ST207 project: Social Media Application

1. LSE Candidate Numbers
   36518, 26002, 32729

2. Description of the chosen topic / scenario

Black = entities and their attributes
Red = relationships among entities
Blue = Constraints to be made
Green = usage operations.

Our chosen scenario is a social media application. It has a User_Profile entity to describe individual profiles. They can follow or block each other. There is a Groups entity, of which User_Profiles can be members, or be banned from. There is a Posts entity to describe the posts User_Profiles make, and there is a Comments entity to describe the comments User_Profiles make on each post. User_Profiles can also react to posts and comments.

USER_PROFILE
User_Profiles are identified by a unique id attribute. User_Profiles must be further described by the unique attributes username and email, and the non-unique attributes first_name, last_name, gender, privacy, password, and date_of_birth. User_Profiles can also be further described by the attributes phone, country, and biography. Each User_Profile can be a member of many Groups, and publish many Posts and Comments. Each User_Profile can also follow many other User_Profiles, and can react to many Posts and Comments. Each User_Profile can also block other User_Profiles, and they can also be banned from Groups. If User_Profile_1 follows User_Profile_2, it does not necessarily mean that User_Profile_2 must follow User_Profile_1.

FOLLOWERS / FOLLOWING
A User_Profile can follow many other User_Profiles, and this is represented in two different many to many relationships. One is called Following and the other is called Followers. The Following table has tuples which relate to a User_Profile, and the User_Profile they want to follow / are following. The Followers table has tuples which relate to a User_Profile, and the User_Profile which wants to / is following them. The reason for two different tables describing the same many to many relationship is to implement two different clustering indexes. Only one clustering index can exist per table, and clustering indexes are very efficient for querying with aggregate functions. Aggregate functions over the Follower / Following tables would be very common, for example, how many followers does this User_Profile have? The index over the Following table will make queries looking for those who a User_Profile follows more efficient. The index over the Followers table will make queries looking for the followers of a certain User_Profile more efficient.

FOLLOWING
Following must be described by a composite key which is made up of the ids of two User_Profiles, one called user_profile_id which relates to the User_Profile doing the following, and one called following_id which relates to the User_Profile who is being followed. It must also be described by the attribute 'accepted' which has a Boolean value. If accepted is set to False, it means that the User_Profile which relates to the following_id has not yet accepted the follow request from the User_Profile relating to the user_profile_id. If accepted is set to True, it means that the follow request has been accepted. If the privacy attribute of a User_Profile is set to FALSE, then any tuples in the Following table which has the id of this User_Profile as the following_id must have the 'accepted' attribute set to TRUE. This is because if the privacy attribute of a User_Profile is set to FALSE then any follow requests are automatically accepted. A User_Profile cannot be a follower or be following themselves. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

FOLLOWERS
Followers must be described by a composite key which consists of the ids of two User_Profiles, one called user_profile_id which relates to the User_Profile being followed, and one called follower_id which relates to the

User_Profile doing the following. It must also be described by the 'accepted' attribute. The constraints on the 'accepted' attribute is the same as in the Following table. Furthermore, if there is an entry into the Following table, a reciprocal entry must be made into the Followers table, and vice versa. Similarly, if there is any update made to the 'accepted' attribute in one of these tables, the reciprocal relationship in the other table must also be updated. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

BLOCKED

The many to many relationship which describes blocked User_Profiles is represented in the Blocked table. Tuples in this table must be described by a composite key which consists of the ids of two User_Profiles, one called user_profile_id which relates to the User_Profile doing the blocking, and one called blocked_id which relates to the User_Profile who is getting blocked. Immediately after a relationship between two User_Profiles is made through the Blocked_Profiles table, any relationship between the same two User_Profiles in the Following and Followers tables must be removed. A User_Profile cannot block themselves. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

GROUPS

Groups are identified by a unique id attribute. Groups must be further described by the unique groupname attribute. Groups must be further described by the attributes creator_id, title, and description. The creator_id attribute is a foreign key which refers to the id of the User_Profile who created the Group. Groups can have many User_Profiles as members. Groups can also have many Posts. Groups can ban certain User_Profiles. As soon as the Group is created, the User_Profile which relates to the creator_id attribute of the Group must be made a member with admin status. A group must have at least one member who is an admin at any one time. If a User_Profile is deleted, and they are the only member with admin status to that Group, the Group must also be deleted. When a Group is deleted, all tuples in other tables where the id of this Group is a foreign key must also be deleted. If a User_Profile is deleted, all tuples containing its id as a foreign key must have the creator_id attribute set to NULL only if there is another admin of the group. If that User_Profile was the only admin, then the group must be deleted.

