POLYTECH©

PROJET
BASE DE DONNÉES

préparé par

Ali Madandar Arani **Olivier Boulet**

Haitam Benabad **Karbal Adam**

SUJET 5: Réseau social

I. Reformulation et Description du projet :

On souhaite créer une base de données permettant de gérer un réseau social décentralisé. L'objectif est de fournir des fonctionnalités avancées qui répondent aux besoins des utilisateurs.

Pour cela on a définit plusieurs objectifs principaux :
- Gérer les utilisateurs : On doit pouvoir gérer les informations d'un utilisateur, ses connexions et les activités de celui-ci.
- Gérer les publications : Être capable de stocker les publications créées par les utilisateurs, ces publications ont un contenu, un auteur précis, une visibilité et des interactions associées (likes,commentaires,partage).
- Gérer les interactions : Pouvoir enregistrer les interactions des utilisateurs avec les publications et entre eux.
- Gérer les groupes thématiques : Structurer les informations sur les groupes, leurs membres, les publications partagées dans ces groupes, et les interactions qu'ils génèrent.
- Evolutivité et intégration : Pouvoir intégrer facilement un utilisateur.

Mais on à également définies les services rendues par la base de données :
- Analyse des connexions entre utilisateurs : Pouvoir trouver les connexions directes et indirectes. Mettre en évidence les intérêts communs entre les utilisateurs.
- Gestion des publications et des interactions : Identifier les publications qui génèrent le plus d'engagement et analyser les types d'interactions (like,commentaires,partage).
- Recommandations personnalisées : Proposer des groupes ou des connexions à un utilisateur en fonction de son activité récente et identifier les utilisateurs influents au sein du groupe.
- Statistiques sur les groupes : identifier les groupes les plus actifs et analyser les types de publications les plus virales dans les groupes.
- Intégration et mise à jour des données : Ajouter de nouveaux utilisateurs de manière cohérente et permettre l'intégration facile de nouvelles publications, interactions ou membres (dans un groupe existant).
- Analyser des interactions hors cercles directs : identifier les utilisateurs qui ont des interactions similaires, pour proposer des amis cohérents.

Grâce à ces deux points on à pu définir les entités principaux :
-Utilisateurs : Contient les informations personnelles des utilisateurs.
-Publications : Contient les détails sur la publication.

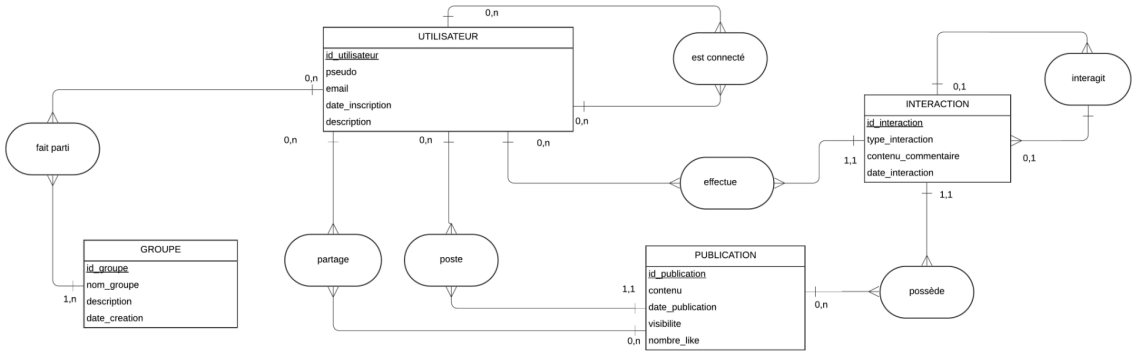
-Interactions : Enregistre les interactions des utilisateurs avec les publications et entre eux.
-Groupes thématiques : Contient les informations sur les groupes et leur membre.

Mais aussi définir des relations tel que :
-La connexions entre utilisateur : une relation entre les utilisateurs définissant les connexions (ami ou abonnée)
-Les publications et interactions
-Les membre de groupes

II. Schémas de la base de données :

A. modèle Entité-Association

Le schéma de la base de données, exprimé d'une manière visuelles selon le modèle EA (vu en
cours). Le commentaire expliquera et justifiera les choix effectués (≈ spécification détaillée),



On a décidé de représenter un utilisateur, une interaction, un groupe et une publication avec des tables, car ce sont des entités à part entière. De plus, chaque instance de ces tables est unique et identifiée par un identifiant unique.

Un utilisateur est identifié par son identifiant, qui est la clé primaire de la table. Un utilisateur possède un pseudo, un e-mail, une date d'inscription et une description représentant la biographie de son compte. Un utilisateur peut être connecté à 0 ou n utilisateurs. Un utilisateur peut poster de 0 à n publications.

