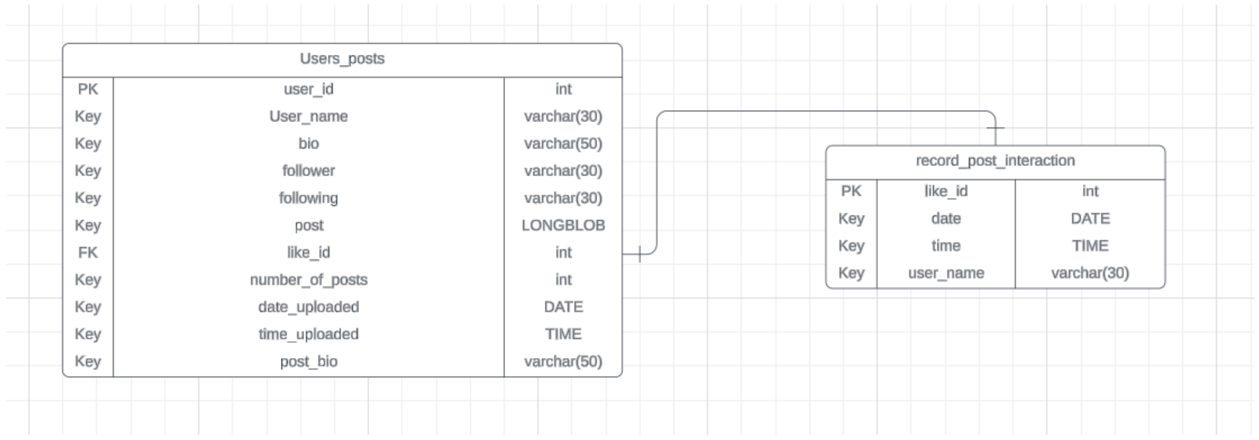
Database Normalization:

(The CREATE table statements have not been made by hand but via the export function in lucidchart, link to final CREATE statements can be found in appendix on lucid app -> export as SQL)

1. First Normal Form (1NF), data breakdown

1NF requires atomic data



Users_posts		
PK	user_id	int
Key	User_name	varchar(30)
Key	bio	varchar(50)
Key	follower	varchar(30)
Key	following	varchar(30)
Key	post	LONGBLOB
FK	like_id	int
Key	number_of_posts	int
Key	date_uploaded	DATE
Key	time_uploaded	TIME
Key	post_bio	varchar(50)

record_post_interaction		
PK	like_id	int
Key	date	DATE
Key	time	TIME
Key	user_name	varchar(30)

The first Normal form of the table only considers the *users_posts* and a table *record_post_interaction*. This will not cause repetition of sets of data as each row will represent a unique set of data, however repeated values on post will happen on each query due to the 1:1 relation. Each follower will also cause the table to repeat which is not optimal as when a person has n followers, there will be an n amount of repetition on the dataset because of many partial dependencies. The following tables can fully handle the applications needs however check the sample table below for the inefficiencies it may cause

A breakdown of all the repetition 1NF may cause for each row:

Users_posts

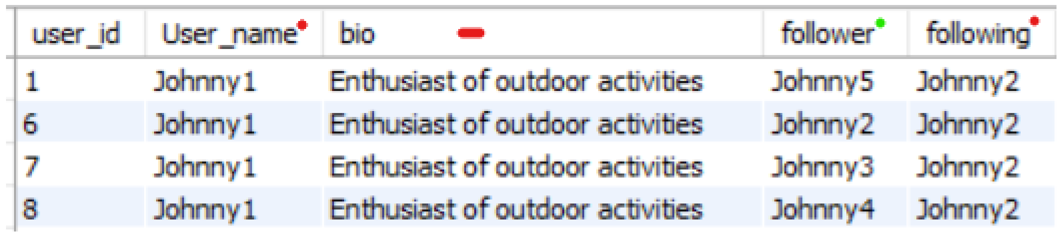
- User_id *:* Unique AA PK identifier for the post - > user interactions (e.g. not an actual identifier for users)
- User_name : Identifies the users' name

