Assignment 3 Report

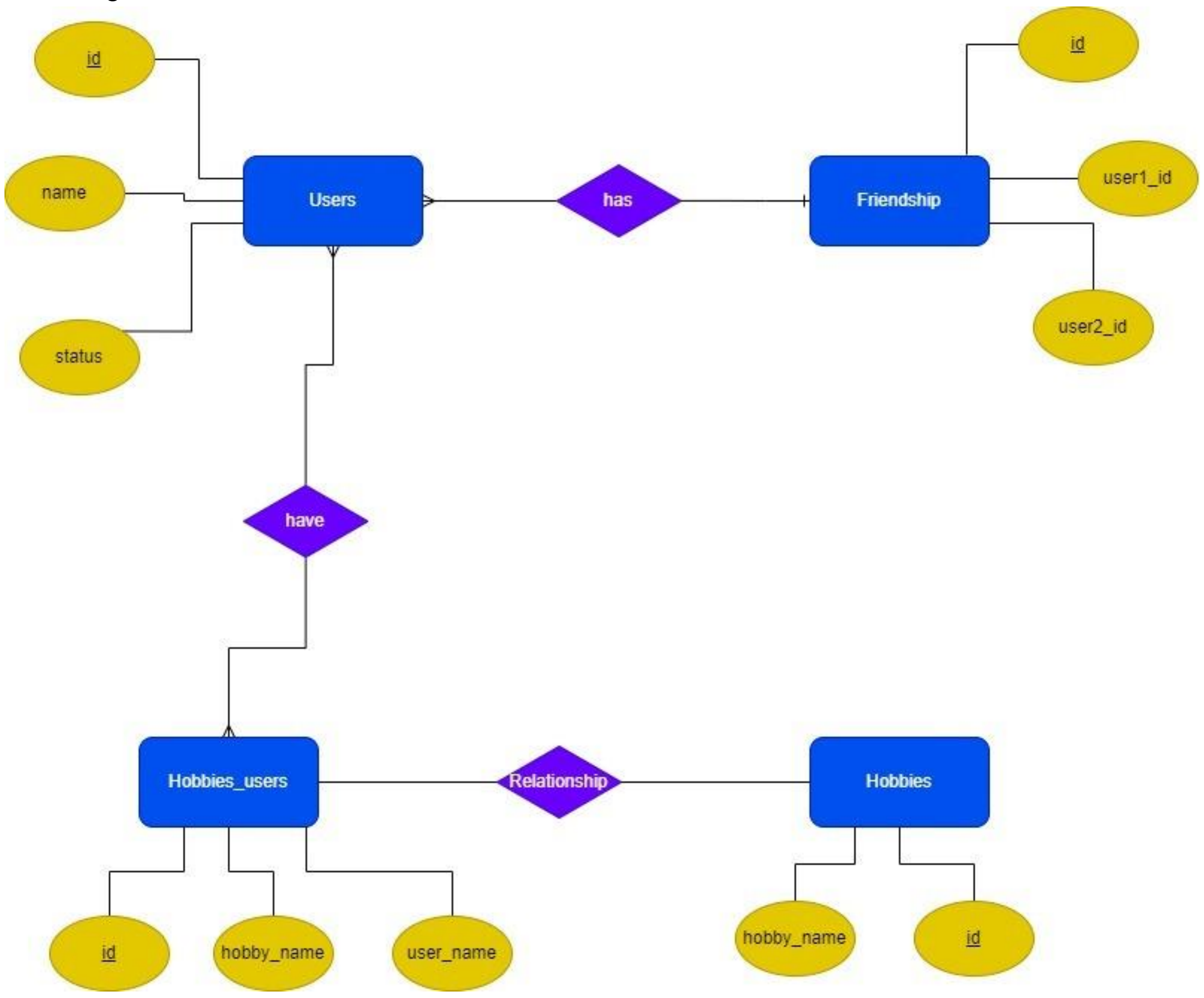
Student(s): Mohammadali Rashidfarokhi – mr223jp@student.lnu.se
Amirhossein Soltaninejad – as225rr@student.lnu.se

