Quackstagram Database report

Group 40

May 2025

Contents

1 Project Management

Student Names	Student ID	Part
Louis Nathan Vessah Njoya Tchuente	I6371413	A(3,4), B(1,3), C, D(11-20), E
Abdul Moiz Akbar	I6375558	A(1,2), B(2), C, D(1-10), E

2 Introduction

The database includes data for users, posts, comments, notifications, follow, message, like_table and two logs tables(follow_log and postlogs) to record histories.

Users Table: This table holds user data such as username, password. See 4.1

Post Table: This table stores images uploaded by users with details like captions, and timestamps. See 4.2

Comments Table: This stores comments made by users under different posts. See 4.5

Like_table: This tables keeps track of users interactions with a post by storing the likes made by different users and the corresponding post reference. See 4.7

Follow Table: This tables defines follower relationships. See 4.6

Message Table: Manages private communications between users. See 4.3

Notification Table: Alerts user to activities under their posts etc. See 4.4

Log Tables: The logs table (follow_log and postlogs) record histories used for analysis purposes. See 4.8 and 4.9

Most tables contain an id as a primary key, except for the follow table and like_table which use composite primary keys such as (follower_id and followed_id for the follow table). This was done so that the validation of the relationship between two entities is enforced by the table in order to minimize human error. Note that all id's are on Auto Increment mode.

The communication between the application and database is done by using SQL queries through a connection managed in the DatabaseConnection class, allowing it to perform operations such as saving posts, and loading user data.

For instance, when a user uploads an image, the IamgeUploadUI class inserts data into the post table, while the commentsUI queries the comment table to retrieve existing comments and insert new ones linked by post_id. Returned queries are read and integrated in to the GUI dynamically, such as populating the JTextArea component inside the DirectMessagingUI ensuring that the GUI refelects the databse data.

3 Entity-Relationship Diagram

This diagram contains the main tables of the database. The two logs tables were added for next parts(triggers).

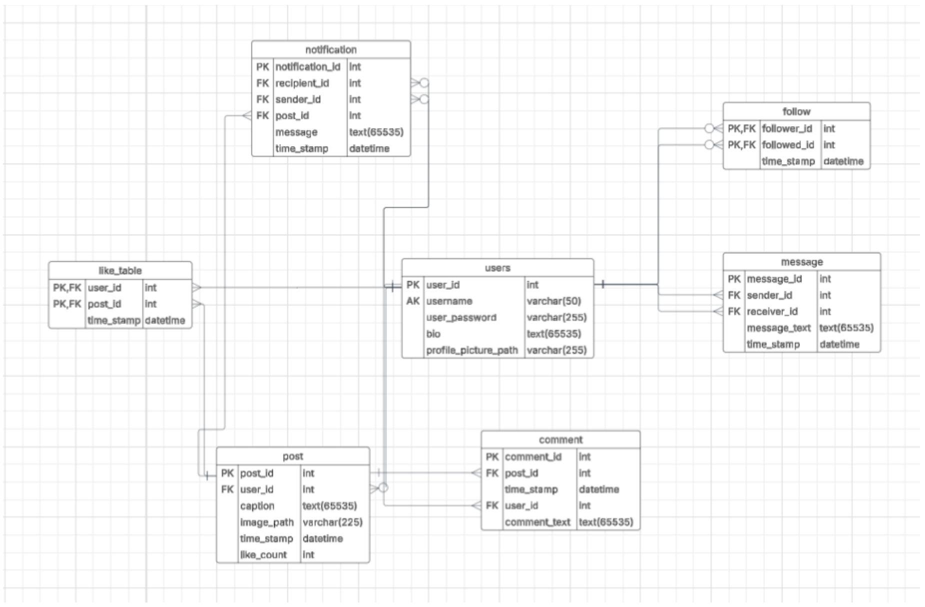

Figure 1: ERD Diagram

4 Tables Views

The following tables contain sample data which was later modified to suit our codebase from last course.

4.1 Users Table

user_id	username	user_password	bio	profile_picture_path
1	Louis	rclla	I love coding!	/profiles/louis.jpg
2	Abdul	abdul1	Music enthusiast.	/profiles/abdul.jpg
3	Nathan	Password	Gamer and streamer.	/profiles/nathan.jpg
4	Freeze	rclla	Chillin like a villain.	/profiles/freeze.jpg

post_id	user_id	caption	image_path	time_stamp	like_count
1	1	Beautiful sunset!	/posts/sunset.jpg	2025-04-29 12:00:00	3
2	2	New song drop!	/posts/song.jpg	2025-04-29 13:00:00	2
3	5	City lights at night.	/posts/city.jpg	2025-04-29 14:00:00	1
4	7	My latest fantasy art.	/posts/art.jpg	2025-04-29 15:00:00	4

