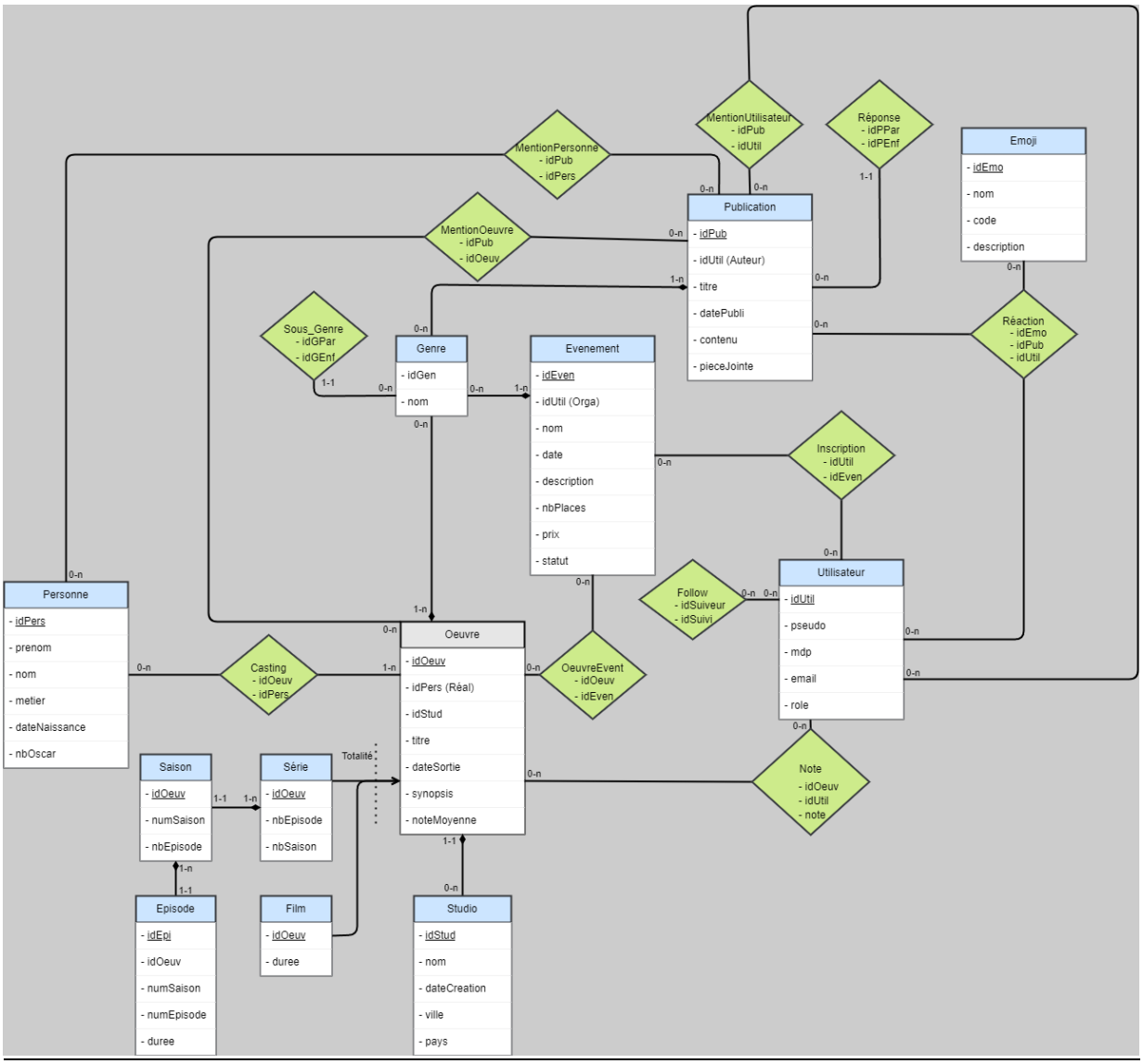

Diagramme Entités-associations :