GROUP_MEMBERS

The many to many relationship between Groups and User_Profiles is represented in the Group_Members table. Tuples in this table must be described by a composite key which consists of the id of a User_Profile and the id of a Group. They must be further described by the Boolean attribute admin which determines whether that member has admin rights for that group. A User_Profile cannot become a member of a Group if they have been banned. If a Group_Member is deleted, as that member was the only admin of a Group, the corresponding Group must also be deleted. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

BANNED_PROFILES

The many to many relationship between Groups and User_Profiles who are banned is represented in the Banned_Profiles table. Tuples in this table must be described by a composite key which consists of the id of the User_Profile who is banned, and the id of the Group they are banned from. As soon as a User_Profile is banned from a group, they must be removed as a member. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

POSTS

Posts are identified by a unique id attribute. Posts must be further described by the id attribute of the User_Profile who created the post as a foreign key, because Posts can only be created by one User_Profile. They must also have the attributes content, and created_on. Each Post can be reacted to by many User_Profiles, and they can have many Comments made by User_Profiles. Each Post can also be related to a Group, in which case, it is a group post. When a Post is deleted, all tuples in other tables where the id of this Post is a foreign key must also be deleted. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

GROUP_POSTS

The many to many relationship between Groups and Posts is represented in the Group_Posts table. Tuples in this table must be described by a composite key which consists of the id of the Post and the id of the Group it relates to.

POST_REACTIONS

Post reactions can be described by a many to many relationship between Posts and User_Profiles in the Post_Reactions table. Tuples in this table must be described by a composite key which consists of the id of the Post and the id of the User_Profile who is reacting. They must also be further described by the reaction_type attribute, which determines whether the User_Profile reacted with a 'like', 'love', or 'dislike'. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

COMMENTS

Comments are identified by a unique id attribute. Comments must be further described by the id attribute of the User_Profile who made the comment, and by the id attribute of the Post which the comment was made on. Comments must also have the attributes content and created_on. Each Comment can be reacted to by many User_Profiles. When a Comment is deleted, all tuples in other tables where the id of this Comment is a foreign key must also be deleted. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

COMMENT_REACTIONS

The Comment_Reactions table is similar to the Post_Reactions table, but references the id of Comments instead of Posts. If a User_Profile is deleted, all tuples containing its id as a foreign key must also be deleted.

USAGE OPERATIONS

Common everyday usage operations would include queries for how many followers a specific User_Profile has, how many reactions a specific Post has, which Groups is a specific User_Profile banned from, what are the details of the members of a specific group and which members are admins. The social media application may also want to implement an algorithm to decide which Posts should be seen first in order to increase the retention of User_Profiles, in which case querying for the Posts which have the highest number of reactions would be useful. For these common a selection of views and clustered indexes have been created.

Common update operations would include accepting a follow request (changing the accepted attribute to TRUE in the Followers / Following tables), updating a User_Profile's personal details, changing the admin status of a Group_Member, changing the privacy setting of a User_Profile, and changing the reaction_type attribute in the Post_Reactions and Comment_Reactions tables.

Common deletion operations would include deleting a User_Profile, which would require the deletion of all tuples in other tables which contain the id of that User_Profile as a foreign key, apart from in the Groups table, where the creator_id attribute would be set to NULL provided that there is another admin of that group. Tuples from the Followers / Following tables would also be a common deletion, along with tuples from the Banned_Profiles table.

Common insertion operations would include inserting new tuples into the Followers / Following tables, Blocked table, and Banned_Profiles table.

