DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES2201800684   SATHVIK SAYA

Summary of the Project:-

I have selected an instagram database which is a model/dummy database of instagram named as insta_db, we have tables like users, photos, likes, comments, follows, tags, photo_tags and unfollows.

Where this database can handle new users, photos they post, comments they comment to the photos, likes they give to them, one user following and unfollowing logs, tag-names they tag to photos.

This database is managed with triggers to avoid any illegal actions, which get triggered when the user creates, updates or deletes rows in tables.

I have also tried to add a few MySQL Queries to understand the database structure and how we can get the data we require.

It's more like a real world database, when this database is integrated with a nice and interactive front-end this can make out a nice application, if not like instagram, we can try out any other social media application.

INDEX

INTRODUCTION

Our database is dummy instagram database named as insta_db, with the following tables listed below:-

```
+-------------------------+
| Tables_in_insta_db |
+-------------------------+
| comments            |
| follows              |
| likes                  |
| photo_tags          |
| photos               |
| tags                   |
| users                 |
| unfollows           |
+-------------------------+
```

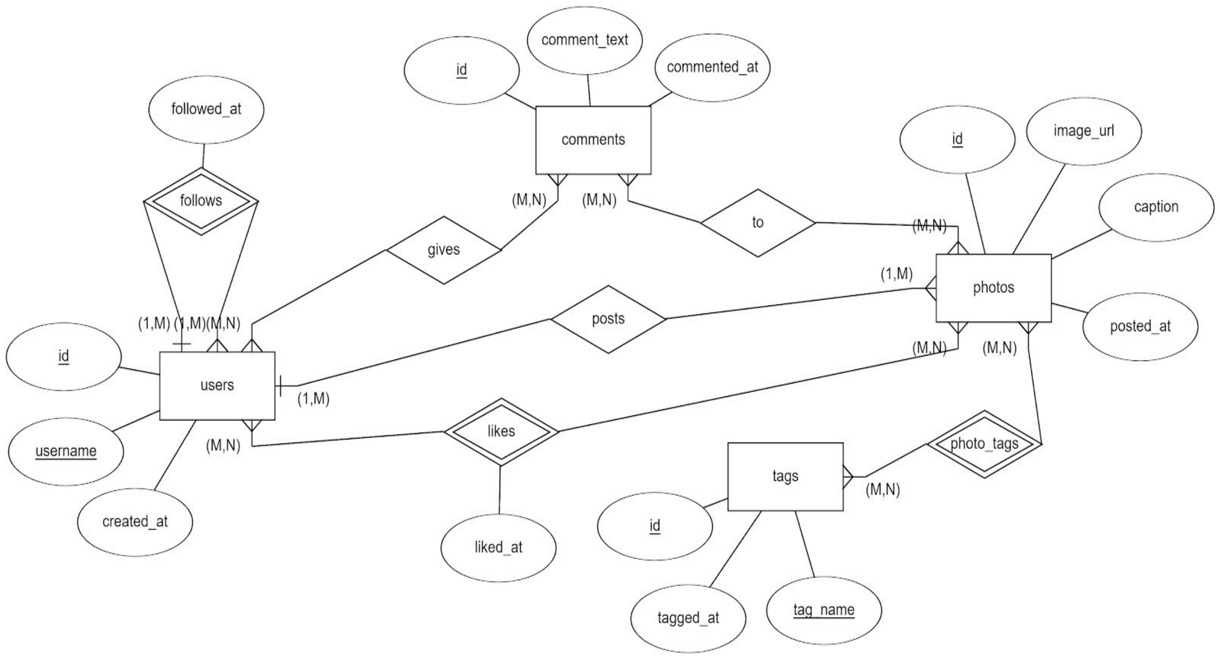
We shall now describe each table one by one,

1. users table:- We add an user into the database just with the username, then the user is inserted into the table with id which primary key and auto-incremented and username which is unique and timestamp created_at using now() add time at the time of addition of the new user.
2. photos table:- We have an id which is a primary key with auto-increment and image_url for the photo and with a caption for it, and also a timestamp posted_at with now() and has a foreign key user_id which references to users(id).
3. comments table:- We have an id which is a primary key with auto-increment and comment_text for the comment and timestamp commented_at with now(), and finally the two foreign keys user_id referencing users(id) and photo_id referencing photo(id).
4. likes table:- We have two foreign keys user_id referencing users(id) and photo_id referencing photo(id), which are together as primary keys so that we can avoid duplicate entries, so that one user cannot like the same photo multiple times and also with a timestamp liked_at with now() to record the time at which the user liked the photo.
5. follows table:- We have follower_id and following_id as two foreign key, both referencing user(id) which is a unary relationship with timestamp followed_at with now() to record the time at which the user followed the other user and same as in likes table, both these id's are together made as primary keys to avoid duplicates, i.e., one user following the other(one-same) user again and again.
6. tags table:- We have an id which is primary key with auto-increment, to record the type of tag user tags their photos as tag_name, and with a timestamp tagged_at with now() at which the user has tagged the photo.
7. photo_tags tabel:- We have two foreign keys tag_id referencing tags(id) and photo_id referencing photos(id) and together as primary keys to avoid the same photo, tagged with the same tag-name.