Œuvre(idOeuv, idPers, IdStud, titre, dateSortie, synopsis)
Studio(idStud, nom, dateCreation, ville, pays)
Film(idOeuv, duree)
Série(idOeuv, nbEpisode, nbSaison)
Saison(idOeuv, numSaison, nbEpisode)
Episode(idEpi, idOeuv, numSaison, numEpisode, duree)
Personne(idPers, prenom, nom, metier, dateNaissance, nbOscar)
Utilisateur(idUtil, pseudo, mdp, email, role)
Genre(idGen, nom)
Evenement(idEven, idUtil, nom, date, description, nbPlaces, prix, statut)

Monrousseau Matthieu 22103129
Montre Maxime 22107299

Publication(idPub, idUtil, titre, datePubli, contenu, pieceJointe)
Emoji(idEmo, nom, code, description)

MentionUtilisateur(idPub, idUtil)
MentionPersonne(idPub, idPers)
MentionOeuvre(idPub, idOeuv)
Reponse(idPPar, idPEnf)
Sous_genre(idGPar, idGEnf)
Reaction(idEmo, idPub, idUtil)
Inscription(idUtil, idEven)
Follow(idSuiveur, idSuivi)
Note(idOeuv, idUtil, note)
OeuvreEvent(idOeuv, idEven)
Casting(idOeuv, idPers)

Contraintes externes :
Dans Reponse, les publication enfant doivent avoir une date plus récente que celle de la publication Parents.
La datePubli des publications doivent être antérieur à la date d'aujourd'hui.
Le rôle de l'utilisateur dont l'id est renseigné dans l'évènement doit être du rôle 'orga' car il s'agit de l'organisateur de l'évènement.
Le prix et le nombre de places d'un évènement doivent être positif.
Le statut d'un évènement doit correspondre avec le fait que sa date soit passée ou non.
Le rôle d'un Utilisateur doit être parmi les rôles suivants : admin, user, orga
Le idPers d'une œuvre doit correspondre à l'id d'une personnalité étant réalisateur.
La date de Sortie d'une œuvre doit être antérieur à celle d'aujourd'hui.
Le nombre d'épisodes dans une saison doit correspondre au nombre d'épisode relié à cette dite saison.
La note moyenne d'une œuvre doit correspondre à la moyenne des notes données par les utilisateurs sur cette œuvre.
La ville doit être dans le pays du Studio.
Un genre ne peut être sous_genre de lui-même.
Une publication ne peut être une réponse à elle-même.
Un Utilisateur ne peut se suivre lui-même.

Nous avons choisi de représenté notre base de données de la manière ci-dessus. Elle est articulée autour des classes Publications, Utilisateur, Œuvre et Personne.
Les publications peuvent contenir des mentions aux différents utilisateur, aux différentes personnalités du monde du cinéma ou directement à des œuvres grâce aux tables de relations MentionUtilisateur, MentionPersonne et MentionOeuvre et aux idPub, idPers et idOeuv. Les publications peuvent être des réponses à des publications déjà existante. C'est pourquoi il y a une table de relation Reponse avec les idPPar et idPEnf. On peut également ajouter des réactions aux publications grâce à la table

Réaction qui relie une publication à un utilisateur à un emoji avec idEmo, idPub et idUtil.

Un utilisateur peut suivre d'autres utilisateurs et être suivi par d'autres utilisateurs grâce à la table de relation Follow et les idSuiveur et idSuivi. Un utilisateur peut attribuer une note à une œuvre allant de 0 jusqu'à 100. Cette note met alors à jour la note moyenne de cette œuvre.

Les Œuvres sont réparties entre deux types, les Séries et les Films. Chaque œuvre à un idOeuv spécifique que l'on retrouve dans les tables sous-jacentes. Les séries ont un certain nombre de saisons, ayant chacune un certain nombre d'épisodes ayant chacun une certaine durée. Chaque épisode d'une même saison a la même durée pour des soucis de simplicité. Les films n'ont eux qu'une durée spécifique qui est naturellement plus longue que celle d'un épisode. Chaque œuvre à été créer par un studio retrouvable grâce à l'idStud qui se trouve dans une certaine ville et pays correspondant.