1. Project Idea

Regarding this assignment, we have decided to create and implement a social media API / SCRAPER system.
Moving on, the problem that we intend to solve is that users, due to the high number of friends, lose track of their friends, hobbies. Moreover, to display who is the best match in case a user wants to make new friends or expand their network. Therefore, we have decided to create a program to **track** and **analyze** users' friends' data.
Our application will be a console application that will enable the user with different options to choose from them. As a result, the meant user of our designed program is the user who is registered and has friends.
Based on the above system, our proposed solution is appropriate for people who have social media accounts who wish to analyze and have a better view of who they are friends with on social media or with whom they are willing to be friends.
Moreover, our proposal for social media Analyzer will handle the information of the users, their friends, and their hobbies, which will make it easier for the user to have a better and clearer view of their social media. Apart from that, a user can get the information of all registered users, their friends based on the user's name, all users with how many friends they have.
In addition, the user can check friends of friends for a specific user and the most popular user (has most of the friends). The user also can get a list of Suggest Friends; it is based on friends of friends.

2. Logical Model

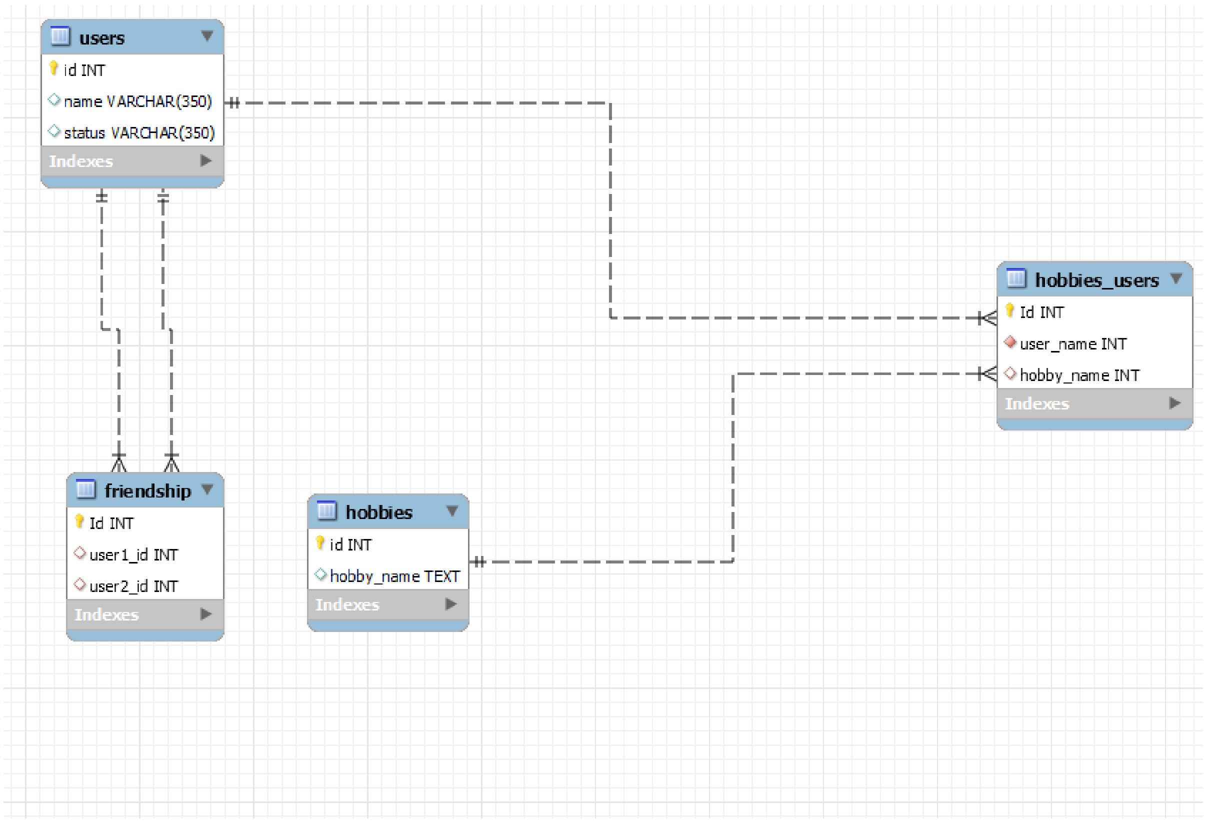
E/R Diagram:



- Since our proposal is social media analyzer, it is critical to have users and hobbies as entity sets.
- Therefore, the user entity set has been included by id, name, and status. The hobbies entity set has been included by id and hobby_name. We have added two more entity sets ("hobbies_users”, "friendship”) to be able to manage the users and hobbies as well as the friendship and connect them correctly.
- To be able to retrieve users' info data uniquely, we have decided to use users.id and hobbies.id together as a key for the hobbies_users info entity set.

Regarding the relation between Users and hobbies entity sets through hobbies_users, the relation has been set as many to many.
As a result, a user who is registered can have different hobbies. On the other hand, different users can have the same hobby.
Considering the relation between users and friendship entity sets, the relation is also one to many. What it means is that one user from a user's table can have multiple friendships. Therefore, "friendship" may have more than users.id of the same user.

3. Design in SQL



❖ A relationship exists between the users table and the friendship, we can give each existing user a friendship with another existing user, that can be randomly or according to the mutual friends between two users (suggestFriends).

❖ A relationship exists between the hobbies_users table, the users table, and the hobbies table; we can give each existing user a hobby by getting the id of the hobby and inserting it into hobbies_users table. This will give as the ability to suggest friends for the user according to their hobbies as well as the mutual friends.

The relationship between hobbies_users table and hobbies exists to connect the user with the hobby name.

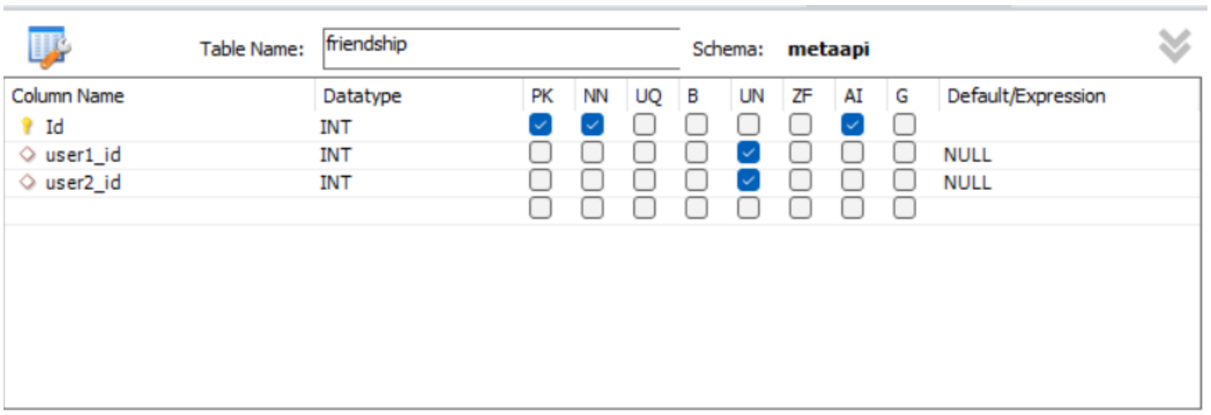
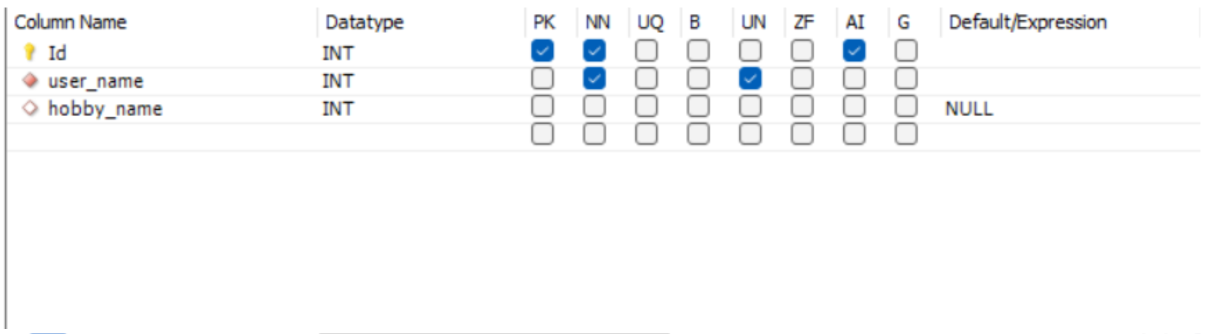