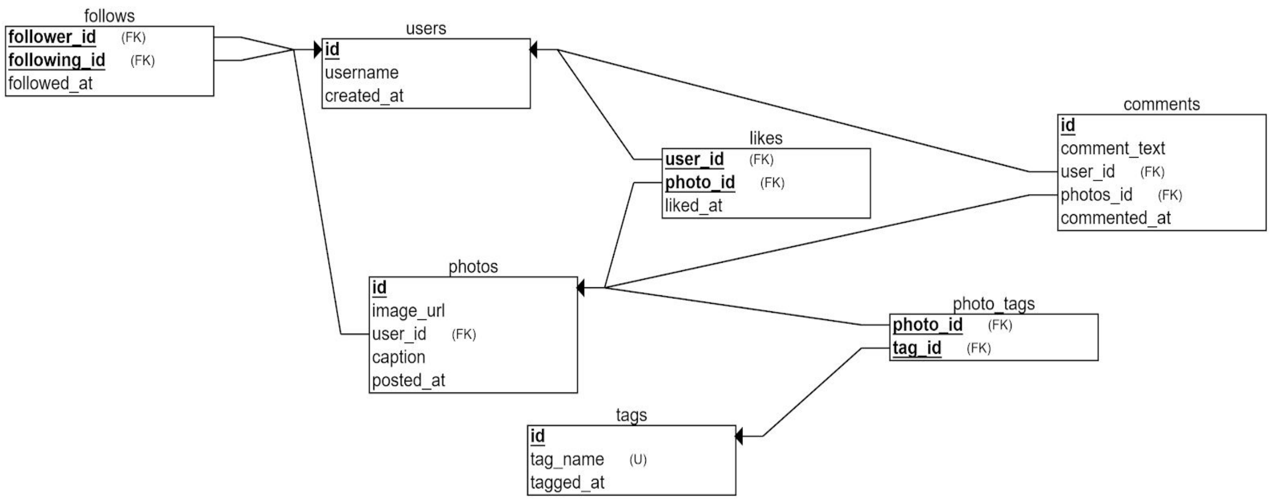
8. unfollows table:- We have the same description of follows table, where this unfollows table keeps record of users unfollowing, this is a new logger-trigger-table.

DATA MODEL

1) Entity Relationship Diagram for Instagram Database.



2) Relational Schema for Instagram Database.



FUNCTIONAL DEPENDENCIES AND NORMALIZATION

We shall go through each table discussing their functional dependencies and normalization.

1) users table.

   Functional dependencies:-

   id → username

   id → created_at

   Normalization:-

   - 1NF not violated, all the cells have atomic values.
   - 2NF not violated, no partial dependencies found.
   - 3NF not violated, no transitive dependencies found.

2) photos table.

   Functional dependencies:-

   id → image_url

   id → user_id (foreign key for users(id))

   id → caption

   id → posted_at

   Normalization:-

   - 1NF not violated, all the cells have atomic values.
   - 2NF not violated, no partial dependencies found, only one candidate key, that is only one prime attribute, id and one non-prime attribute, user_id.
   - 3NF not violated, no transitive dependencies found, id is a super key and user_id is the only prime attribute in the relation, id → user_id.

3) comments table.

   Functional dependencies:-

   id → comment_text

   id → user_id (foreign key for users(id))

   id → photo_id (foreign key for photo(id))

   id → commented_at

   Normalization:-

   - 1NF not violated, all the cells have atomic values.
   - 2NF not violated, no partial dependencies found, as there is only one candidate key.
   - 3NF not violated, no transitive dependencies found.

4) likes table.

Functional dependencies:-

(user_id, photo_id) → liked_at

Normalization:-

- 1NF not violated, all the cells have atomic values.
- 2NF not violated, no partial dependencies found, as both user_id and photo_id are both candidate keys.
- 3NF not violated, no transitive dependencies found.

5) follows table.

Functional dependencies:-

(follower_id, following_id) → followed_at

Normalization:-

- 1NF not violated, all the cells have atomic values.
- 2NF not violated, no partial dependencies found, as both follower_id and following_id are both candidate keys.
- 3NF not violated, no transitive dependencies found.