- Bio: the bio is a varchar string with the users' bio
- Follower: The followers list a unique followers' user name in the table (see below). This will thus cause high amounts of repetition in the database with the above and following columns
- Following: Same logic as for Followers, it will only list a unique user which will cause repetition
- Post : LONGBLOB datatype under the assumption that files are not compressed
- <u>Like_id</u> : FK to record_post_interaction table, follows the follower following logic, for each like the user receives the set of entries above will repeat as per the 1:1 relationship with the record_post_interaction.
- date uploaded : Date datatype which records the upload date of the post
- Post_bio : varchar data type that records the posts bio

Record_post_interaction

- Like_id : PK records when a "like" happens on the post, unique AA.
- DATE
- TIME : both describe the date and time of the like interaction.
- User_name : records the users' name that liked the post

Sample table of 1NF:



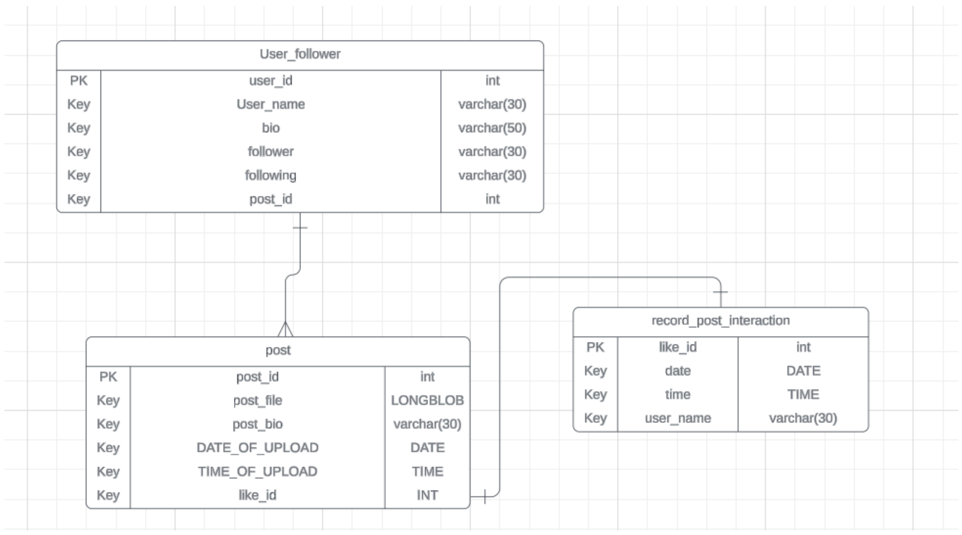
user_id	User_name	bio	follower	following
1	Johnny1	Enthusiast of outdoor activities	Johnny5	Johnny2
6	Johnny1	Enthusiast of outdoor activities	Johnny2	Johnny2
7	Johnny1	Enthusiast of outdoor activities	Johnny3	Johnny2
8	Johnny1	Enthusiast of outdoor activities	Johnny4	Johnny2

The table shows the repetition involved in 1NF, the user_id only describes the interaction of following followers, and so entries such as bio and user_name will be repeated at least n*m times, where n is the number of followers Johnny1 has and m is the number of people Johnny follows, however each row represents a UNIQUE COLLECTION of attributes.

2. Second Normal Form (2NF)

2NF requires the table to be in 1NF and all the attributes to be non-partially dependent on the primary key.

The second normal form includes removing transitive dependencies. These are dependencies that are subsets of other entries which are dependent on the primary key. From the 1NF we would first have to break down the posts and users, as the date_uploaded, time_uploaded and bio_uploaded all have a dependency on the post itself.



User_follower		
PK	user_id	int
Key	User_name	varchar(30)
Key	bio	varchar(50)
Key	follower	varchar(30)
Key	following	varchar(30)
Key	post_id	int

post		
PK	post_id	int
Key	post_file	LONGBLOB
Key	post_bio	varchar(30)
Key	DATE_OF_UPLOAD	DATE
Key	TIME_OF_UPLOAD	TIME
Key	like_id	INT