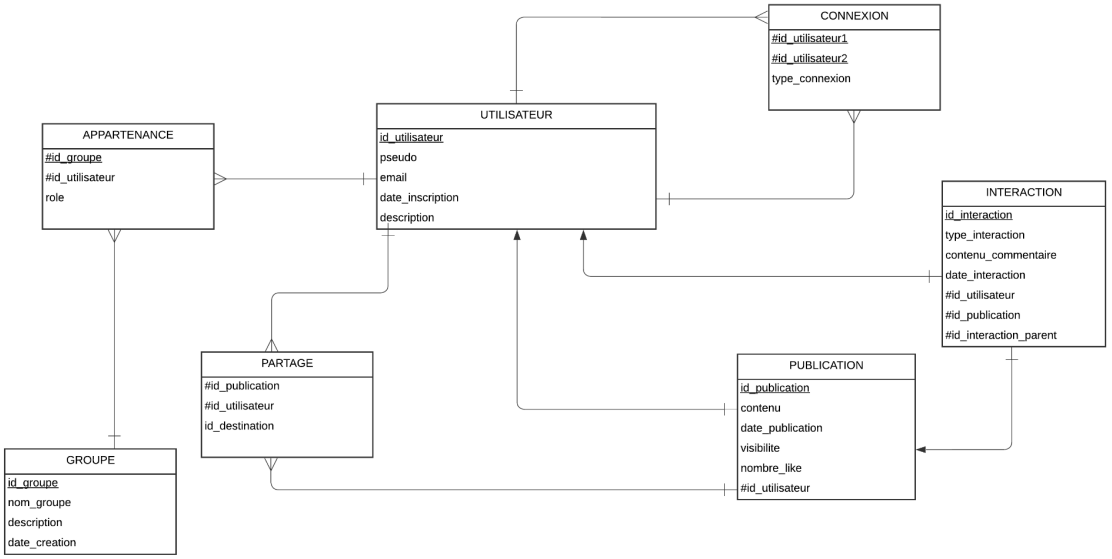
Une publication est identifiée par son identifiant, possède un contenu, une date, une visibilité (privée ou publique) et un nombre de likes. Une publication est postée par un seul utilisateur. Elle possède 0 ou n interactions.

Une interaction est identifiée par son identifiant unique et possède un type (like ou commentaire), un contenu de commentaire si le type est un commentaire, ainsi qu'une date à laquelle l'interaction a été effectuée. L'interaction est effectuée par un seul utilisateur et est associée à une seule publication. Un utilisateur peut effectuer de 0 à n interactions sur n'importe quelle publication à laquelle il a accès.
Une interaction peut avoir une sous interaction comme par exemple commenter ou liker un commentaire.

Un utilisateur peut faire partie d'un ou plusieurs groupes. Un groupe est identifié par son identifiant, il possède un nom, une description ainsi qu'une date de création. Un groupe est composé de 1 à n utilisateurs. Il est possible de partager des publications dans un groupe mais seulement dans un groupe.

B. Modèle relationnel :

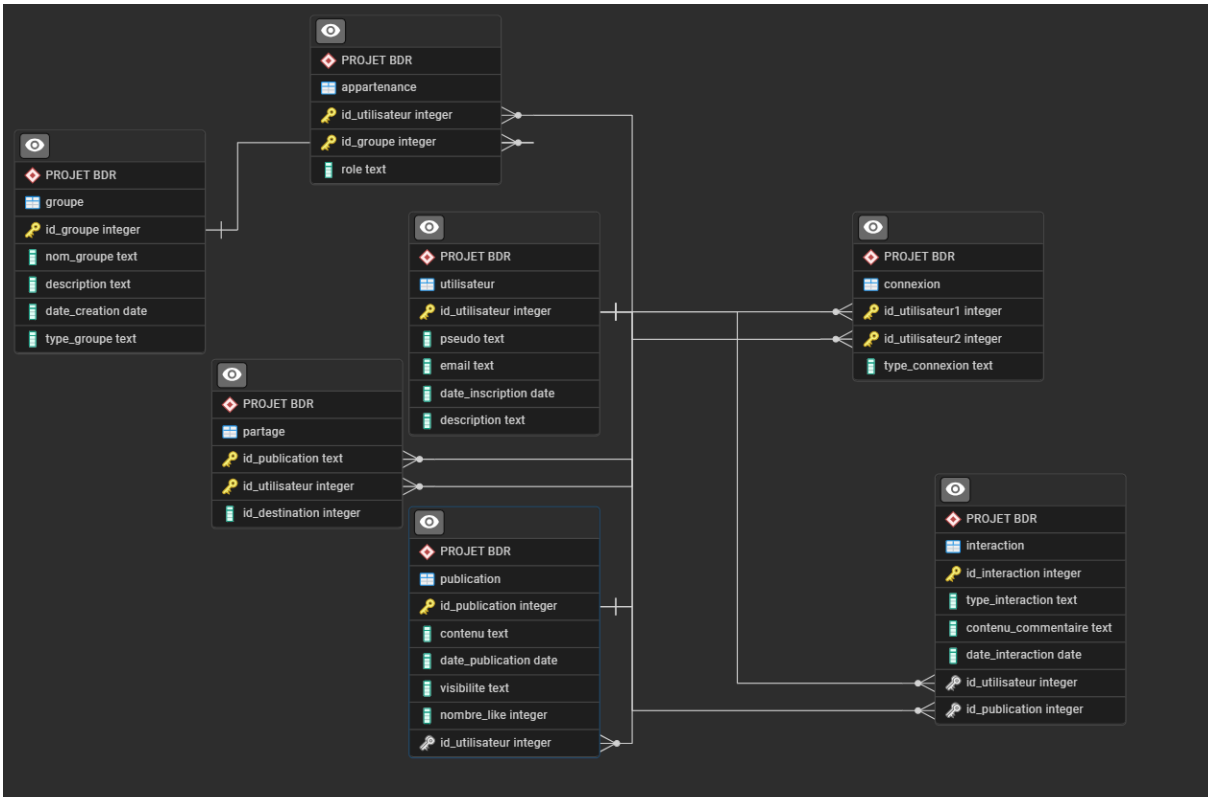
Nous avons aussi utilisé une représentation relationnelle, en transformant des relations en table ou en clés étrangères selon les cas:



Olivier et Ali

C. Modèle pgAdmin :

Nous avons décidé d'utiliser les outils de pgAdmin afin de créer un schéma de la base de données. Nous avons donc pu générer le code de sa création puis l'adapter et le modifier.