6) tags table.

Functional dependencies:-

id → tag_name

id → tagged_at

Normalization:-

- 1NF not violated, all the cells have atomic values.
- 2NF not violated, no partial dependencies found, as there is only one candidate key, id.
- 3NF not violated, no transitive dependencies found.

7) photo_tags table.

Functional dependencies:-

tag_id and photo_id.

There are functional dependencies for this table.

Normalization:-

- 1NF not violated, all the cells have atomic values.
- 2NF and 3NF not violated, as no partial and transitive dependencies found between photo_id and tag_id.

DATA DEFINITION LANGUAGE

1) Create a database called insta_db.

```
CREATE DATABASE insta_db;
```

2) Use that database.

```
USE insta_db;
```

3) Create tables with all the integrity constraints.

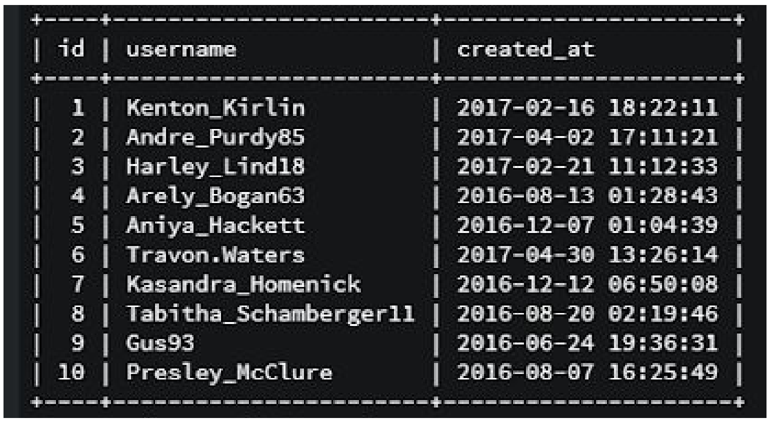
```
CREATE TABLE users (id INT AUTO_INCREMENT PRIMARY KEY,
        username VARCHAR(50) NOT NULL UNIQUE,
        created_at TIMESTAMP DEFAULT NOW());

CREATE TABLE photos (id INT AUTO_INCREMENTPRIMARY KEY,
        image_url VARCHAR(50) NOT NULL, user_id INT NOT NULL,
        caption VARCHAR(255), posted_at TIMESTAMP DEFAULT NOW(),
        FOREIGN KEY(user_id) REFERENCES users(id));

CREATE TABLE comments (id INT AUTO_INCREMENTPRIMARY KEY,
        comment_text VARCHAR(255) NOT NULL, user_id INT NOT NULL,
        photo_id INT NOT NULL, commented_at TIMESTAMP DEFAULT NOW(),
        FOREIGN KEY(user_id) REFERENCES users(id),
        FOREIGN KEY(photo_id) REFERENCES photos(id));

CREATE TABLE likes (user_id INT NOT NULL,
        photo_id INT NOT NULL, liked_at TIMESTAMP DEFAULT NOW(),
        FOREIGN KEY(user_id) REFERENCES users(id),
        FOREIGN KEY(photo_id) REFERENCES photos(id),
        PRIMARY KEY(user_id, photo_id));

CREATE TABLE follows (follower_id INT NOT NULL,
        following_id INT NOT NULL, followed_at TIMESTAMP DEFAULT NOW(),
        FOREIGN KEY(follower_id) REFERENCES users(id),
        FOREIGN KEY(following_id) REFERENCES users(id),
        PRIMARY KEY(follower_id, following_id));

CREATE TABLE tags (id INT AUTO_INCREMENTPRIMARY KEY,
        tag_name VARCHAR(100) UNIQUE,
        tagged_at TIMESTAMP DEFAULT NOW());

CREATE TABLE photo_tags (tag_id INT NOT NULL,
        photo_id INT NOT NULL, FOREIGN KEY(tag_id) REFERENCES tags(id),
            FOREIGN KEY(photo_id) REFERENCES photos(id),
            PRIMARY KEY(photo_id, tag_id));
```

4) Insert data into tables(just to show how to update).

```
INSERT INTO users (username, created_at) VALUES
            ('Kenton_Kirlin', '2017-02-16 18:22:10.846'),
            ('Andre_Purdy85', '2017-04-02 17:11:21.417');
INSERT INTO photos(image_url, user_id) VALUES
            ('http://elijah.biz', 1), ('https://shanon.org', 1);
INSERT INTO comments(comment_text, user_id, photo_id) VALUES
            ('unde at dolorem', 2, 1), ('quae ea ducimus', 3, 1);
INSERT INTO likes(user_id,photo_id) VALUES
            (2, 1), (5, 1);
INSERT INTO follows(follower_id, following_id) VALUES
            (2, 1), (2, 3);
INSERT INTO tags(tag_name) VALUES
            ('sunset'), ('photography');
INSERT INTO photo_tags(photo_id, tag_id) VALUES
            (1, 18), (1, 17);
```