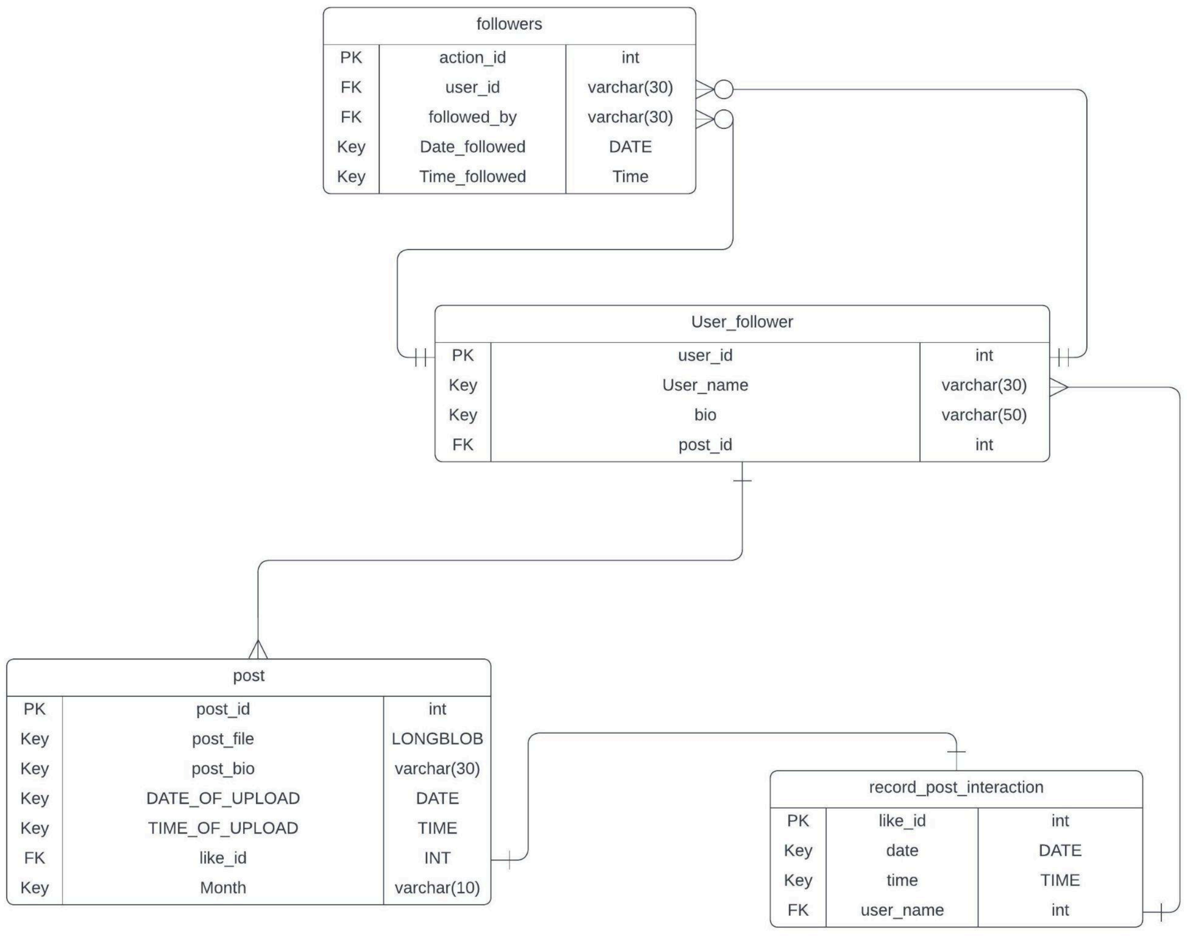
record_post_interaction		
PK	like_id	int
Key	date	DATE
Key	time	TIME
Key	user_name	varchar(30)

Now each key `user_id` will not need to be created for each interaction on the post

2.1 Second Normal Form (2NF)

Intermediate design changes to prevent data redundancy

Intermediate changes such as better handling of following events have been added since 2NF allows for better handling of information. New `followers` table describes the action of following by action_id with the following `Date_followed` and `Time_Followed` such that more information is recorded for data collection of CSS. The `Month` column has been added to the `post` table as it is a crucial description for one of the queries which CSS needs.



followers		
PK	action_id	int
FK	user_id	varchar(30)
FK	followed_by	varchar(30)
Key	Date_followed	DATE
Key	Time_followed	Time