Olivier et Ali

III. Requêtes utiles :

Requête 1 (Haitam) :

Trouver les amis d'amis d'un utilisateur donné :

```
SELECT DISTINCT u2.id_utilisateur, u2.pseudo
FROM "PROJET BDR".connexion c1
JOIN "PROJET BDR".connexion c2
   ON c1.id_utilisateur2 = c2.id_utilisateur1
JOIN "PROJET BDR".utilisateur u2
   ON c2.id_utilisateur2 = u2.id_utilisateur
WHERE c1.id_utilisateur1 = 1
   AND c2.id_utilisateur2 != c1.id_utilisateur1
   AND NOT EXISTS (
      SELECT 1 FROM "PROJET BDR".connexion c3
      WHERE c3.id_utilisateur1 = 1 AND c3.id_utilisateur2 = c2.id_utilisateur2
   )
   AND EXISTS (
      SELECT 1 FROM "PROJET BDR".connexion c4
      WHERE c4.id_utilisateur1 = c1.id_utilisateur2 AND c4.id_utilisateur2 =
c1.id_utilisateur1
   )
   AND EXISTS (
      SELECT 1 FROM "PROJET BDR".connexion c5
      WHERE c5.id_utilisateur1 = c2.id_utilisateur2 AND c5.id_utilisateur2 =
c2.id_utilisateur1
   );
```

Premièrement, on sélectionne toutes les connexions de l'utilisateur 1 avec "c1.id_utilisateur2" qui représente les amis directs ou connexions de l'utilisateur. Ensuite, on cherche les connexions des amis directs de l'utilisateur 1 où "c2.id_utilisateur2" représente les amis d'amis.
La jointure "JOIN "PROJET BDR".utilisateur u2" permet d'obtenir le pseudo et l'identifiant "id_utilisateur, pseudo" des amis d'amis.

Ainsi, on mettra en place des conditions de filtrage tel que exclure l'utilisateur lui-même, ne pas suggérer des amis déjà directs, vérifier que la relation c1 est réciproque (connexion mutuelle) et enfin vérifier que l'ami d'ami est bien un ami mutuel avec son ami direct.

Cette requête trouve efficacement les amis d'amis en vérifiant que toutes les relations sont mutuelles.

Requête 2 (Ali) :

Publications les plus populaires:

```
SELECT *
FROM "PROJET BDR".publication
ORDER BY nombre_like DESC
LIMIT 5;
```

Cette requête prend toutes les publications et les classe par popularité donc par nombre de likes sur celles-ci. Ensuite, elle renvoie les 5 publications les plus populaires.

Requête 3 (Ali) :

Recommander des connexions à un utilisateur (Propose des gens qui ont les même interactions):

```
WITH interactions_communes AS (
    SELECT
        i1.id_utilisateur AS utilisateur_cible,
        i2.id_utilisateur AS utilisateur_suggere,
        COUNT(*) AS nombre_interactions_communes
    FROM "PROJET BDR".interaction i1
    JOIN "PROJET BDR".interaction i2
        ON i1.id_publication = i2.id_publication
        AND i1.id_utilisateur <> i2.id_utilisateur
    WHERE i1.id_utilisateur = 1
    GROUP BY i1.id_utilisateur, i2.id_utilisateur
)
SELECT
    uc.pseudo AS pseudo_cible,
    u.id_utilisateur,
    u.pseudo,
    ic.nombre_interactions_communes
FROM interactions_communes ic
JOIN "PROJET BDR".utilisateur u ON ic.utilisateur_suggere = u.id_utilisateur
JOIN "PROJET BDR".utilisateur uc ON ic.utilisateur_cible = uc.id_utilisateur
LEFT JOIN "PROJET BDR".connexion c
    ON ic.utilisateur_cible = c.id_utilisateur1
    AND ic.utilisateur_suggere = c.id_utilisateur2
WHERE c.id_utilisateur1 IS NULL
ORDER BY ic.nombre_interactions_communes DESC
LIMIT 10;
```

Cette requête recommande des connexions en analysant les interactions communes entre utilisateurs sur des publications. Elle commence par identifier les utilisateurs ayant interagi avec les mêmes publications que l'utilisateur cible et compte le nombre d'interactions partagées. Ensuite, elle exclut les utilisateurs déjà connectés à lui pour ne suggérer que de nouvelles relations pertinentes. Les suggestions sont triées par nombre d'interactions communes, mettant en avant ceux avec le plus d'affinités. Enfin, elle affiche le pseudo de l'utilisateur cible et les utilisateurs recommandés.

Requête 4 (Adam) :

Nombre de partage dans un groupe :

Le but de cette requête est d'identifier le nombre de partages dans un groupe. Pour cela, nous allons rechercher tous les partages dont la destination est l'ID du groupe. Une fois cette étape réalisée, nous afficherons le nom du groupe ainsi que le nombre de partages, triés par ordre décroissant du nombre de partages.

```
SELECT g.nom_groupe, COUNT(p.id_destination) AS nombre_interactions
FROM “PROJET BDR”.partage p
JOIN “PROJET BDR”.groupe g ON g.id_groupe=p.id_destination
ORDER BY nombre_interactions DESC;
```

Requête 5 (Adam) :

Utilisateurs influençant le plus les interactions dans un groupe :