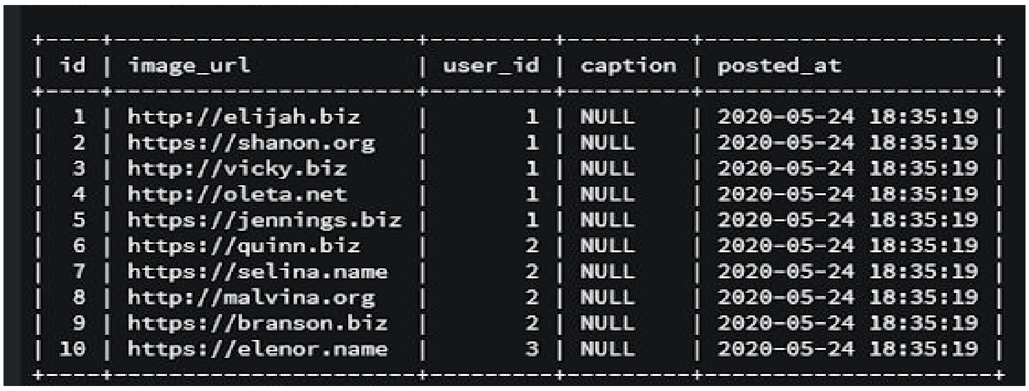
5) insta_db - DATABASE STATE

a) users table.



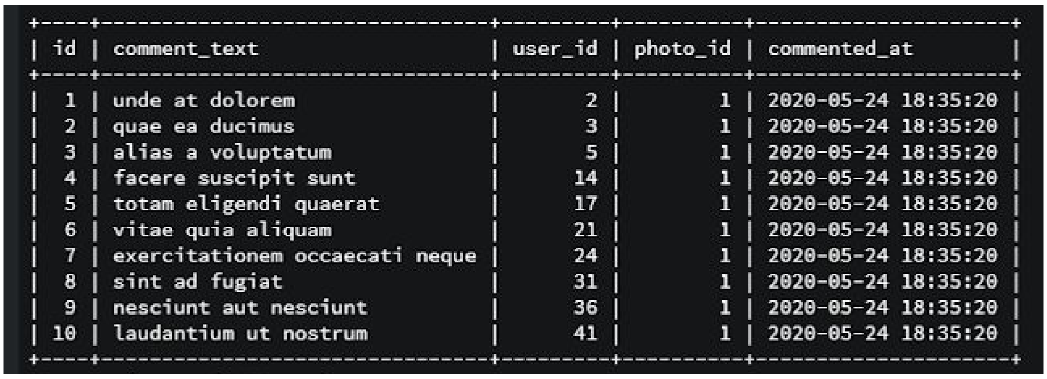
```
+----+----------------------+---------------------+
| id | username             | created_at          |
+----+----------------------+---------------------+
|  1 | Kenton_Kirlin        | 2017-02-16 18:22:11 |
|  2 | Andre_Purdy85        | 2017-04-02 17:11:21 |
|  3 | Harley_Lind18        | 2017-02-21 11:12:33 |
|  4 | Arely_Bogan63        | 2016-08-13 01:28:43 |
|  5 | Aniya_Hackett        | 2016-12-07 01:04:39 |
|  6 | Travon.Waters        | 2017-04-30 13:26:14 |
|  7 | Kasandra_Homenick    | 2016-12-12 06:50:08 |
|  8 | Tabitha_Schamberger11 | 2016-08-20 02:19:46 |
|  9 | Gus93                | 2016-06-24 19:36:31 |
| 10 | Presley_McClure      | 2016-08-07 16:25:49 |
+----+----------------------+---------------------+
```

b) photos table.