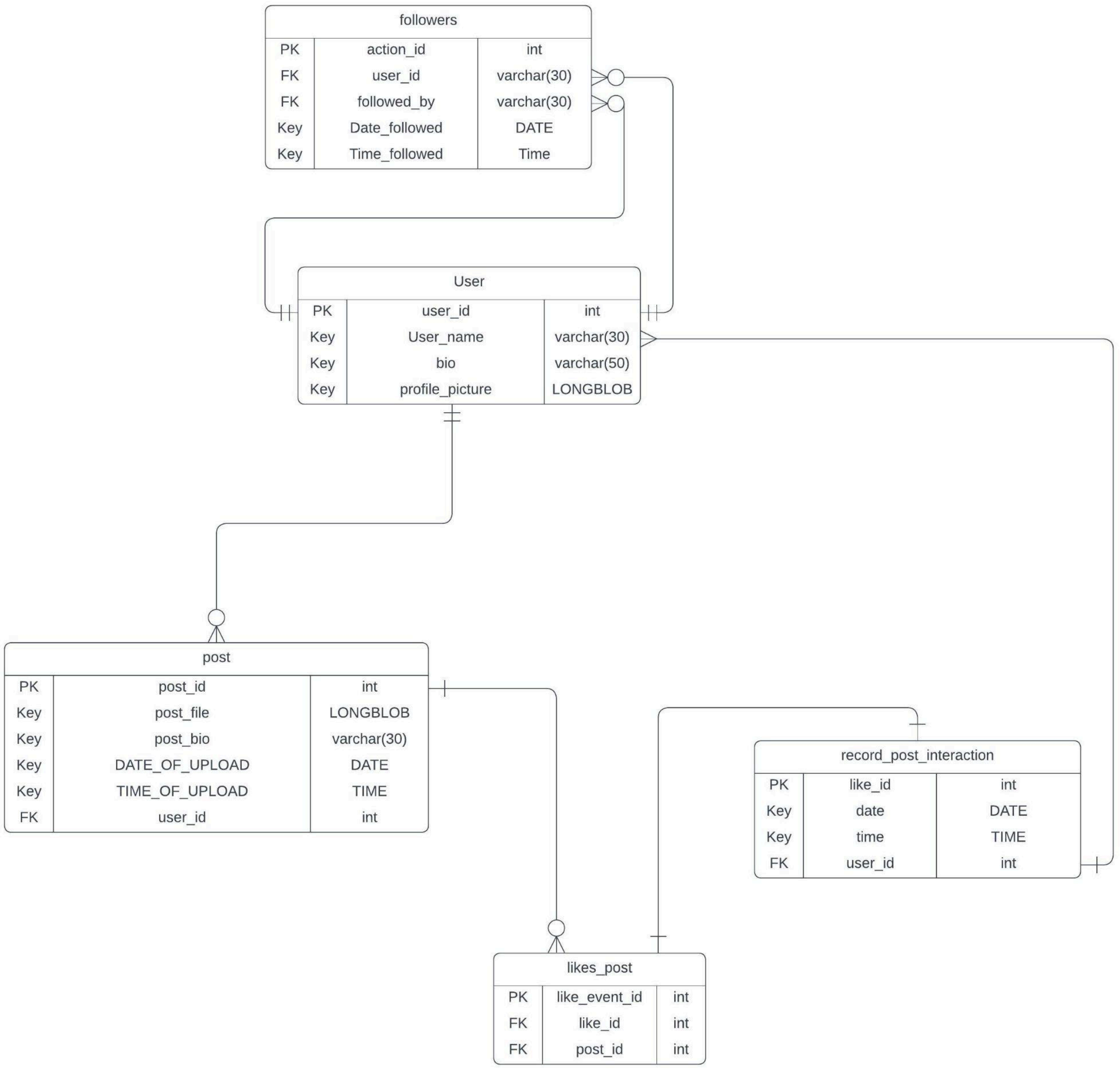
User_follower		
PK	user_id	int
Key	User_name	varchar(30)
Key	bio	varchar(50)
FK	post_id	int

post		
PK	post_id	int
Key	post_file	LONGBLOB
Key	post_bio	varchar(30)
Key	DATE_OF_UPLOAD	DATE
Key	TIME_OF_UPLOAD	TIME
FK	like_id	INT
Key	Month	varchar(10)

record_post_interaction		
PK	like_id	int
Key	date	DATE
Key	time	TIME
FK	user_name	int