La table Personne représente la liste des personnalités travaillant sur un film ou une série. Elles possèdent un métier parmi les métiers suivant : acteur, realisateur, scenariste, producteur, cadreur, doubleur, compositeur. On retrouve la liste des personnalités ayant travaillé sur tel film grâces aux idOeuv et idPers de la table de relation Casting.

Les évènements sont quant à eux organisé par un utilisateur qui possède le rôle d'organisateur. Ils sont indiqués comme ayant lieu bientôt ou étant déjà fini en fonction de leur date. On peut retrouver la liste des utilisateur inscrit aux évènements grâce à la table de relation Inscription et les idUtil et idEven.

Partie 2 : Peuplement des tables :

Nous avons utilisé divers outils pour peupler les tables. Quand il s'agissait de tables n'ayant pas de contraintes particulières elles ont été générées par le site : https://generatedata.com/generator . Cependant quand elles ont nécessité une certaine attention aux contraintes, nous avons utiliser des scripts python afin de récupérer des informations des autres tables pour respecter les diverses contraintes des fonctionnalités. Nous avons peuplé les tables casting, épisodes, saison, follow, reaction, sous_Genre, Œuvre, Evenement, genreEvenement, genrePublication et genreOeuvre avec les scripts présents dans l'archive.

Partie 3 : Requêtes SQL :

Monrousseau Matthieu 22103129
Montre Maxime 22107299

Requête n°1 : Requête qui porte sur au moins trois tables
Cette requête récupère le titre d'une œuvre et le nom de son genre pour une œuvre spécifique, identifiée par son titre. Les jointures internes (JOIN) permettent de relier les tables Oeuvre, Film, GenreOeuvre et Genre afin de récupérer les informations pertinentes. La condition WHERE filtre les résultats pour inclure uniquement l'œuvre spécifiée.

```
SELECT Oeuvre.titre, Genre.nom
FROM Oeuvre
JOIN Film ON Oeuvre.idOeuv = Film.idOeuv
JOIN GenreOeuvre ON GenreOeuvre.idOeuv = Oeuvre.idOeuv
JOIN Genre ON GenreOeuvre.idGen = Genre.idGen
WHERE Oeuvre.titre = :'t_film';
```

Requête n°2 : Requête d'auto-jointure
Cette requête est une auto-jointure qui récupère les saisons d'une même œuvre ayant le même nombre d'épisodes. La jointure permet de filtrer les saisons qui ont le même nombre d'épisodes et de récupérer le nom de la série.

```
SELECT DISTINCT Oeuvre.titre, S1.idOeuv, S1.numSaison, S1.nbEpisodes,
S2.numSaison, S2.nbEpisodes
FROM Saison S1, Saison S2
JOIN Oeuvre ON S2.idOeuv = Oeuvre.idOeuv
WHERE S1.idOeuv = S2.idOEuv
AND S1.numSaison < S2.numSaison
AND S1.nbEpisodes = S2.nbEpisodes;
```

Requête n°3 : Requête avec sous requête corrélée
Cette requête récupère les pseudos des utilisateurs qui ont publié quelque chose l'année spécifiée par :t_annee. La condition WHERE EXISTS vérifie pour chaque utilisateur s'il existe au moins une publication dont l'année est postérieure à :t_annee. Si c'est le cas, le pseudo de cet utilisateur est sélectionné.

```
SELECT P1.Pseudo
FROM Utilisateur P1
WHERE EXISTS (
   SELECT 1
   FROM Publication Pub
   WHERE Pub.idUtil = P1.idUtil
   AND EXTRACT(YEAR FROM Pub.DatePubli) > :'t_annee'::int
);
```

Requête n°4 : Requête avec sous-requête dans le FROM
Cette requête récupère les titres des œuvres, leurs dates de sortie, et le nom d'un genre spécifique. Elle utilise une sous-requête pour filtrer les genres en fonction du nom du genre (:t_genre), puis fait des jointures pour associer les œuvres et leurs genres. La clause DISTINCT assure que les résultats ne contiennent pas de doublons.

```
SELECT Distinct Oeuvre.titre, Oeuvre.dateSortie, G.nom
```

```
FROM (SELECT G.idGen, G.nom FROM Genre G WHERE G.nom = :'t_genre') as G,
GenreOeuvre, Oeuvre
WHERE Oeuvre.idOeuv = GenreOeuvre.idOeuv AND GenreOeuvre.idGen = G.idGen;
```