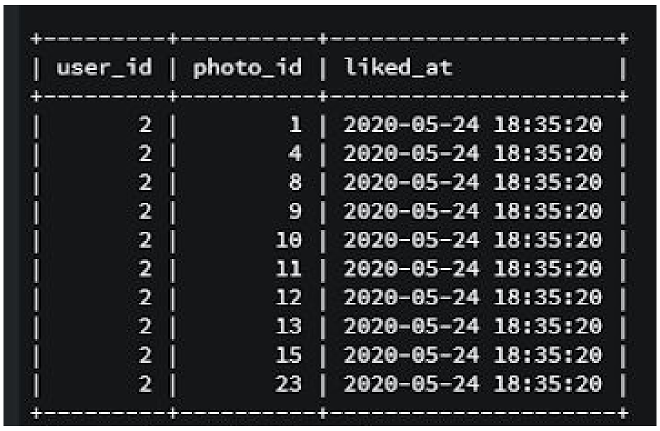
```
+----+-------------------------+---------+---------+---------------------+
| id | image_url               | user_id | caption | posted_at           |
+----+-------------------------+---------+---------+---------------------+
|  1 | http://elijah.biz       |       1 | NULL    | 2020-05-24 18:35:19 |
|  2 | https://shanon.org      |       1 | NULL    | 2020-05-24 18:35:19 |
|  3 | http://vicky.biz        |       1 | NULL    | 2020-05-24 18:35:19 |
|  4 | http://oleta.net        |       1 | NULL    | 2020-05-24 18:35:19 |
|  5 | https://jennings.biz    |       1 | NULL    | 2020-05-24 18:35:19 |
|  6 | https://quinn.biz       |       2 | NULL    | 2020-05-24 18:35:19 |
|  7 | https://selina.name     |       2 | NULL    | 2020-05-24 18:35:19 |
|  8 | http://malvina.org      |       2 | NULL    | 2020-05-24 18:35:19 |
|  9 | https://branson.biz     |       2 | NULL    | 2020-05-24 18:35:19 |
| 10 | https://elenor.name     |       3 | NULL    | 2020-05-24 18:35:19 |
+----+-------------------------+---------+---------+---------------------+
```

c) comments table.



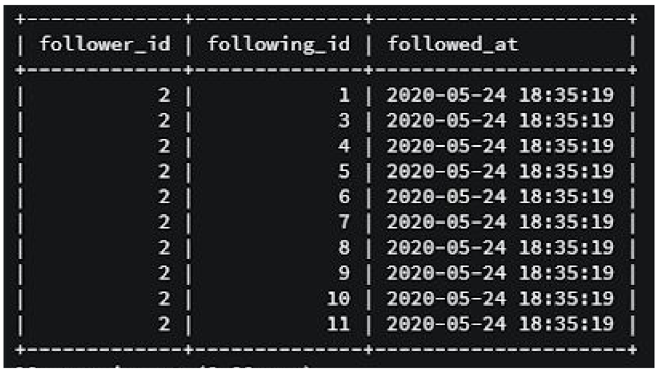
```
+----+-------------------------------+---------+----------+---------------------+
| id | comment_text                  | user_id | photo_id | commented_at        |
+----+-------------------------------+---------+----------+---------------------+
|  1 | unde at dolorem               |       2 |        1 | 2020-05-24 18:35:20 |
|  2 | quae ea ducimus               |       3 |        1 | 2020-05-24 18:35:20 |
|  3 | alias a voluptatum            |       5 |        1 | 2020-05-24 18:35:20 |
|  4 | facere suscipit sunt          |      14 |        1 | 2020-05-24 18:35:20 |
|  5 | totam eligendi quaerat        |      17 |        1 | 2020-05-24 18:35:20 |
|  6 | vitae quia aliquam            |      21 |        1 | 2020-05-24 18:35:20 |
|  7 | exercitationem occaecati neque |      24 |        1 | 2020-05-24 18:35:20 |
|  8 | sint ad fugiat                |      31 |        1 | 2020-05-24 18:35:20 |
|  9 | nesciunt aut nesciunt         |      36 |        1 | 2020-05-24 18:35:20 |
| 10 | laudantium ut nostrum         |      41 |        1 | 2020-05-24 18:35:20 |
+----+-------------------------------+---------+----------+---------------------+
```

d) likes table.



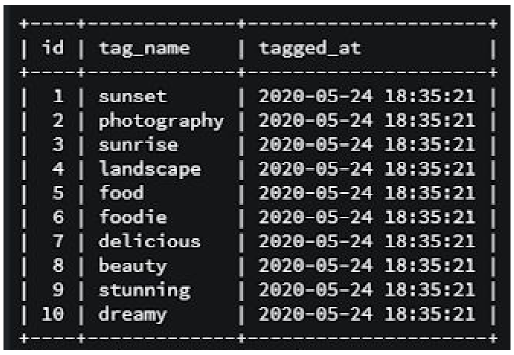
```
+---------+----------+---------------------+
| user_id | photo_id | liked_at            |
+---------+----------+---------------------+
|       2 |        1 | 2020-05-24 18:35:20 |
|       2 |        4 | 2020-05-24 18:35:20 |
|       2 |        8 | 2020-05-24 18:35:20 |
|       2 |        9 | 2020-05-24 18:35:20 |
|       2 |       10 | 2020-05-24 18:35:20 |
|       2 |       11 | 2020-05-24 18:35:20 |
|       2 |       12 | 2020-05-24 18:35:20 |
|       2 |       13 | 2020-05-24 18:35:20 |
|       2 |       15 | 2020-05-24 18:35:20 |
|       2 |       23 | 2020-05-24 18:35:20 |
+---------+----------+---------------------+
```

e) follows table.