3. Third Normal Form (3NF)

3NF requires the table to be in 2NF and all the attributes to be non-transitively dependent on the primary key.



followers		
PK	action_id	int
FK	user_id	varchar(30)
FK	followed_by	varchar(30)
Key	Date_followed	DATE
Key	Time_followed	Time

User		
PK	user_id	int
Key	User_name	varchar(30)
Key	bio	varchar(50)
Key	profile_picture	LONGBLOB

post		
PK	post_id	int
Key	post_file	LONGBLOB
Key	post_bio	varchar(30)
Key	DATE_OF_UPLOAD	DATE
Key	TIME_OF_UPLOAD	TIME
FK	user_id	int

record_post_interaction		
PK	like_id	int
Key	date	DATE
Key	time	TIME
FK	user_id	int

likes_post		
PK	like_event_id	int
FK	like_id	int
FK	post_id	int

4. Relationships between Entities

1. **User and Followers**: One-to-Many (One user can follow many users, and many users can follow one user).
2. **User and Post**: One-to-Many (One user can have many posts).
3. **Post and Likes Post**: Many-to-Many (Many posts can have many likes, and many users can like many posts).
4. **User and Record Post Interaction**: One-to-Many (One user can have many interactions recorded).

4.1 Final Attributes and Their Definitions

1. **`User` Table**
   - **`<u>user_id</u>`**: Integer, super key.
   - **`user_name`**: String, the name of the user.
   - **`bio`**: String, biography of the user.
   - **`profile_picture`**: BLOB, the profile picture of the user.
2. **Followers Table**
   - **`<u>action_id</u>`** : Integer, primary key.
   - **`user_id**: Integer, foreign key referencing `User`.
   - **`followed_by`**: String, foreign key referencing `User (has been changed to user_ids)`.
   - **`Date_followed`**: Date, the date when the user was followed.
   - **`Time_followed`**: Time, the time when the user was followed.
3. **Post Table**
   - **<u>post_id</u>**: Integer, primary key.
   - **post_file**: BLOB, the file of the post.
   - **post_bio**: String, the bio or description of the post.

  - **DATE_OF_UPLOAD**: Date, the date when the post was uploaded.
  - **TIME_OF_UPLOAD**: Time, the time when the post was uploaded.
  - **user_id**: Integer, foreign key referencing User.
4. **Likes Post Table**
  - **like_event_id**: Integer, primary key.
  - like_id: Integer, foreign key referencing User.
  - post_id: Integer, foreign key referencing Post.
5. **Record Post Interaction Table**
  - like_id: Integer, primary key.
  - **date**: Date, the date of the like.
  - time: Time, the time of the like.
  - user_id: Integer, foreign key referencing User.

Appendix:

Note: Slight ramifications have been added to the database as the integration process started, however the skeleton and the initial design of the database persists in the report version.