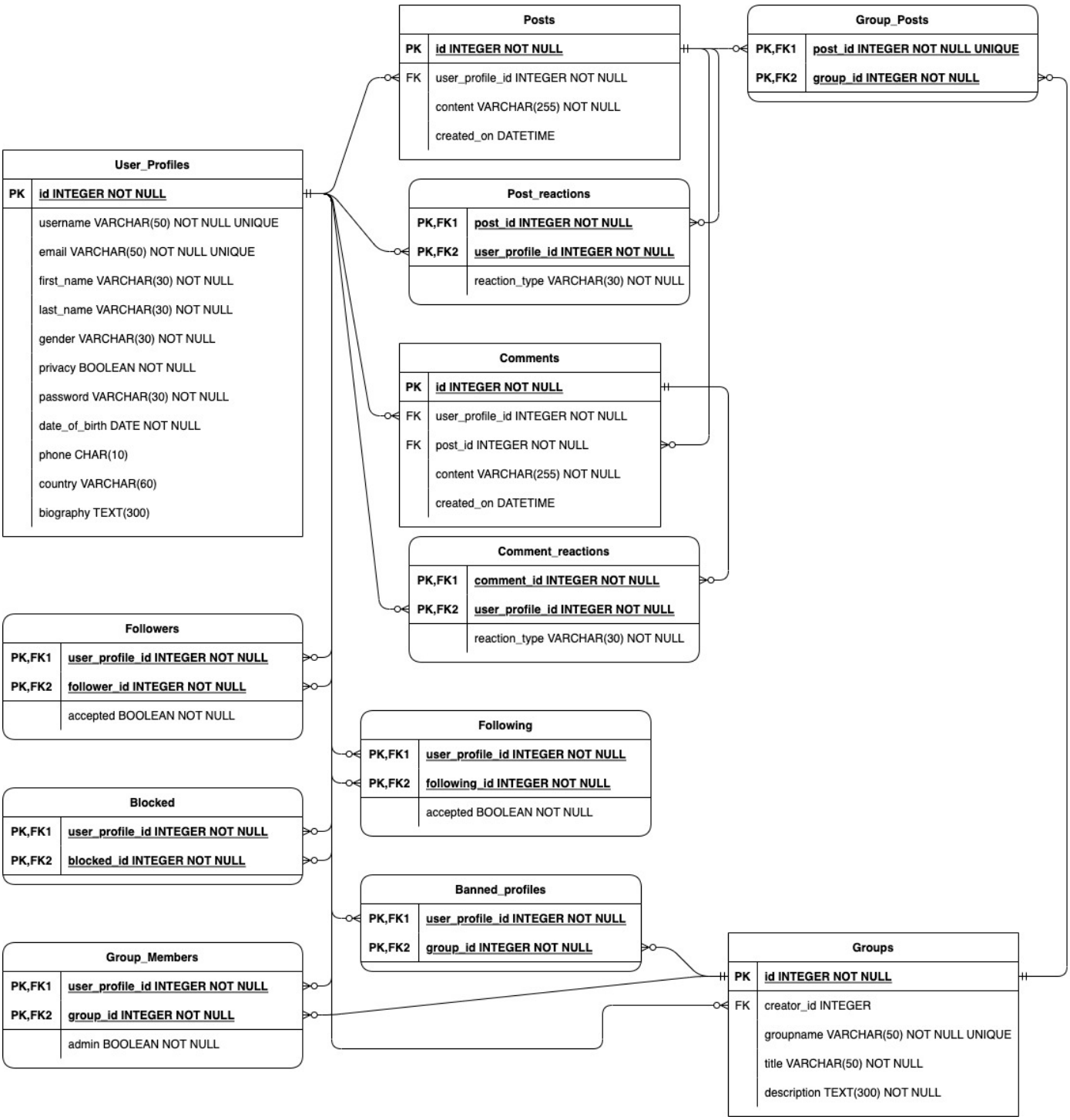
3. Description of data

We have generated synthetic data from https://www.mockaroo.com/ for all of the tables except from the Following table. This is because the reciprocal relationship of the data inserted into the Followers table is automatically inserted into the Following table. The synthetic data is stored in separate files which relate to the respective tables in the synthetic_data folder.