Requête n°5 : Requête avec sous-requête dans le WHERE

Cette requête récupère les titres et les dates de sortie des œuvres du genre spécifié (:t_genre) qui ont la date de sortie la plus récente. Elle utilise des jointures pour associer les œuvres avec leurs genres via la table de relation GenreOeuvre. La sous-requête est utilisée pour trouver la date de sortie maximale parmi toutes les œuvres.

```
SELECT Oeuvre.titre, Oeuvre.dateSortie FROM Oeuvre, GenreOeuvre, Genre
WHERE Oeuvre.DateSortie = (SELECT MAX(DateSortie) FROM Oeuvre)
AND Oeuvre.idOeuv = GenreOeuvre.idOeuv AND Genre.nom = :'t_genre';
```

Requête n°6 : Requête impliquant deux agrégats nécessitant GROUP BY et HAVING

Cette requête identifie les œuvres qui ont été le plus mentionnées dans les publications pour une année donnée (:t_annee). Elle utilise des jointures pour associer les œuvres avec leurs mentions dans les publications. Le filtrage par année est réalisé avec la clause WHERE et l'agrégation des mentions est faite avec GROUP BY. La condition HAVING compare le nombre de mentions de chaque œuvre avec le nombre maximum de mentions pour l'année spécifiée, retournant ainsi l'œuvre la plus mentionnée.

```
SELECT Oeuvre.titre, COUNT(*) as nb
FROM Oeuvre
JOIN MentionOeuvre as MO ON Oeuvre.idOeuv = MO.idOeuv
JOIN Publication PU ON MO.idPub = PU.idPub
WHERE EXTRACT(YEAR FROM PU.DatePubli) = :'t_anne'::int
GROUP BY Oeuvre.titre
HAVING COUNT(*) >= ALL (
   SELECT COUNT(*)
   FROM Oeuvre
   JOIN MentionOeuvre as MO ON Oeuvre.idOeuv = MO.idOeuv
   JOIN Publication PU ON MO.idPub = PU.idPub
   WHERE EXTRACT(YEAR FROM PU.DatePubli) = :'t_anne'::int
   GROUP BY Oeuvre.titre
);
```

Requête n°7 : Requête impliquant le calcul de deux agrégats

Cette requête identifie l'œuvre ayant le plus grand nombre d'épisodes en utilisant des sous-requêtes imbriquées pour calculer et comparer les totaux d'épisodes par œuvre. Les sous-requêtes permettent de d'abord agrégater les données par œuvre et de trouver le maximum parmi ces agrégats. Ensuite, la requête principale sélectionne les titres des œuvres et leurs totaux d'épisodes, en filtrant pour ne garder que l'œuvre ayant le total maximal.

```
SELECT O.titre, S.total_episodes
```

```
FROM Oeuvre O
JOIN (
   SELECT idOeuv, SUM(nbEpisodes) AS total_episodes
   FROM Saison
   GROUP BY idOeuv
) AS S ON O.idOeuv = S.idOeuv
WHERE S.total_episodes = (
   SELECT MAX(total_episodes)
   FROM (
      SELECT SUM(nbEpisodes) AS total_episodes
      FROM Saison
      GROUP BY idOeuv
   ) AS TotalEpisodes
);
```

Requête n°8 : Requête utilisant une jointure externe
Cette requête liste tous les utilisateurs et le nombre de notations qu'ils ont faites. Pour les utilisateurs qui n'ont pas fait de notations, la requête retourne 0 grâce à la fonction COALESCE. Une jointure externe gauche est utilisée pour s'assurer que tous les utilisateurs sont inclus dans les résultats, même ceux sans notations.

```
SELECT U.pseudo, COALESCE(S.notations, 0) AS total_notations
FROM Utilisateur U
LEFT JOIN (
   SELECT idUtil, COUNT(*) AS notations FROM Note GROUP BY idUtil
) AS S ON U.idUtil = S.idUtil;
```