ERD FIRST VERSION OF 3NF
OR MYSQL EXPORT:

```
use quackstagram;
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERRO
R_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema quackstagram
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema quackstagram
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `quackstagram` DEFAULT CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci ;
USE `quackstagram` ;

-- -----------------------------------------------------
-- Table `quackstagram`.`user`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `quackstagram`.`user` (
  `user_id` INT NOT NULL AUTO_INCREMENT,
  `User_name` VARCHAR(30) NULL DEFAULT NULL,
  `bio` VARCHAR(50) NULL DEFAULT NULL,
  `profile_picture` LONGBLOB NULL DEFAULT NULL,
  PRIMARY KEY (`user_id`),
  INDEX `Key` (`User_name` ASC, `bio` ASC) VISIBLE,
  INDEX `profile_picture_index` (`profile_picture`(255) ASC) VISIBLE)
ENGINE = InnoDB
AUTO_INCREMENT = 17
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `quackstagram`.`followers`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `quackstagram`.`followers` (
  `action_id` INT NOT NULL AUTO_INCREMENT,
  `user_id` INT NULL DEFAULT NULL,
  `followed_by` INT NULL DEFAULT NULL,
  `Date_followed` DATE NULL DEFAULT NULL,
  `Time_followed` TIME NULL DEFAULT NULL,
  PRIMARY KEY (`action_id`),
  INDEX `user_id` (`user_id` ASC) VISIBLE,
  INDEX `followed_by` (`followed_by` ASC) VISIBLE,
  INDEX `Key` (`Date_followed` ASC, `Time_followed` ASC) VISIBLE,
  CONSTRAINT `followers_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `quackstagram`.`user` (`user_id`)
    ON DELETE CASCADE,
  CONSTRAINT `followers_ibfk_2`
    FOREIGN KEY (`followed_by`)
    REFERENCES `quackstagram`.`user` (`user_id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `quackstagram`.`record_post_interaction`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `quackstagram`.`record_post_interaction` (
  `like_id` INT NOT NULL AUTO_INCREMENT,
  `date` DATE NULL DEFAULT NULL,
  `time` TIME NULL DEFAULT NULL,
  `user_id` INT NULL DEFAULT NULL,
  PRIMARY KEY (`like_id`),
  INDEX `user_id` (`user_id` ASC) VISIBLE,
  INDEX `Key` (`date` ASC, `time` ASC) VISIBLE,
  CONSTRAINT `record_post_interaction_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `quackstagram`.`user` (`user_id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `quackstagram`.`post`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `quackstagram`.`post` (
  `post_id` INT NOT NULL AUTO_INCREMENT,
  `post_file` LONGBLOB NULL DEFAULT NULL,
  `post_bio` VARCHAR(30) NULL DEFAULT NULL,
  `DATE_OF_UPLOAD` DATE NULL DEFAULT NULL,
  `TIME_OF_UPLOAD` TIME NULL DEFAULT NULL,
  `user_id` INT NULL DEFAULT NULL,
  PRIMARY KEY (`post_id`),
  INDEX `user_id` (`user_id` ASC) VISIBLE,
  INDEX `post_file_index` (`post_file`(255) ASC) VISIBLE,
  INDEX `Key` (`post_bio` ASC, `DATE_OF_UPLOAD` ASC, `TIME_OF_UPLOAD` ASC) VISIBLE,
  CONSTRAINT `post_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `quackstagram`.`user` (`user_id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE TABLE IF NOT EXISTS `quackstagram`.`comments` (
  `comment_id` INT NOT NULL AUTO_INCREMENT,
  `post_id` INT,
  `user_id` INT,
  `comment_time` TIME NULL DEFAULT NULL,
  `comment_content` VARCHAR(100) DEFAULT NULL,
  PRIMARY KEY (`comment_id`),
  INDEX `fk_post_id` (`post_id` ASC) VISIBLE,
  INDEX `fk_user_id` (`user_id` ASC) VISIBLE,
  CONSTRAINT `comment_ibfk_1`
    FOREIGN KEY (`user_id`)
    REFERENCES `quackstagram`.`user` (`user_id`)
    ON DELETE CASCADE,
  CONSTRAINT `comment_ibfk_2`
    FOREIGN KEY (`post_id`)
    REFERENCES `quackstagram`.`post` (`post_id`)
    ON DELETE CASCADE
) ENGINE = InnoDB DEFAULT CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci;



-- -----------------------------------------------------
-- Table `quackstagram`.`likes_post`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `quackstagram`.`likes_post` (
  `like_event_id` INT NOT NULL AUTO_INCREMENT,
  `like_id` INT NULL DEFAULT NULL,
  `post_id` INT NULL DEFAULT NULL,
  PRIMARY KEY (`like_event_id`),
  INDEX `like_id` (`like_id` ASC) VISIBLE,
  INDEX `post_id` (`post_id` ASC) VISIBLE,
  CONSTRAINT `likes_post_ibfk_1`
    FOREIGN KEY (`like_id`)
    REFERENCES `quackstagram`.`record_post_interaction` (`like_id`)
    ON DELETE CASCADE,
  CONSTRAINT `likes_post_ibfk_2`
    FOREIGN KEY (`post_id`)
    REFERENCES `quackstagram`.`post` (`post_id`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `quackstagram`.`login_data`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `quackstagram`.`login_data` (
  `username` VARCHAR(20) NULL DEFAULT NULL,
  `Hashed_password` VARCHAR(50) NULL DEFAULT NULL)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


);
```