Le but de cette requête est d'identifier les utilisateurs qui interagissent le plus dans un groupe donné (en l'occurrence, le groupe numéro 1 ici).Pour cela, nous allons afficher le nom du groupe, le pseudo de l'utilisateur, son identifiant ainsi que le nombre d'interactions qu'il a réalisées au sein de ce groupe.Nous commencerons par récupérer tous les utilisateurs appartenant au groupe 1, puis nous comptabiliserons leurs interactions. Enfin, nous trierons les résultats par ordre décroissant du nombre d'interactions.

```
SELECT g.nom_groupe, u.pseudo, u.id_utilisateur, COUNT(i.id_interaction) AS
nombre_interactions
FROM "PROJET BDR".utilisateur u
JOIN "PROJET BDR".appartenance a ON a.id_utilisateur = u.id_utilisateur
JOIN "PROJET BDR".groupe g ON g.id_groupe = a.id_groupe
```

```
JOIN "PROJET BDR".interaction i ON i.id_utilisateur = u.id_utilisateur
WHERE g.id_groupe = 1  -- ID DU GROUPE VOULU
GROUP BY u.id_utilisateur, u.pseudo, g.nom_groupe
ORDER BY nombre_interactions DESC;
```

Requête 6 (Olivier) :

Recommander des groupes basés sur l'activité d'un utilisateur (recommande le ou les groupes avec le plus d'interaction de la part de l'utilisateur sur les publications qui ont ce thème):

```
SELECT DISTINCT g.nom_groupe
FROM "PROJET BDR".groupe g
JOIN (
    SELECT p.theme, COUNT(*) AS total_likes
    FROM "PROJET BDR".interaction i
    JOIN "PROJET BDR".publication p ON i.id_publication = p.id_publication
    WHERE i.type_interaction = 'like' AND i.id_utilisateur = 1
    GROUP BY p.theme
    ORDER BY total_likes DESC
) user_themes ON g.nom_groupe = user_themes.theme;
```

Explication :
Dans un premier temps, nous comptons le nombre de likes de l'utilisateur pour chaque thème de publication et les classons par popularité. Ensuite, nous effectuons une jointure entre ces thèmes et les groupes ayant le même nom que ces thèmes, afin de recommander à l'utilisateur des groupes correspondant à ses centres d'intérêt.

Requête Bonus (Olivier) :

FEED par rapport aux connexions de l'utilisateur :

```
SELECT DISTINCT p.contenu, p.date_publication, p.nombre_like
FROM "PROJET BDR".utilisateur u
JOIN "PROJET BDR".connexion c ON u.id_utilisateur = c.id_utilisateur1
JOIN "PROJET BDR".publication p ON p.id_utilisateur = c.id_utilisateur2
WHERE c.type_connexion IN ('ami', 'suivi')
  AND u.id_utilisateur = 1
ORDER BY p.date_publication DESC;
```

Explication:

Cette requête récupère les publications des utilisateurs connectés à un utilisateur donné (ici : le 1). Elle commence par identifier les connexions de l'utilisateur grâce à une jointure entre les tables utilisateur et connexion, puis récupère les publications des utilisateurs connectés via une jointure avec la table publication. Un filtre est appliqué pour ne sélectionner que les publications des personnes liées à id_utilisateur = 1, et un tri est effectué pour afficher en premier les publications les plus récentes grâce à ORDER BY p.date_publication DESC. Ainsi, l'utilisateur peut voir en priorité les dernières publications de ses amis ou des personnes qu'il suit.