Requête n°9: Deux Requêtes équivalentes exprimant une condition de totalité, l’une avec sous-requêtes corrélées et l’autre avec agrégation :

```
SELECT O.titre
FROM Oeuvre O
WHERE EXISTS (
   SELECT 1
   FROM Saison S
   WHERE S.idOeuv = O.idOeuv
);
SELECT O.titre
FROM Oeuvre O
JOIN Saison S ON O.idOeuv = S.idOeuv
GROUP BY O.titre
HAVING COUNT(S.numSaison) > 0;
```

Requête n°10 : Requête récursive
Cette requête trouve la prochaine date sans événement en vérifiant chaque jour successivement à partir de la date actuelle. Elle utilise une requête récursive pour ajouter un jour à chaque itération et vérifier s'il y a un événement prévu pour cette nouvelle date.

En combinant ces éléments, la requête est capable de trouver et retourner la première date sans événement et d'inclure le nom de l'événement pour ce jour, s'il en existe un.

```
WITH RECURSIVE prochainEvent AS (
   SELECT CURRENT_DATE::timestamp AS prochain
   UNION ALL
   SELECT prochain + INTERVAL '1 day'
   FROM prochainEvent
   WHERE NOT EXISTS (
      SELECT 1
      FROM Evenement
      WHERE dateEven = prochain + INTERVAL '1 day'
   )
)
SELECT prochain::date
FROM prochainEvent
WHERE prochain > CURRENT_DATE
ORDER BY prochain
LIMIT 1;
```

Requête n°11 : Requête utilisant le fenêtrage
Cette requête sélectionne l'identifiant de l'utilisateur et la date de publication à partir de la table "Publication". Elle compte le nombre de publications pour chaque mois de l'année 2023, en utilisant une fonction de fenêtrage pour partitionner les données par mois.

```
SELECT idUtil, DatePubli,
COUNT(*) OVER (PARTITION BY EXTRACT(MONTH FROM DatePubli) ORDER BY
DatePubli) AS MonthlyCount
FROM Publication
WHERE EXTRACT(YEAR FROM DatePubli) = 2023;
```

Partie 4 : Algorithme de recommandation :

Pour l'algorithme de recommandation, nous avons essayer de passer par une requête qui prend l'id d'un utilisateur et renvoie la liste des films qui pourrait l'intéresser. Nous avons évalué deux critères pour cela. Le premier est la liste des publications que l'utilisateur à fait et notamment les mentions aux œuvres faites dans celles-ci. La seconde est les notes que l'utilisateur à attribué aux films. Tout d'abord nous récupérons la liste des genres des œuvres que l'utilisateur à mentionner dans les publications qu'il a écrit. Nous faisons alors une union pour rajouter les genres des films pour lesquels l'utilisateur à mis une note dépassant les 60/100. Avec ceci, nous rajoutons les sous-genres découlant de ces genres-là. Et nous récupérons enfin la liste des œuvres ayant ces genres là et qui ne sont pas parmi ceux que l'utilisateur à déjà noté.

Requête :

Monrousseau Matthieu 22103129
Montre Maxime 22107299

```
WITH UserGenres AS (
        SELECT DISTINCT gp.idGen
        FROM publication p
        JOIN GenrePublication gp ON p.idPub = gp.idPub
        WHERE p.idUtil = :userId
        UNION
        SELECT DISTINCT g.idGen
        FROM note n
        JOIN oeuvre o ON n.idOeuv = o.idOeuv
        JOIN genreOeuvre gp ON o.idOeuv = gp.idOeuv
        JOIN genre g ON gp.idGen = g.idGen
        WHERE n.idUtil = :userId AND n.note > 60 ),

SousUserGenre AS (
        SELECT DISTINCT sg.idGEnfant
        FROM SousGenre sg
        JOIN UserGenres ug ON sg.idGParent = ug.idGen
        WHERE gp.idGen IN (SELECT idGen FROM UserGenres)

)

SELECT DISTINCT o.titre
FROM UserGenres ug, SousUserGenre sug, Genre g
JOIN GenreOeuvre go ON g.idGen = go.idGen
JOIN Oeuvre o ON go.idOeuv = o.idOeuv
WHERE ug.idGen = g.idGen OR sug.idGenfant = g.idGen
```