4. Justification of database technology

Our focus for this project has been to explore how data can be managed and how data integrity can be preserved in a database via constraints and triggers. SQL databases are very adept in this regard as ACID properties are built into database design. SQL databases are also very mature technology that has been become refined over the years and thus, it may be better optimised than newer NoSQL systems. This means our application which largely relies on retrieving data (posts,comments etc.), can execute queries much faster. Therefore, we chose to create a Relational Database.

5. ER diagram


Posts
PK id INTEGER NOT NULL
FK user_profile_id INTEGER NOT NULL
content VARCHAR(255) NOT NULL
created_on DATETIME
Group_Posts
PK,FK1 post_id INTEGER NOT NULL UNIQUE
PK,FK2 group_id INTEGER NOT NULL
User_Profiles
PK id INTEGER NOT NULL
username VARCHAR(50) NOT NULL UNIQUE
email VARCHAR(50) NOT NULL UNIQUE
first_name VARCHAR(30) NOT NULL
last_name VARCHAR(30) NOT NULL
gender VARCHAR(30) NOT NULL
privacy BOOLEAN NOT NULL
password VARCHAR(30) NOT NULL
date_of_birth DATE NOT NULL
phone CHAR(10)
country VARCHAR(60)
biography TEXT(300)
Post_reactions
PK,FK1 post_id INTEGER NOT NULL
PK,FK2 user_profile_id INTEGER NOT NULL
reaction_type VARCHAR(30) NOT NULL
Comments
PK id INTEGER NOT NULL
FK user_profile_id INTEGER NOT NULL
FK post_id INTEGER NOT NULL
content VARCHAR(255) NOT NULL
created_on DATETIME
Comment_reactions
PK,FK1 comment_id INTEGER NOT NULL
PK,FK2 user_profile_id INTEGER NOT NULL
reaction_type VARCHAR(30) NOT NULL
Followers
PK,FK1 user_profile_id INTEGER NOT NULL
PK,FK2 follower_id INTEGER NOT NULL
accepted BOOLEAN NOT NULL
Following
PK,FK1 user_profile_id INTEGER NOT NULL
PK,FK2 following_id INTEGER NOT NULL
accepted BOOLEAN NOT NULL
Blocked
PK,FK1 user_profile_id INTEGER NOT NULL
PK,FK2 blocked_id INTEGER NOT NULL
Banned_profiles
PK,FK1 user_profile_id INTEGER NOT NULL
PK,FK2 group_id INTEGER NOT NULL
Group_Members
PK,FK1 user_profile_id INTEGER NOT NULL
PK,FK2 group_id INTEGER NOT NULL
admin BOOLEAN NOT NULL
Groups
PK id INTEGER NOT NULL
FK creator_id INTEGER
groupname VARCHAR(50) NOT NULL UNIQUE
title VARCHAR(50) NOT NULL
description TEXT(300) NOT NULL

6. Textual description of all operations and corresponding outputs

QUERIES

Query 1: How many followers does a random specific User_Profile have? For example, the User_Profile with the username 'rgadesby0'?

OUTPUT:

```
QUERY 1
   USER_NAME  number_of_followers
0  rgadesby0                   1
```

Query 2: Which User_Profile has the highest number of followers?

OUTPUT:

```
QUERY 2
   USER_ID   USER_NAME  highest_number_of_followers
0       95  evolleth2m                           4
```

Query 3: Which Group has the highest number of members?

OUTPUT:

```
QUERY 3
   GROUP_ID  GROUP_NAME  NUMBER_OF_MEMBERS
0        49  ethorlby1c                  5
```

Query 4: How many reactions does a specific Post have? (Where the id of the Post = 53?)

OUTPUT:

```
QUERY 4
   POST_ID  NUMBER_OF_REACTIONS
0       53                    7
```

Query 5: What are the details of the Posts which have the top 3 number of reactions?

OUTPUT:

POST_ID	CREATOR_ID	POST_CONTENT	DATE_CREATED
13	97	pharetra magna vestibulum aliquet ...	2022-01-21 10:36:3
28	49	gravida sem praesent id massa id nis...	2022-01-21 10:36:3
53	31	interdum in ante vestibulum ante ...	2022-01-21 10:36:3

Query 6: List how many Posts have been created by User_Profiles in the top three countries in descending order.