IV. Jeu de données :

Ce jeu de données a été sélectionné en fonction des requêtes utiles, afin de garantir un minimum d'informations permettant d'obtenir des résultats cohérents.

```
-- Remplissage de la table utilisateur
INSERT INTO "PROJET BDR".utilisateur (id_utilisateur, pseudo, email,
date_inscription, description)
VALUES
    (1, 'JohnDoe', 'john.doe@example.com', '2023-01-01', 'Utilisateur passionné de
technologie.'),
    (2, 'JaneSmith', 'jane.smith@example.com', '2023-02-15', 'Aime la photographie.'),
    (3, 'Alex99', 'alex99@example.com', '2023-03-10', 'Fan de jeux vidéo.'),
    (4, 'EmmaW', 'emma.w@example.com', '2023-04-10', 'Adepte du voyage.'),
    (5, 'MikeT', 'mike.t@example.com', '2023-05-15', 'Musicien amateur.'),
    (6, 'SarahG', 'sarah.g@example.com', '2023-06-20', 'Passionnée de lecture.'),
    (7, 'DavidK', 'david.k@example.com', '2023-07-05', 'Ingénieur en informatique.'),
    (8, 'LauraP', 'laura.p@example.com', '2023-08-12', 'Fan de sport et de fitness.'),
    (9, 'TomR', 'tom.r@example.com', '2023-09-01', 'Amateur de cinéma.'),
    (10, 'AliceB', 'alice.b@example.com', '2023-10-03', 'Développeuse full stack.'),
    (11, 'RyanS', 'ryan.s@example.com', '2023-11-11', 'Gourmand et passionné de
cuisine.'),
    (12, 'SophiaL', 'sophia.l@example.com', '2023-12-20', 'Férue d'art et de
peinture.'),
    (13, 'NathanM', 'nathan.m@example.com', '2024-01-05', 'Étudiant en
informatique.'),
    (14, 'OliviaD', 'olivia.d@example.com', '2024-02-10', 'Cycliste et aventurière.'),
    (15, 'LucasF', 'lucas.f@example.com', '2024-03-15', 'Fan de jeux vidéo.'),
    (16, 'EllaC', 'ella.c@example.com', '2024-04-20', 'Aime la photographie.'),
    (17, 'JamesN', 'james.n@example.com', '2024-05-05', 'Développeur mobile.'),
    (18, 'LiamB', 'liam.b@example.com', '2024-06-10', 'Streamer gaming.'),
```

```
    (19, 'MiaV', 'mia.v@example.com', '2024-07-25', 'Danseuse et chorégraphe.'),
    (20, 'EthanW', 'ethan.w@example.com', '2024-08-30', 'Photographe
professionnel.');

-- Remplissage de la table connexion
INSERT INTO "PROJET BDR".connexion (id_utilisateur1, id_utilisateur2,
type_connexion)
VALUES
    (1, 2, 'ami'),
    (1, 3, 'suivi'),
    (2, 3, 'ami'),
    (1, 4, 'ami'),
    (4, 1, 'ami'),
    (1, 5, 'suivi'),
    (2, 6, 'suivi'),
    (2, 7, 'suivi'),
    (3, 8, 'ami'),
    (8, 3, 'ami'),
    (3, 9, 'suivi'),
    (4, 10, 'ami'),
    (10, 4, 'ami'),
    (4, 11, 'suivi'),
    (5, 12, 'suivi'),
    (6, 13, 'suivi'),
    (7, 14, 'suivi'),
    (8, 15, 'ami'),
    (15, 8, 'ami'),
    (9, 16, 'suivi'),
    (10, 17, 'suivi'),
    (11, 18, 'suivi'),
    (12, 19, 'suivi'),
    (13, 20, 'ami'),
    (20, 13, 'ami'),
    (14, 1, 'suivi'),
    (15, 2, 'suivi'),
    (16, 3, 'suivi');


-- Remplissage de la table publication
INSERT INTO "PROJET BDR".publication (id_publication, contenu,
date_publication, visibilite, nombre_like, id_utilisateur,theme)
VALUES
    (1, 'Bonjour tout le monde !', '2023-01-15', 'public', 10, 1,NULL),
```

```
    (2, 'Voici une nouvelle photo de mon voyage.', '2023-02-20', 'amis', 25,
2,'Photographie'),
    (3, 'Quelqu'un a des recommandations de livres ?', '2023-03-05', 'public', 5,
3,'Lecture'),
(4, 'Quel beau coucher de soleil !', '2023-04-15', 'public', 12, 4,'Paysage'),
    (5, 'Nouvelle chanson en préparation.', '2023-05-20', 'amis', 7, 5,'Musique'),
    (6, 'Quel est votre livre préféré ?', '2023-06-25', 'public', 9, 6,'Lecture'),
    (7, 'Mon projet de développement avance bien.', '2023-07-10', 'public', 5, 7,NULL),
    (8, 'Entraînement intensif aujourd'hui !', '2023-08-18', 'amis', 14, 8,'Sport'),
    (9, 'Dernier film vu au cinéma : incroyable !', '2023-09-02', 'public', 6, 9,'Cinéma'),
    (10, 'Petit bug résolu, quel soulagement !', '2023-10-05', 'amis', 8, 10,NULL),
    (11, 'Recette du jour : tarte aux pommes.', '2023-11-12', 'public', 10,
11,'Nourriture'),
    (12, 'Nouvelle toile en cours de création.', '2023-12-22', 'amis', 4, 12,'Art'),
    (13, 'Quel est votre langage de programmation préféré ?', '2024-01-07', 'public',
11, 13,'Informatique'),
    (14, 'Belle balade à vélo ce matin !', '2024-02-12', 'public', 3, 14,'Sport'),
    (15, 'Top 5 de mes jeux vidéo préférés.', '2024-03-17', 'amis', 7, 15,'Jeux Vidéos'),
    (16, 'Nouveau shooting photo terminé !', '2024-04-22', 'public', 9,
16,'Photographie'),
    (17, 'Appli mobile en développement.', '2024-05-07', 'amis', 5, 17,NULL),
    (18, 'Live gaming ce soir sur Twitch.', '2024-06-12', 'public', 13, 18,NULL),
    (19, 'Répétition avant le spectacle !', '2024-07-27', 'public', 2, 19,NULL),
    (20, 'Expo photo en préparation.', '2024-08-31', 'amis', 6, 20,'Photographie');



-- Remplissage de la table interaction
INSERT INTO "PROJET BDR".interaction (id_interaction, type_interaction,
contenu_commentaire, date_interaction, id_utilisateur, id_publication)
VALUES
    (1, 'like', NULL, '2023-01-16', 2, 1),
    (2, 'commentaire', 'Belle photo !', '2023-02-21', 1, 2),
    (3, 'like', NULL, '2023-03-06', 1, 3),
    (4, 'like', NULL, '2023-04-16', 5, 4),
    (5, 'commentaire', 'Superbe !', '2023-04-17', 6, 4),
    (6, 'like', NULL, '2023-05-21', 7, 5),
    (7, 'commentaire', 'J'ai hâte d'écouter ça !', '2023-05-22', 8, 5),
    (8, 'like', NULL, '2023-06-26', 9, 6),
    (9, 'commentaire', 'J'adore cet auteur !', '2023-06-27', 10, 6),
    (10, 'like', NULL, '2023-07-11', 11, 7),
    (11, 'commentaire', 'Bon courage pour ton projet.', '2023-07-12', 12, 7),
    (12, 'like', NULL, '2023-08-19', 13, 8),
    (13, 'commentaire', 'Bravo pour tes progrès !', '2023-08-20', 14, 8),
```

```
    (14, 'like', NULL, '2023-09-03', 15, 9),
    (15, 'commentaire', 'Ce film était génial !', '2023-09-04', 16, 9),
    (16, 'like', NULL, '2023-10-06', 17, 10),
    (17, 'commentaire', 'Je connais ce sentiment !', '2023-10-07', 18, 10),
    (18, 'like', NULL, '2023-11-13', 19, 11),
    (19, 'commentaire', 'Miam, je vais essayer.', '2023-11-14', 20, 11),
    (20, 'like', NULL, '2023-12-23', 1, 12),
    (21, 'like', NULL, '2023-02-21', 3, 2),
    (22, 'like', NULL, '2023-02-21', 4, 2),
    (23, 'like', NULL, '2023-02-21', 5, 2),
    (24, 'like', NULL, '2023-02-21', 6, 2),
    (25, 'like', NULL, '2023-02-21', 7, 2),
    (26, 'like', NULL, '2023-02-21', 8, 2),
    (27, 'like', NULL, '2023-02-21', 9, 2),
    (28, 'like', NULL, '2023-02-21', 10, 2),
    (29, 'like', NULL, '2023-02-21', 11, 2),
    (30, 'like', NULL, '2023-02-21', 12, 2),
    (31, 'like', NULL, '2023-02-21', 13, 2),
    (32, 'like', NULL, '2023-02-21', 14, 2),
    (33, 'like', NULL, '2023-02-21', 15, 2),
    (34, 'like', NULL, '2023-02-21', 16, 2),
    (40, 'like', NULL, '2023-03-06', 1, 16),
    (41, 'like', NULL, '2023-03-06', 2, 3),
    (42, 'like', NULL, '2023-03-06', 4, 3),
    (43, 'like', NULL, '2023-12-23', 1, 15),
    (44, 'like', NULL, '2023-12-23', 6, 15),
    (45, 'like', NULL, '2023-12-23', 6, 12),
    (46, 'like', NULL, '2023-12-23', 1, 10),
    (47, 'like', NULL, '2023-12-23', 6, 10);



-- Remplissage de la table partage
INSERT INTO "PROJET BDR".partage (id_publication, id_utilisateur, id_destination)
VALUES
    (1, 6, 3),
    (2, 1, 3),
    (3, 3, 2),
    (2, 2, 1),
    (5, 2, 2),
    (2, 3, 5),
    (5, 6, 3),
    (6, 7, 4),
    (7, 3, 3),
```

```
    (1, 3, 2),
    (3, 1, 1),
    (4, 1, 3);

-- Remplissage de la table groupe
INSERT INTO "PROJET BDR".groupe (id_groupe, nom_groupe, description,
date_creation, type_groupe)
VALUES
    (1, 'Développeurs', 'Groupe pour les passionnés de programmation.', '2023-01-20',
'public'),
    (2, 'Photographie', 'Amateurs de photographie et de retouche photo.',
'2023-02-01', 'privé'),
(3, 'Jeux vidéo', 'Discussions sur les jeux et l'esport.', '2023-06-20', 'public'),
 (4, 'Voyages', 'Partage d'expériences de voyages.', '2023-07-10', 'public'),
    (5, 'Lecture', 'Club de lecture pour échanger des recommandations.', '2023-08-05',
'privé');


-- Remplissage de la table appartenance
INSERT INTO "PROJET BDR".appartenance (id_utilisateur, id_groupe, role)
VALUES
    (1, 1, 'admin'),
    (2, 1, 'membre'),
    (2, 2, 'admin'),
    (3, 2, 'membre'),
    (3, 3, 'admin'),
    (1, 3, 'membre'),
    (6, 3, 'membre'),
    (7, 4, 'admin'),
    (2, 4, 'membre'),
    (8, 4, 'membre'),
    (9, 5, 'admin'),
    (10, 5, 'membre'),
    (3, 5, 'membre');
```