Table Name: friendship Schema: **metaapi**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
Id	INT	✓	✓					✓		
user1_id	INT					✓				NULL
user2_id	INT					✓				NULL



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
Id	INT	✓	✓	☐	☐	☐	☐	✓	☐	
user_name	INT	☐	✓	☐	☐	✓	☐	☐	☐	
hobby_name	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
		☐	☐	☐	☐	☐	☐	☐	☐	

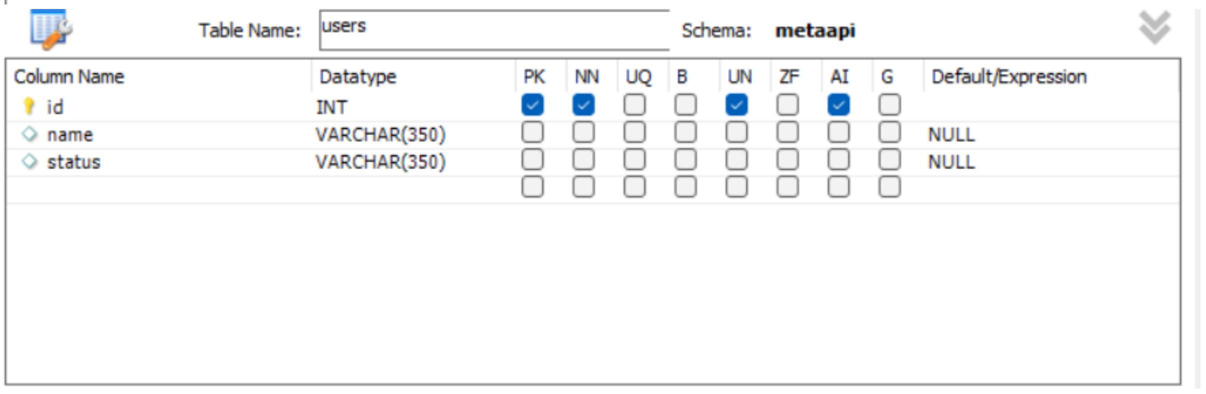


Table Name: users Schema: **metaapi**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	✓	✓	☐	☐	✓	☐	✓	☐	
name	VARCHAR(350)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
status	VARCHAR(350)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
		☐	☐	☐	☐	☐	☐	☐	☐	

4. SQL Queries

Inserting user into users' table:

```
INSERT INTO `users`(`name`, `status`) VALUES (?,?)
```

All existing users and their hobbies:
We start by selecting the needed properties which are name and hobby name, we join both hobbies and users tables, we get the hobby name from the hobbies table using the foreign key from hobbies_users and we get the user name from the users table using the foreign key from hobbies_users. The results are ordered by users.name to get the results in alphabetical order.

```
SELECT name, hob.hobby_name
FROM hobbies_users
        JOIN hobbies as hob
             ON hobbies_users.hobby_name = hob.id
        JOIN users as us
             ON user_name = us.id
ORDER BY us.name, hob.hobby_name
```

all existing users who have more than 2 friends:
As we told before users table and friendship tables has a relationship. Therefore, it is why we start by selecting the name of the user as well as the number of the user's friends but to be able to count the number of friends we need to join both users and friendship tables where the users.id equals the id in the friendship table.

```
SELECT users.name, COUNT(friendship.user2_id) as BFF
FROM friendship
       JOIN users
           ON users.id = friendship.user2_id
GROUP BY users.name
HAVING BFF > 1
ORDER BY users.name
```