OUTPUT:

```
QUERY 6
   NUMBER_OF_POSTS COUNTRY
0               16   China
1                7  Brazil
2                6    Peru
```

Query 7: Which Groups is a specific User_Profile banned from? (For example, where the id of the User_Profile = 1?)

OUTPUT:

```
QUERY 7
   GROUP_ID GROUP_NAME  USER_ID
0        32  cmorillav        1
```

Query 8: What are the details of the members of a specific group, and which members are admins? (For example, the Group with the id of 23?)

OUTPUT:

```
     USER_NAME  ADMIN_STATUS  ...  USER_DATE_OF_BIRTH USER_COUNTRY
0  btoffoloni1d             1  ...          2007-07-15      Finland
1     zjubert1m             0  ...          1978-03-09  Netherlands
2     cechalied             0  ...          1992-09-27     Colombia
3     ariddle29             0  ...          1995-06-15      Iceland
```

Query 9: For the groups where a specific User_Profile is an admin, return how many admins there are for each of those groups? (For example, where the id of the User_Profile = 1?)

OUTPUT:

```
QUERY 9
         group_name  NO_OF_ADMINS
0       lhonnicotty             2
1         mdymond1i             3
2  cchristoffels1r              2
```

UPDATES

Update 1: Updating a follow request in the Followers table so that it is accepted between the User_Profiles with ids 85 and 80. When accepted = TRUE it is represented by a 1; when it is FALSE it is represented by a 0. Due to TRIGGER 13, the reciprocal relationship in the Following table will also be updated.

OUTPUT:

```
FOLLOWERS TABLE
   user_profile_id  follower_id  accepted
0               85           80         0

FOLLOWING TABLE
   user_profile_id  following_id  accepted
0               80            85         0

UPDATED FOLLOWERS TABLE
   user_profile_id  follower_id  accepted
0               85           80         1

UPDATED FOLLOWING TABLE
   user_profile_id  following_id  accepted
0               80            85         1
```

Update 2: Updating the phone number of a User_Profile where the id = 1.

OUTPUT:

```
USER_PROFILES TABLE
        phone
0  5898348601

UPDATED USER_PROFILES TABLE
        phone
0  0755795833
```

Update 3: Changing the admin status of a group member from FALSE (represented as 0) to TRUE (represented as 1), where the user_profile_id = 24 and group_id = 66.

OUTPUT:

```
GROUP_MEMBERS TABLE
   user_profile_id  group_id  admin
0               24        66      0

UPDATED GROUP_MEMBERS TABLE
   user_profile_id  group_id  admin
0               24        66      1
```

Update 4: Changing the privacy setting of a User_Profile where the id = 40.

OUTPUT:

```
USER_PROFILES TABLE
   id  privacy
0  40        0

UPDATED USER_PROFILES TABLE
   id  privacy
0  40        1
```

Update 5: Updating the reaction_type attribute on a reaction to a post in the Post_Reactions table where the post_id = 76 and the user_profile_id = 91.

OUTPUT:

```
POST_REACTIONS TABLE
   post_id  user_profile_id reaction_type
0       76               91          love

UPDATED POST_REACTIONS TABLE
   post_id  user_profile_id reaction_type
0       76               91       dislike
```

DELETIONS

Deletion 1: Delete a relationship from the Follower table where user_profile_id = 11 and follower_id = 62. Due to TRIGGER 16, the reciprocal relationship will also be deleted from the Following table.

OUTPUT:

```
FOLLOWERS TABLE
   user_profile_id  follower_id  accepted
0               11           62         1

FOLLOWING TABLE
   user_profile_id  following_id  accepted
0               62           11         1

FOLLOWERS TABLE AFTER DELETION
Empty DataFrame
Columns: [user_profile_id, follower_id, accepted]
Index: []

FOLLOWING TABLE AFTER DELETION
Empty DataFrame
Columns: [user_profile_id, following_id, accepted]
Index: []
```

Deletion 2: Delete a tuple from the Banned_Profiles table where user_profile_id = 16 and group_id = 1.