Update le nombre de like dans les publications pour qu'il soit cohérent avec la table interaction :

```
UPDATE "PROJET BDR".publication p
SET nombre_like = (
    SELECT COUNT(*)
    FROM "PROJET BDR".interaction i
    WHERE i.type_interaction = 'like' AND i.id_publication = p.id_publication
);
```

Ajouter des sous interactions:

```
INSERT INTO "PROJET BDR".interaction (id_interaction, type_interaction,
contenu_commentaire, date_interaction, id_utilisateur, id_publication,
id_interaction_parent)
values (101,'like',null,'2025-01-26',1,2,2),
            (102,'commentaire','Incroyable !','2025-01-26','2',2,2);
```

Une connexion est définie comme ami si les 2 utilisateurs sont connectés entre eux, on doit mettre à jour les données pour s'assurer que tout est en ordre :

```
 UPDATE "PROJET BDR".connexion c1
SET type_connexion =
   CASE
      WHEN EXISTS (
         SELECT 1
         FROM "PROJET BDR".connexion c2
         WHERE c1.id_utilisateur1 = c2.id_utilisateur2
         AND c1.id_utilisateur2 = c2.id_utilisateur1
      ) THEN 'ami'  -- Si connexion réciproque, type 'ami'
      ELSE 'suivi'  -- Sinon, type 'suivi'
   END;
```

V. Analyse Critique du déroulement du projet

A. Répartition du travail :

Pour le projet, nous avons décidé de réaliser ensemble le schéma entité-association (E/A) afin de comprendre les rouages du projet et de notre base de données. De plus, un schéma relationnel a été réalisé par BOULET Olivier et MADANDAR-ARANI Ali. Ils ont également créé le schéma sur PgAdmin de la base de données, puis généré et modifié le code de création de celle-ci.
Une fois la base de données finalisée et validée, MADANDAR ARANI Ali et KARBAL Adam se sont occupés de reformuler la présentation du projet.
Chaque membre du groupe a participé à la rédaction des requêtes :

- requêtes 1 : Haitam
- requêtes 2 et 3 : Ali

- requêtes 4 et 5 : Adam
- requêtes 6 et bonus : Olivier

Suite à cela, chaque membre du groupe a rédigé sa propre requête en choisissant le type de requête proposé dans le sujet.
Madandar-Arani Ali s'est occupé de la mise en page, Boulet Olivier, Karbal Adam et Benabad Haitam ont complété la rédaction du document.

B. Analyse du travail :

La première étape consistait à choisir le sujet. Cependant, cette étape a pris trop de temps alors que ce n'était pas nécessaire. La prochaine fois, il serait plus pertinent de réaliser un tableau des avantages et inconvénients de chaque sujet afin de faire un choix plus rapidement et efficacement.

