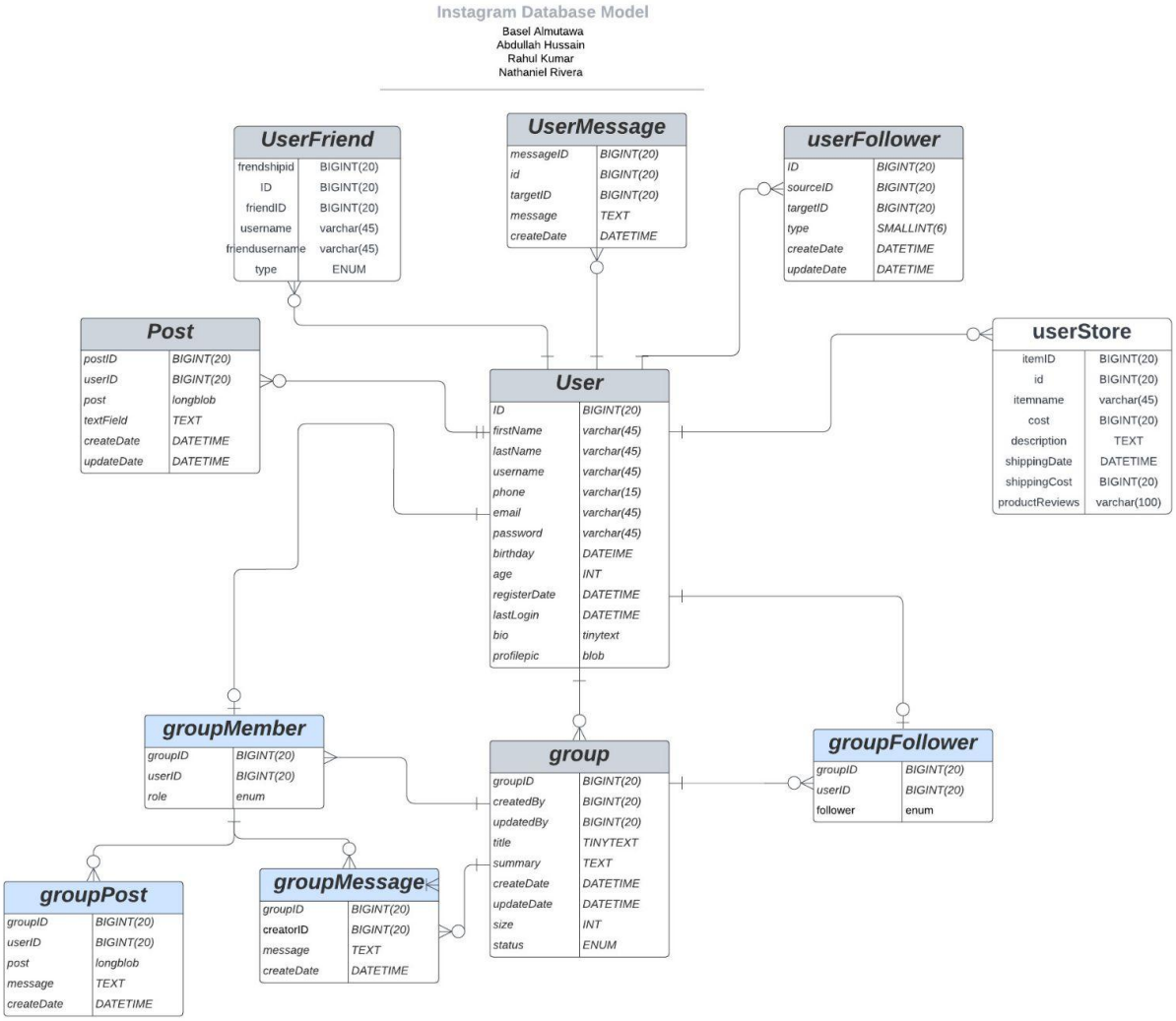
Updated E&R Diagram:



Report:

The E&R diagram was designed to satisfy the needs of users and groups in a social media platform. Also, it needs to follow the notion of reliability and security hence crow foot notation was used to determine the relation between the tables and data. For example, the user table was designed to have one and one /one only to avoid having null users because the system can't have that from a database management perspective . Also, zero or many was implemented to most tables to allow the user the choice of having a post or not and it was also implemented in groups in a way that a user can be a part of a group or not. Nevertheless, the connection between the groups is all relying on the user ID and the group ID as primary key based on which is being used in a certain table hence, they were set to not null because having null values could cause the database to be messy and dirty. Thus, most variables in the E&R diagram are set to not null. Also, user ID, group ID and other ID's values in the E&R diagram use BIGINT data type to avoid the issue of data might be exceeding the value of int data type and to make the database more organized and convenient to track data. Also, most other data like email, phone etc., were set to varchar data type since most of them are String characters / values. Thus, the E&R diagram is the foundation of the application and based on it the type of data and amount could be determined so it was important to have the E&R diagram set to cover most situations where data, or the database might be compromised and based on that the E&R diagram of Dawgstagram finalized. User tables were created by Rahul and Nathaniel while the group tables were created by Abdul and Basel.