OUTPUT:

```
BANNED_PROFILES TABLE
   user_profile_id  group_id
0               16         1

BANNED_PROFILES TABLE AFTER DELETION
Empty DataFrame
Columns: [user_profile_id, group_id]
Index: []
```

Deletion 3: Delete a tuple from the User_Profiles table where id = 45. Due to many TRIGGERS, many other tuples in tables where the id of the User_Profile is a foreign key will also be deleted. This prevent a foreign key violation.

OUTPUT:

```
USER_PROFILES TABLE
   id    username
0  45  rmcgooch18

POST_REACTIONS TABLE
   post_id  user_profile_id reaction_type
0       65               45          love
1       28               45          love

GROUP_MEMBERS TABLE
   user_profile_id  group_id  admin
0               45        85      1

USER_PROFILES TABLE AFTER DELETION
Empty DataFrame
Columns: [id, username]
Index: []

POST_REACTIONS TABLE AFTER DELETION
Empty DataFrame
Columns: [post_id, user_profile_id, reaction_type]
Index: []

GROUP_MEMBERS TABLE AFTER DELETION
Empty DataFrame
Columns: [post_id, user_profile_id, reaction_type]
Index: []
```

Deletion 4: Delete a Group_Member which is the only admin of a Group, for example, where user_profile_id = 71 and group_id = 1. The Group_Member will be deleted, along with the corresponding Group because a Group cannot exist without an admin.

OUTPUT:

```
GROUP_MEMBERS TABLE
   user_profile_id  group_id  admin
0               71         1      1

GROUPS TABLE
   id  creator_id groupname  title                                          description
0   1          71  dupcott0  nulla  turpis elementum ligula vehicula consequat mor...

GROUP_MEMBERS TABLE AFTER DELETION
Empty DataFrame
Columns: [user_profile_id, group_id, admin]
Index: []

GROUPS TABLE AFTER DELETION
Empty DataFrame
Columns: [id, creator_id, groupname, title, description]
Index: []
```

INSERTIONS

Insertion 1: Inserting into the Blocked table where user_profile_id = 85, and the blocked_id = 80. Any Relationship between these two ids in the Following and Followers table must be removed.

OUTPUT:

```
FOLLOWERS TABLE
   user_profile_id  follower_id  accepted
0               85           40         0
1               85           80         1

FOLLOWING TABLE
   user_profile_id  following_id  accepted
0               80            85         1
1               40            85         0

FOLLOWERS TABLE AFTER INSERTION INTO BLOCKED
   user_profile_id  follower_id  accepted
0               85           40         0

FOLLOWING TABLE AFTER INSERTION INTO BLOCKED
   user_profile_id  following_id  accepted
0               40            85         0
```

Insertion 2: After inserting a User_Profile into the Banned_Profiles table where user_profile_id = 78 and group_id = 2, they should be removed as a member if they are one.

OUTPUT:

```
GROUP_MEMBERS TABLE
   user_profile_id  group_id  admin
0               78         2      0

BANNED_PROFILES TABLE
Empty DataFrame
Columns: [user_profile_id, group_id]
Index: []

GROUP_MEMBERS TABLE AFTER INSERTION INTO BANNED PROFILES
Empty DataFrame
Columns: [user_profile_id, group_id, admin]
Index: []

BANNED_PROFILES TABLE AFTER INSERTION INTO BANNED PROFILES
   user_profile_id  group_id
0               78         2
```

Insertion 3: Inserting a tuple into the Following table where user_profile_id = 1 and following_id = 16, with the accepted attribute set to FALSE (represented as a 0). However, because the privacy attribute of the User_Profile relating to the following_id foreign key is set to FALSE, the accepted attribute will be set to TRUE (represented as a 1) because of TRIGGER 14. This relationship will then be updated in the Followers table as well.

OUTPUT:

```
USER_PROFILES TABLE
   id username  privacy
0  16  pwyeldf        0

INSERTING INTO FOLLOWING TABLE WITH ACCEPTED ATTRIBUTE SET TO FALSE

FOLLOWING TABLE AFTER INSERTION INTO FOLLOWING TABLE
   user_profile_id  following_id  accepted
0                1            16         1

FOLLOWERS TABLE AFTER INSERTION INTO FOLLOWING TABLE
   user_profile_id  follower_id  accepted
0               16            1         1
```