Specific user's friends of friends:
We start by selecting the needed properties, which are the name and count of friends, We join both users and friendship tables where the users.id equals the id in the friendship table and then we specify that we only need the users that are friends with a specific user.

```
SELECT us.name AS KFL
FROM friendship
         INNER JOIN users as us
                    ON us.id = friendship.user2_id
WHERE friendship.user2_id
          IN
       (SELECT user2_id
        FROM friendship
        WHERE user1_id  = ? )
GROUP BY user2_id;
```

The user who has the most friends:
This is a view that returns the most popular user. We select both name and count of total friends by using the existing relationship between friendship and users tables.

```
CREATE VIEW mustPopularUser AS
SELECT users.name, COUNT(friendship.user2_id) as Popular
FROM friendship
       JOIN users
           ON users.id = friendship.user2_id
GROUP BY users.name
ORDER BY COUNT(friendship.user2_id) DESC
LIMIT 1;

SELECT * from mustPopularUser
```

Suggest Friends (Mutual Friends):
We start by selecting the needed properties which are names, then we look into the friendship table on the users that are friends with the user's friend.

```
SELECT name
FROM users
WHERE id NOT IN
     (SELECT user2_id
      FROM friendship
      WHERE user1_id = ?)
 AND id IN
     (SELECT user2_id
      FROM friendship
      WHERE user1_id IN
          (SELECT user2_id
           FROM friendship
           WHERE user1_id = ?))
 AND id != 7
```

Suggest Friends (Mutual Hobby):
We start by selecting the needed properties which are names, then we investigate the friendship table on the users that are friends with the users and have the same hobby.

```
SELECT name
FROM users
WHERE id IN
     (SELECT user2_id
      FROM friendship
      WHERE user1_id = ?)
 AND id IN
     (SELECT user_name
      FROM hobbies_users
      WHERE hobby_name IN
          (SELECT hobby_name
           FROM hobbies_users
           WHERE user_name = ?));
```

5- Discussion and Resources

How to run the program:
1-Please, put the Java files and the SQL file available in the src folder on GitLab, into the "src" folder of your preferred IDE.
2- Add the libraries provided in the "Libraries" folder to the library classpath of your IDE.
3-To run the program, please run the "Main.java" class.
Important note regarding running: If you go to the "Main" class, in line 7 "DatabaseInitializer.run(args);" that initializes the database and it will insert already provided data to the database. However, if you run the program again, it will drop out of the existing metaAPI database. On the other hand, if you put this line in the comment, whatever data has been inserted before, will not be removed. Therefore, **For the first initialization, line 7 should be uncommented. And For the second and later program running, line 7 on the Main class should be commented.**
Source code: https://gitlab.lnu.se/mr223jp/database-theory
Video demonstration: https://youtu.be/ZQLXybdRVFI

Appendix 1

Changelog

Student:	**Task**	**Date**
MohammadAli	Setting up the Git repository	2022/01/13
Amirhossein	Setting-up MAMP and MySQL Workbench	2022/01/13
MohammadAli	Bringing up the project idea and describing it	2022/01/15
Amirhossein	Designing a logical model for our design	2022/01/16
MohammadAli	Designing the SQL entities and relationships	2022/01/19
Amirhossein	implementing the SQL codes needed for the program	2022/01/23
MohammadAli	implementing the program Java code, the classes ()	2022/01/28
Amirhossein	implementing the program Java code, the classes ()	2022/01/28
Amirhossein and MohammadAli	Implementing the 5 SQL queries	2022/02/01
MohammadAli	Documenting tasks 1, 2, and 3.	2022/02/03
Amirhossein	Documenting tasks 4 and 5.	2022/02/07
MohammadAli	Making the Video presentation.	2022/02/25