4.2 Post table

4.3 Message Table

message_id	sender_id	receiver_id	message_text	time_stamp
1	1	2	Hey, loved your new song!	2025-04-29 13:00:00
2	2	1	Thanks, glad you liked it!	2025-04-29 13:15:00
3	5	1	Your sunset pic is stunning!	2025-04-29 12:30:00
4	7	8	Hey, let's grab some food so	2025-04-29 16:00:00

4.4 Notification Table

notification_id	recipient_id	sender_id	post_id	message	time_stamp
1	1	2	1	Abdul liked your post.	2025-04-29 12:02:00
2	1	3	1	Nathan commented on your post.	2025-04-29 12:10:00
3	2	1	2	Louis Liked your post.	2025-04-29 13:02:00
4	7	6	4	Lorin commented on your post.	2025-04-29 15:05:00

4.5 Comment Table

comment_id	post_id	time_stamp	user_id	comment_text
1	1	2025-04-29 12:05:00	2	Amazing view.
2	1	2025-04-29 12:10:00	3	Love the colors!
3	2	2025-04-29 13:05:00	1	Great track!
4	4	2025-04-29 15:05:00	6	Stunning artwork!

4.6 Follow Table

follower_id	followed_id	time_stamp
1	2	2025-04-29 10:00:00
6	7	2025-04-29 10:25:00
8	1	2025-05-05 13:05:00
8	5	2025-05-05 13:05:00

4.7 Like_table Table

follower_id	followed_id	time_stamp
1	2	2025-04-29 10:00:00
6	7	2025-04-29 10:25:00
8	1	2025-05-05 13:05:00
8	5	2025-05-05 13:05:00

4.8 postlogs Table

log_id	message
1	Nathan has 1 post
2	Mystra has 2 post

4.9 follow_log

log_id	follow message
1	Yurpi has followed Lorin
2	Yurpi has followed Lyor

5 Functional Dependencies

Users :

$$user\_id \rightarrow (username, user\_password, bio, profile\_picture\_path)$$

post :

$$post\_id \rightarrow (caption, image\_path, time\_stamp, like\_count)$$

Message :

$$message\_id \rightarrow (sender\_id, receiver\_id, message\_text, time\_stamp)$$

Notification :

$$notification\_id \rightarrow (recipient\_id, sender\_id, post\_id, message, time\_stamp)$$

6 Normalization

Users :

$user\_id \rightarrow username$

$user\_id \rightarrow user\_password$

$user\_id \rightarrow bio$

$user\_id \rightarrow profile\_picture\_path$

Every dependency has a super key on the left hand side, hence the Users table is in 3NF.

Post :

$post\_id \rightarrow caption$

$post\_id \rightarrow image\_path$

$post\_id \rightarrow time\_stamp$

$post\_id \rightarrow like\_count$

Every dependency has a super key on the left hand side, hence the Post table is in 3NF.

Message :

$message\_id \rightarrow sender\_id$

$message\_id \rightarrow receiver\_id$

$message\_id \rightarrow message\_text$

$message\_id \rightarrow time\_stamp$

Every dependency has a super key on the left hand side, hence the Message table is in 3NF.

Notification :

$notification\_id \rightarrow recipient\_id$

$notification\_id \rightarrow sender\_id$

$notification\_id \rightarrow post\_id$

$notification\_id \rightarrow message$

$notification\_id \rightarrow time\_stamp$

Every dependency has a super key on the left hand side, hence the Notification table is in 3NF.

7 Views

Usefulness of Proposed Views

The three proposed views offer critical insights for enhancing user engagement, optimizing content delivery, and ensuring platform scalability:

- **most_liked_posts** helps identify trending content, enabling features like ”Popular Posts” or Explore Feed.
- **top_active_users** supports recognition of highly engaged users, which can power features such as badges, user rankings, or recommendations.
- **avg_likes_per_post** provides a high-level engagement metric useful for admin dashboards or health checks.

Each view addresses a different category: content popularity, user behavior, and system analytics.

8 Indexes

8.1 Query Optimization and Index Justification

To improve performance of the views, two indexes were introduced on the `like_table`:

- **idx_like_post_id:** Optimizes aggregations by post in `most_liked_posts`.
- **idx_like_user_id:** Improves filtering and grouping in `top_active_users`.

8.2 Performance Testing Results

Query execution times were measured using DBeaver:

- `SELECT post_id, COUNT(*) FROM like_table GROUP BY post_id HAVING COUNT(*) > 2`
  - Without index: 0.004s
  - With index: 0.002s

  - **Result:** 50% improvement in execution time

These results show how indexing improves scalability and responsiveness of view-based queries.