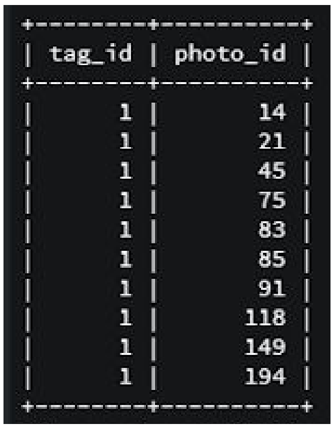
```
+-------------+--------------+---------------------+
| follower_id | following_id | followed_at         |
+-------------+--------------+---------------------+
|           2 |            1 | 2020-05-24 18:35:19 |
|           2 |            3 | 2020-05-24 18:35:19 |
|           2 |            4 | 2020-05-24 18:35:19 |
|           2 |            5 | 2020-05-24 18:35:19 |
|           2 |            6 | 2020-05-24 18:35:19 |
|           2 |            7 | 2020-05-24 18:35:19 |
|           2 |            8 | 2020-05-24 18:35:19 |
|           2 |            9 | 2020-05-24 18:35:19 |
|           2 |           10 | 2020-05-24 18:35:19 |
|           2 |           11 | 2020-05-24 18:35:19 |
+-------------+--------------+---------------------+
```

f) tags table.



```
+----+-------------+---------------------+
| id | tag_name    | tagged_at           |
+----+-------------+---------------------+
|  1 | sunset      | 2020-05-24 18:35:21 |
|  2 | photography | 2020-05-24 18:35:21 |
|  3 | sunrise     | 2020-05-24 18:35:21 |
|  4 | landscape   | 2020-05-24 18:35:21 |
|  5 | food        | 2020-05-24 18:35:21 |
|  6 | foodie      | 2020-05-24 18:35:21 |
|  7 | delicious   | 2020-05-24 18:35:21 |
|  8 | beauty      | 2020-05-24 18:35:21 |
|  9 | stunning    | 2020-05-24 18:35:21 |
| 10 | dreamy      | 2020-05-24 18:35:21 |
+----+-------------+---------------------+
```

g) photo_tags table.

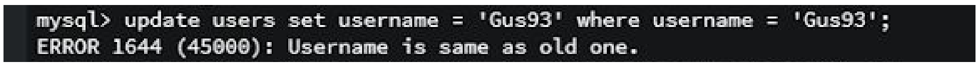


```
+--------+----------+
| tag_id | photo_id |
+--------+----------+
|      1 |       14 |
|      1 |       21 |
|      1 |       45 |
|      1 |       75 |
|      1 |       83 |
|      1 |       85 |
|      1 |       91 |
|      1 |      118 |
|      1 |      149 |
|      1 |      194 |
+--------+----------+
```

TRIGGERS

1) Trigger if the user updates his username same as the old one.

```
DELIMITER $$
CREATE TRIGGER must_not_update
BEFORE UPDATE ON users FOR EACH ROW
BEGIN
        IF NEW.username = OLD.username
        THEN
                SIGNAL SQLSTATE '45000'
                        SET MESSAGE_TEXT = "Username is same as old one.";
        END IF;
END;
$$
DELIMITER;
```

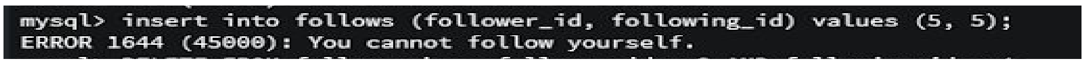


```
mysql> update users set username = 'Gus93' where username = 'Gus93';
ERROR 1644 (45000): Username is same as old one.
```

2) Trigger if the user follows himself.

```
DELIMITER $$
CREATE TRIGGER prevent_self_follows
BEFORE INSERT ON follows FOR EACH ROW
```

```
BEGIN
        IF NEW.follower_id = NEW.following_id
        THEN
                SIGNAL SQLSTATE '45000'
                                SET MESSAGE_TEXT = "You cannot follow yourself.";
        END IF;
END;
$$
DELIMITER;
```