La seconde étape du projet était la conception de la base de données. Pour ce faire, tous les membres du projet se sont réunis sur un logiciel (Lucid.app) afin de la concevoir. Le travail a été efficace et le résultat satisfaisant. Cependant, en raison du démarrage tardif de l'équipe, nous n'avons pas pu bénéficier des conseils du professeur. Par conséquent, si nous devons refaire cette étape, il serait essentiel de commencer plus tôt.

Une fois la base de données finalisée, nous devions rédiger les questions du sujet. Pour cela, nous avons réparti le travail comme mentionné précédemment. Cette méthode de travail est certes efficace dans un premier temps, mais elle peut entraîner une perte de temps par la suite. En effet, même si nous avons été productifs, nous aurions pu obtenir un meilleur résultat en intégrant les avis de tous ou en nous assurant que chacun comprenne bien l'ensemble du projet. Il est également important de prévoir un temps de débriefing à l'issue de chaque sprint, où chacun explique son travail.

Ainsi, si nous devions recommencer, nous répartirions à nouveau le travail pour optimiser l'efficacité, mais nous incluons également des temps de débriefing pour que chacun puisse expliquer son avancement, poser des questions si nécessaire et suggérer des améliorations.

VI. Requêtes SQL avancées :

A. Requête avec création d'une vue faite par Madandar Arani Ali:

<u>Vue des publications avec le nombre total de commentaires et de likes</u>

```
CREATE OR REPLACE VIEW vue_publication_stat AS
SELECT p.id_publication,
p.contenu,
p.date_publication,
```

```
u.pseudo AS auteur,

(SELECT COUNT(*)
        FROM "PROJET BDR".interaction i
        WHERE i.id_publication = p.id_publication AND i.type_interaction = 'like') AS
nombre_likes,

        (SELECT COUNT(*)
        FROM "PROJET BDR".interaction i
        WHERE i.id_publication = p.id_publication AND i.type_interaction = 'commentaire')
AS nombre_commentaires

FROM "PROJET BDR".publication p
JOIN "PROJET BDR".utilisateur u ON p.id_utilisateur = u.id_utilisateur;

--Affichage de toutes les publications ordonné par le nombre de likes décroissant en utilisant
cette Vue.

SELECT * FROM vue_publication_stat ORDER BY nombre_likes DESC;
```

Cette vue SQL (vue_publication_stat) permet d'afficher toutes les publications avec leur contenu, leur date, et le pseudo de l'auteur. Elle inclut aussi le nombre total de likes et de commentaires en comptant les interactions associées à chaque publication. Une jointure avec la table utilisateur permet d'obtenir le pseudo de l'auteur. Grâce à cette vue, on peut rapidement analyser les publications les plus populaires. Elle simplifie les requêtes en évitant de recalculer ces statistiques à chaque fois.

B. Requête récursive faite par Boulet Olivier:

Les tendances, affiche les publications avec le plus d'interaction dessus en prenant en compte les intéractions filles

```
WITH RECURSIVE recursive_interaction AS (
SELECT i.id_publication,i.id_interaction, i.id_interaction_parent, 1 as
number_interaction
from "PROJET BDR".interaction i
where i.id_interaction_parent is null

UNION ALL

SELECT i.id_publication,i.id_interaction, i.id_interaction_parent,
ri.number_interaction
```

```
from "PROJET BDR".interaction i join recursive_interaction ri on
i.id_interaction_parent = ri.id_interaction
)

SELECT p.id_publication, SUM(i.number_interaction) as nombre
FROM "PROJET BDR".publication p join recursive_interaction i on p.id_publication
= i.id_publication
group by p.id_publication
order by nombre desc
```

Explication :

Dans un premier temps, on récupère les interactions sans parent, c'est-à-dire les interactions principales. Ensuite, grâce à une requête récursive, on ajoute les interactions filles liées aux interactions principales. Enfin, on somme toutes les interactions pour chaque publication et on les classe par popularité.

C. Requête trigger faite par Karbal Adam

Un petit système de notification peut déjà être créé dans notre base de données grâce à l'utilisation de triggers. En effet, on va créer un trigger qui va s'activer à chaque fois qu'on a une nouvelle publication ou une nouvelle interaction.

Tout d'abord, pour tout ce qui est publication, on va définir une fonction destinée à être utilisée comme un trigger qui va afficher un message dans les logs de la console de pgAdmin.
Ensuite, on crée un trigger associé à la fonction qui va s'activer à chaque fois qu'un INSERT sur "PROJET BDR".publication a lieu :

```
CREATE OR REPLACE FUNCTION notifier_publication()
RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE 'Nouvelle publication de l''utilisateur % : %', NEW.id_utilisateur,
NEW.contenu;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_publication
AFTER INSERT ON "PROJET BDR".publication
FOR EACH ROW
EXECUTE FUNCTION notifier_publication();
```

On fait de même pour tous ce qui est interactions :

```
CREATE OR REPLACE FUNCTION notifier_interaction()
RETURNS TRIGGER AS $$
BEGIN
   RAISE NOTICE 'Nouvelle interaction de l''utilisateur % sur la publication %',
NEW.id_utilisateur, NEW.id_publication;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_interaction
AFTER INSERT ON "PROJET BDR".interaction
FOR EACH ROW
EXECUTE FUNCTION notifier_interaction();
```

Maintenant dès qu'on vas insérer de nouvelle valeur on aura une notification comme celle ci :

```
NOTICE:  Nouvelle interaction de l'utilisateur 2 sur la publication 4
```

D. Requête avec la logique conditionnelle faite par Haitam BENABAD

L'utilisation de la logique conditionnelle est particulièrement pertinente dans la gestion des rôles dans un groupe, car elle permet de centraliser plusieurs opérations essentielles dans une seule requête dynamique.