9 Procedures, Functions and Triggers

9.1 Procedures

A procedure implemented is registration of a new user (register_new_user). This is use to add a new user with its corresponding information into the users table.

Another procedure implemented is Following of a user (followUser). This is used to insert data into the follow table.

Saving of comments (save_comment) and saving of posts (save_post) are also implemented to insert data into the comment and post table respectively.

With these procedures, basic operations are made easier instead of having to create duplicate code every time an operation is to be performed.

9.2 Functions

Basic functions are implemented; get_Username to get the username given a user_id, count_users_posts to count the total number of posts made by a user so that we have a metric of the user's activity, and getUser_id to the get the user_id given a username.

These functions ease work when using triggers to get required data and also in the actual database implmentation with quackstagram.

9.3 Triggers

Two triggers are implemented to return log messages based on a user's activity.

One is used to update the follow_log table where a record is created stating the recent following made by a user (can be called a notification) although this is not recorded into the notification table(this table only contains post related notifications).

The other is used to update the total number of posts made by a user and the record is stored into the postlogs table.

These triggers help to get the activity made by a user and their contributions.

10 SQL Queries

10.1 List all users who have more than X followers where X can be any integer value.

Query and Answer :

```
SELECT u.username, COUNT(f.followed_id) AS follower_count

FROM users u

JOIN follow f ON u.user_id = f.followed_id

GROUP BY u.username

HAVING COUNT(f.followed_id) > 1
```

username	follower_count
Louis	2
Lyor	2

10.2 Show the total number of posts made by each user. (You will have to decide how this is done, via a username or user_id)

Query and Answer :

```
SELECT u.username, COUNT(p.post_id) AS total_posts

FROM users u

LEFT JOIN post p ON u.user_id = p.user_id

GROUP BY u.username
```

username	total_posts
abdul	1
Lorin	0
Louis	1
Lyor	1
Mystar	2
Xylo	1
Yurpi	0
Zara	0

10.3 Find all comments made on a particular user's post.

Query and Answer :

```
SELECT c.comment_text, c.time_stamp, u.username AS commenter

FROM comment c

JOIN post p ON c.post_id = p.post_id

JOIN users target ON p.user_id = target.user_id

JOIN users u ON c.user_id = u.user_id

WHERE target.username = 'Louis'
```

comment_text	time_stamp	commenter
Amazing view!	2025-04-29 12:05:00	abdul
Love the colors!	2025-04-29 12:10:00	Xylo

10.4 Display the top X most liked posts.

Query and Answer :

```
SELECT p.post_id, u.username, COUNT(l.user_id) AS like_count

FROM post p

JOIN users u ON p.user_id = u.user_id

LEFT JOIN like_table l ON p.post_id = l.post_id

GROUP BY p.post_id, u.username

ORDER BY like_count DESC

LIMIT 3
```

post_id	username	like_count
4	Mystar	4
1	Louis	3
2	abdul	2

10.5 Count the number of posts each user has liked.

Query and Answer :

```
SELECT u.username, COUNT(l.post_id) AS likes_given

FROM users u

LEFT JOIN like_table l ON u.user_id = l.user_id

GROUP BY u.username
```

username	likes_given
abdul	2
Lorin	1
Louis	3
Lyor	0
Mystar	0
Xylo	2
Yurpi	1
Zara	1

10.6 List all users who haven't made a post yet.

Query and Answer :

```
SELECT u.username

FROM users u

LEFT JOIN post p ON u.user_id = p.user_id

WHERE p.post_id IS NULL
```

username
Lorin
Yurpi
Zara

10.7 List users who follow each other.

Query and answer :

```
SELECT u1.username AS user1, u2.username AS user2
```

FROM follow f1

JOIN follow f2 ON f1.follower_id = f2.followed_id AND f1.followed_id = f2.follower_id

JOIN users u1 ON f1.follower_id = u1.user_id

JOIN users u2 ON f1.followed_id = u2.user_id

WHERE u1.user_id ¡ u2.user_id – prevents duplicate pairs

Answer: based on our data the query returns empty columns

10.8 Show the user with the highest number of posts.

Query and answer :

SELECT u.username, COUNT(p.post_id) AS post_count

FROM users u

JOIN post p ON u.user_id = p.user_id

GROUP BY u.username

ORDER BY post_count DESC

LIMIT 1 – else will return every user who posted at least once

username	post_count
Mystar	2

10.9 List the top X users with the most followers.

Query and answer :