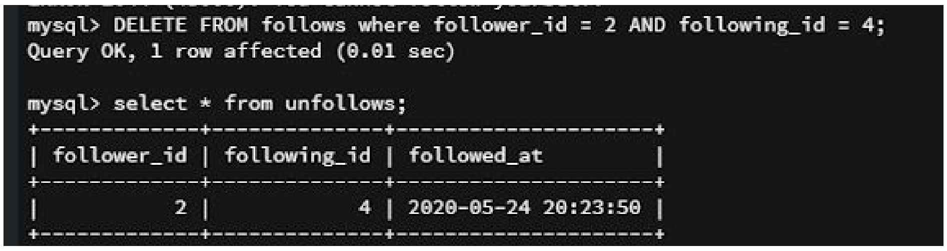
```
mysql> insert into follows (follower_id, following_id) values (5, 5);
ERROR 1644 (45000): You cannot follow yourself.
```

3) Trigger if the user unfollows.

Before that we need to create a new logger-trigger-table.

```
CREATE TABLE unfollows (follower_id INT NOT NULL,
        following_id INT NOT NULL, unfollowed_at TIMESTAMP DEFAULT NOW(),
        FOREIGN KEY(follower_id) REFERENCES users(id),
        FOREIGN KEY(following_id) REFERENCES users(id),
        PRIMARY KEY(follower_id, following_id));
```

```
DELIMITER $$
CREATE TRIGGER capture_unfollow
AFTER DELETE ON follows FOR EACH ROW
BEGIN
        INSERT INTO unfollows(follower_id, following_id)
        VALUES (OLD.follower_id, OLD.following_id);
END;
$$
DELIMITER;
```

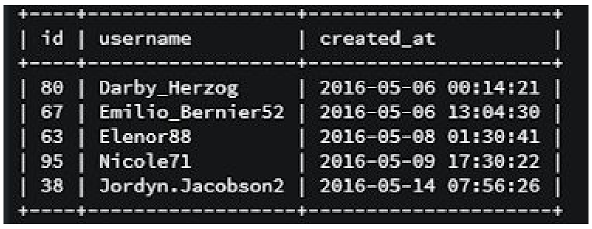


```
mysql> DELETE FROM follows where follower_id = 2 AND following_id = 4;
Query OK, 1 row affected (0.01 sec)

mysql> select * from unfollows;
+-------------+--------------+---------------------+
| follower_id | following_id | followed_at         |
+-------------+--------------+---------------------+
|           2 |            4 | 2020-05-24 20:23:50 |
+-------------+--------------+---------------------+
```

MySQL QUERIES

1) Finding 5 oldest users.

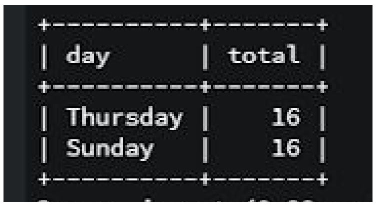
```
SELECT * FROM users ORDER BY created_at LIMIT 5;
```



```
+----+------------------+---------------------+
| id | username         | created_at          |
+----+------------------+---------------------+
| 80 | Darby_Herzog     | 2016-05-06 00:14:21 |
| 67 | Emilio_Bernier52 | 2016-05-06 13:04:30 |
| 63 | Elenor88         | 2016-05-08 01:30:41 |
| 95 | Nicole71         | 2016-05-09 17:30:22 |
| 38 | Jordyn.Jacobson2 | 2016-05-14 07:56:26 |
+----+------------------+---------------------+
```

2) Most Popular Registration Date.

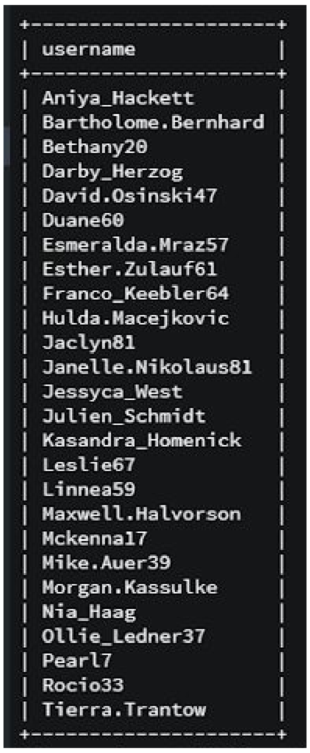
SELECT DAYNAME(created_at) AS day, COUNT(*) AS total FROM users GROUP BY day ORDER BY total DESC LIMIT 2;



```
+----------+-------+
| day      | total |
+----------+-------+
| Thursday |    16 |
| Sunday   |    16 |
+----------+-------+
```

3) Identify Inactive Users (users with no photos).