Grâce à la structure "IF...ELSIF...ELSE", le traitement est automatisé en fonction de l'action demandée, évitant ainsi de multiplier les requêtes manuelles. De plus, l'utilisation de "ON CONFLICT (id_utilisateur, id_groupe) DO UPDATE" assure une mise à jour fluide sans risque d'erreur si l'utilisateur est déjà inscrit dans le groupe.

Enfin, cette approche garantit une gestion dynamique et robuste des rôles, essentielle pour éviter les incohérences et assurer une organisation claire des groupes.

Requête 1: Lister les membres d'un groupe avec leurs rôles

```
SELECT u.pseudo AS nom_utilisateur, a.role, g.nom_groupe
FROM "PROJET BDR".appartenance a
JOIN "PROJET BDR".utilisateur u ON a.id_utilisateur = u.id_utilisateur
JOIN "PROJET BDR".groupe g ON a.id_groupe = g.id_groupe
WHERE g.id_groupe = 1;  -- Remplace 1 par l'ID du groupe voulu
```

Cette requête récupère la liste des membres d’un groupe donné ainsi que leurs rôles dans ce groupe.

Elle joint trois tables :Appartenance, Utilisateur et Groupe

Requête 2: Ajouter un utilisateur à un groupe (ou mettre à jour son rôle)

```
INSERT INTO "PROJET BDR".appartenance (id_utilisateur, id_groupe, role)
VALUES (1, 2, 'Admin')  -- Remplace 1 et 2 par les IDs appropriés
ON CONFLICT (id_utilisateur, id_groupe)
DO UPDATE SET role = EXCLUDED.role;
```

Cette requête ajoute un utilisateur à un groupe avec un rôle spécifique.

Si l'utilisateur existe déjà dans le groupe, son rôle sera mis à jour grâce à ON CONFLICT sur (id_utilisateur, id_groupe), ce qui empêche d'insérer un doublon.

Requête 3: Supprimer un utilisateur d’un groupe s'il n'a plus de rôle actif

```
DELETE FROM "PROJET BDR".appartenance
WHERE id_utilisateur = 1 AND id_groupe = 2; -- Remplace par les IDs correspondants
```

Cette requête supprime un utilisateur spécifique (id_utilisateur = 1) d'un groupe (id_groupe = 2).

Elle est utile pour retirer un utilisateur d'un groupe si son rôle n'est plus nécessaire.

Requête 4: Afficher tous les groupes où un utilisateur est administrateur

```
SELECT g.nom_groupe
FROM "PROJET BDR".appartenance a
JOIN "PROJET BDR".groupe g ON a.id_groupe = g.id_groupe
WHERE a.id_utilisateur = 1 AND a.role = 'Admin'; -- Remplace 1 par l'ID de l'utilisateur
```

Cette requête liste tous les groupes où un utilisateur donné (id_utilisateur = 1) possède le rôle "Admin".

Elle utilise une jointure pour récupérer les noms des groupes.

Requête tout-en-un avec logique conditionnelle

Cette requête permet de gérer dynamiquement les actions (ajouter, mettre à jour ou supprimer) en fonction d'une variable.

```
-- Requête tout-en-un avec logique conditionnelle

DO $$
DECLARE
   action TEXT := 'ajouter';  -- Changer en 'mettre à jour' ou 'supprimer'
   v_id_utilisateur INTEGER := 3;
   v_id_groupe INTEGER := 2;
   v_role TEXT := 'moderateur';
BEGIN
   IF action = 'ajouter' THEN
      INSERT INTO "PROJET BDR".appartenance (id_utilisateur, id_groupe, role)
      VALUES (v_id_utilisateur, v_id_groupe, v_role)
      ON CONFLICT (id_utilisateur, id_groupe)
      DO UPDATE SET role = EXCLUDED.role;

   ELSIF action = 'mettre à jour' THEN
      UPDATE "PROJET BDR".appartenance
      SET role = v_role
      WHERE id_utilisateur = v_id_utilisateur
        AND id_groupe = v_id_groupe;

   ELSIF action = 'supprimer' THEN
      DELETE FROM "PROJET BDR".appartenance
      WHERE id_utilisateur = v_id_utilisateur
        AND id_groupe = v_id_groupe;

   ELSE
      RAISE NOTICE 'Action inconnue';
   END IF;
END $$;
```

Cette procédure exécute une action en fonction de la valeur de la variable action. Trois actions sont possibles :

- Ajouter un utilisateur à un groupe (INSERT ... ON CONFLICT pour gérer le cas où il est déjà présent).
- Mettre à jour le rôle d'un utilisateur dans un groupe (UPDATE).
- Supprimer un utilisateur d'un groupe (DELETE).

Si l’action ne correspond à aucune des trois options, un message 'Action inconnue' est affiché.
→ Ces requêtes permettent de gérer les rôles des utilisateurs dans des groupes de manière efficace.
→ Elles couvrent les cas les plus courants : affichage, ajout, mise à jour, suppression, et gestion dynamique.
→ L'utilisation de ON CONFLICT et de logique conditionnelle optimise la manipulation des données.

VII. Point d’amélioration (Bonus)

Ajout de plusieurs contraintes pour améliorer la base de données :

-Automatisation de la mise à jour des connexions passant de ‘suivi’ à ’ami’ lorsque la connexion devient réciproque uniquement.
-Automatisation de la mise à jour des likes dans la table publication.
-Empêcher une personne d’envoyer une publication dans un groupe auquel elle n’appartient pas.