User tables were created and linked according to the ER model and were made before group tables to ensure compatibility with data generation functions. Functions were created for different data types and were designed to be used across multiple tables within the python program. The user tables were updated as new rows were added, and generator functions were added with new data types. All user tables use their own ID primary keys, along with foreign keys referencing the central User table.

Group tables were also based on the E & R diagram that was created. Also, the work was splitted for two students to work on. The main table for this part is the Group table and then all the other tables can relate to it. Each group table has its own primary key, and foreign keys which are linked to the main group table. Each table is connected through the group ID and user ID which makes the table more organized and accessible to track data source .Moreover, the data was created randomly (which will be discussed in the next section). As for the user tables, group tables were created by using python and MySQL queries and using similar tactics to methods of generating user data.

When going about data creation for this project, we tried to be as real and transparent as possible in terms of user data. For the user and group_table tables, there was a lot of core data that other tables would require. For example, the id from user, the username, and group id are all examples of columns that were often found as foreign keys in other tables. For the actual data generation, we used a combination of generation libraries in Python as well as creating our own functions for generating data. The two libraries we used to generate data were ***names*** and ***essential-generators***. *Names* is a Python library that generates random realistic names using functions such as get_first_name() and get_last_name(). This library was primarily used in generating the names of users, their username, and their email. *Essential-generators* is a library for generating sentences (even though they are not coherent). Instead of inputting random

characters that don’t make words for the text columns, we may as well use a library that can generate words. Columns such as the user bio, user messages, and the message for a user post utilized the sentence() command from the DocumentGenerator provided by the library.
Finally, there were plenty of methods that we created ourselves to fulfill some table requirements. Creating a legitimate date, converting a profile picture to upload to MySQL, and retrieving user information for foreign keys were amongst the functions created. For the date() function, we made sure that the update info date was later than the creation date of a message or post because it wouldn't make sense to have it vice versa. It was necessary to include functions to retrieve user data because within the loop, the values of each variable are always changing so we had to call on this function to retrieve a random user already inputted into our MySQL database. Additionally, we were able to use the username to retrieve information as each username is unique like the user id. When executing these functions as parameters for the tables, we first executed all the user tables code before executing the group tables code. This was because we wanted the group tables to pull randomly from all 100 rows in the user data instead of out of a limited pool with every loop iteration. Finally, we added images to make this social media database look very similar to Instagram’s database model. We used images of dogs and cats from Kaggle to randomly select images from a folder to input. By transforming the images into binary sequences, we were able to input images into MySQL Workbench for each user (right-click on BLOB in the profilepic section of the user → Open Value In Editor → Select Image). All in all, our group made the data in these tables look as realistic as possible to simulate what a real-world database would look like.

There were a couple of validations in this project. The first time was to validate the E-R model that was created, and if it was the best method to start creating the tables. After testing and checking, some of the relations were changed. Also, crow’s feet notation were changed a couple of times in order to check the relations. The second validation was done after creating python code. Where to check the contents of the table, the keys, the rows, and uniqueness of each data. Some of the validations for the python code were done when creating complex queries and if it works or not. Complex queries will be discussed in the next section.

Four different queries were used to test the database. The first was a select query, bringing up the central group table and central user table with aliases and using a where statement to identify group creators by their user IDs. The second complex query brought up specified rows from the user table, then created a left join from the user store table, causing users to be displayed along with items they listed on the store. The third query was similar to the previous one, only using an inner join with the user post table to show users along with their posts. The fourth and final query selected specific rows from the group table and grouped them by size, specifying values greater than 90 and ordering them in descending order.

Group Participation Percentage:

Steps	Basel Almutawa	Abdullah Hussain	Rahul Kumar	Nathaniel Rivera	Total
E&R Diagram	25%	25%	25%	25%	100%
User Table Creation	0%	0%	50%	50%	100%
GroupTable Creation	50%	50%	0%	0%	100%
Data Creation	25%	25%	25%	25%	100%
Validation	25%	25%	25%	25%	100%
Complex Queries	25%	25%	25%	25%	100%