SELECT u.username, COUNT(f.follower_id) AS follower_count

FROM users u

LEFT JOIN follow f ON u.user_id = f.followed_id

GROUP BY u.username

ORDER BY follower_count DESC

LIMIT 3

username	follower_count
Louis	2
Lyor	2
abdul	1

10.10 Find posts that have been liked by all users.

Query and Answer :

SELECT post_id FROM post

WHERE like_count = (SELECT COUNT(user_id) FROM users)

post_id
4
5

10.11 Display the most active user (based on likes).

Query and answer :

SELECT p.user_id, MAX(like_count) as max_likes from post p

left join top_active_users tau on get_Username(p.post_id) = tau.username

group by p.user_id

user_id	maxLikes
1	3
2	2
3	8
5	1
7	100

10.12 Find the average number of likes per post for each user.

Query and Answer :

```
SELECT user_id, AVG(like_count) AS avg_likes FROM post

GROUP By user_id
```

user_id	avg_likes
1	3
2	2
3	8
5	1
7	54

10.13 Show posts that have more comments than likes.

Query :

```
SELECT p.post_id from post p

left join comment c on p.post_id = c.post_id

group by p.post_id, p.like_count

having count(c.comment_id) > p.like_count
```

Answer: based on our data this query returns empty columns

10.14 List the users who have liked every post of a specific user.

Query and answer :

```
SELECT DISTINCT lt.user_id

FROM like_table lt

WHERE NOT EXISTS (

SELECT p.post_id

FROM post p

WHERE p.user_id = X

AND NOT EXISTS (

SELECT 1
FROM like_table lt2

WHERE lt2.post_id = p.post_id

AND lt2.user_id = lt.user_id))
```

Taking X as 1:

user_id
2
3
4

10.15 Display the most popular post of each user (based on likes).

Query and answer :

```
SELECT p1.user_id, p1.post_id, p1.like_count

FROM post p1

WHERE p1.post_id = (

SELECT p2.post_id

FROM post p2

WHERE p2.user_id = p1.user_id

ORDER BY p2.like_count desc

LIMIT 1)
```

user_id	post_id	like_count
1	1	3
2	2	2
5	3	1
3	5	8
7	6	100

10.16 Find the user(s) with the highest ratio of followers to following.

Query and answer :

```
SELECT u.username,

COUNT(DISTINCT f1.follower_id) AS followers, COUNT(DISTINCT f2.followed_id)
AS following,

COUNT(DISTINCT f1.follower_id) / NULLIF(COUNT(DISTINCT f2.followed_id),
0) AS ratio FROM users u

LEFT JOIN follow f1 ON u.user_id = f1.followe_id

LEFT JOIN follow f2 ON u.user_id = f2.follower_id

GROUP BY u.user_id
ORDER BY ratio DESC
```

These are the top 2 users

username	follower	following	ratio
Louis	2	1	2
Lyor	2	1	2

10.17 Show the month with the highest number of posts made.

Query and answer :

```
SELECT DATE_FORMAT(time_stamp, '%Y-%m') AS post_month, COUNT(*)
AS total_posts

FROM post

GROUP BY post_month

ORDER BY total_posts DESC

Limit 1
```

post_month	post_count
2025-04-	4

10.18 Identify users who have not interacted with a specific user's posts.

Query and answer :

SELECT u.user_id, u.username FROM users u

WHERE u.user_id NOT IN (

SELECT l.user_id

FROM like_table l

JOIN post p ON l.pos_id = p.post_id

WHERE p.user_id = X

UNION

SELECT c.user_id

FROM comment c

JOIN post p ON c.post_id = p.post_id

WHERE p.user_id = X)

order by u.user_id ASC

Taking X as 1;

user_id	username
1	Louis
5	Lyor
6	Lorin
7	Mystar
8	Yurpi

10.19 Display the user with the greatest increase in followers in the last X days.

Query and answer :

SELECT followed_id AS user_id, COUNT(follower_id) AS follower_increase

FROM follow

WHERE time_stamp ¿= DATE_SUB('2025-05-08 17:01:00', INTERVAL X DAY)

GROUP BY followed_id

ORDER BY follower_increase DESC

LIMIT 1

Taking X as 10

user_id	follower_increase
1	2

10.20 Find users who are followed by more than X% of the platform users.

Query and answer :

SELECT u.username, COUNT(f.follower_id) AS follower_count

FROM users u

JOIN follow f ON u.user_id = f.followed_id

GROUP BY u.user_id, u.username

HAVING COUNT(f.follower_id) ¿ (SELECT COUNT(*) * 0.1 FROM users)

Username	Follower Count
Louis	2
abdu	1
Xylo	1
Zara	1
Lyor	2
LorIn	1
MyStar	1
topi	1