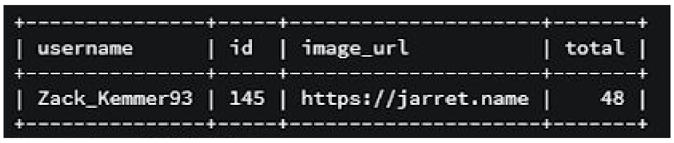
SELECT username FROM users LEFT JOIN photos ON users.id = photos.user_id WHERE photos.id IS NULL;



```
+---------------------+
| username            |
+---------------------+
| Aniya_Hackett       |
| Bartholome.Bernhard |
| Bethany20           |
| Darby_Herzog        |
| David.Osinski47     |
| Duane60             |
| Esmeralda.Mraz57    |
| Esther.Zulauf61     |
| Franco_Keebler64    |
| Hulda.Macejkovic    |
| Jaclyn81            |
| Janelle.Nikolaus81  |
| Jessyca_West        |
| Julien_Schmidt      |
| Kasandra_Homenick   |
| Leslie67            |
| Linnea59            |
| Maxwell.Halvorson   |
| Mckenna17           |
| Mike.Auer39         |
| Morgan.Kassulke     |
| Nia_Haag            |
| Ollie_Ledner37      |
| Pearl7              |
| Rocio33             |
| Tierra.Trantow      |
+---------------------+
```

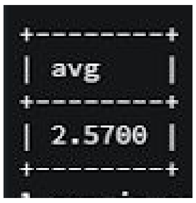
4) Identify the most popular photo (and user who created it).

SELECT username, photos.id, photos.image_url, COUNT(*) AS total FROM photos INNER JOIN likes ON likes.photo_id = photos.id INNER JOIN users ON photos.user_id = users.id GROUP BY photos.id ORDER BY total DESC LIMIT 1;


```
+---------------+-----+---------------------+-------+
| username      | id  | image_url           | total |
+---------------+-----+---------------------+-------+
| Zack_Kemmer93 | 145 | https://jarret.name |    48 |
+---------------+-----+---------------------+-------+
```

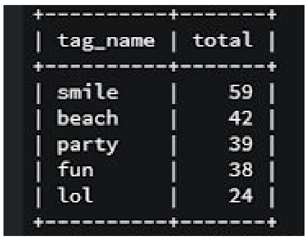
5) Calculate average number of photos per user.

SELECT (SELECT Count(*) FROM photos)/(SELECT Count(*) FROM users) AS avg;


```
+--------+
| avg    |
+--------+
| 2.5700 |
+--------+
```

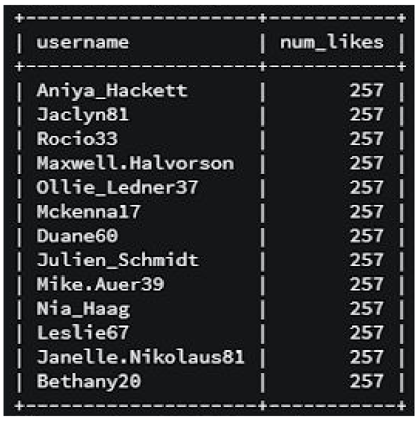
6) Find the five most popular hashtags.

SELECT tags.tag_name, Count(*) AS total FROM photo_tags JOIN tags ON photo_tags.tag_id = tags.id GROUP BY tags.id ORDER BY total DESC LIMIT 5;


```
+----------+-------+
| tag_name | total |
+----------+-------+
| smile    |    59 |
| beach    |    42 |
| party    |    39 |
| fun      |    38 |
| lol      |    24 |
+----------+-------+
```

7) Finding the bots - the users who have liked every single photo.

SELECT username, Count(*) AS num_likes FROM users INNER JOIN likes ON users.id = likes.user_id GROUP BY likes.user_id HAVING num_likes = (SELECT Count(*) FROM photos);


```
+--------------------+-----------+
| username           | num_likes |
+--------------------+-----------+
| Aniya_Hackett      |       257 |
| Jaclyn81           |       257 |
| Rocio33            |       257 |
| Maxwell.Halvorson  |       257 |
| Ollie_Ledner37     |       257 |
| Mckenna17          |       257 |
| Duane60            |       257 |
| Julien_Schmidt     |       257 |
| Mike.Auer39        |       257 |
| Nia_Haag           |       257 |
| Leslie67           |       257 |
| Janelle.Nikolaus81 |       257 |
| Bethany20          |       257 |
+--------------------+-----------+
```

CONCLUSION

- Capabilities:-

This database is structured so flexible and simple, that the queries can be easily queried to the database to get the data we require so easily. It can also be added with new tables without any discrepancies.

- Limitations:-

  This database without fail requires a front-end to operate as a whole application as we cannot get the feel of the working and if integrated with the front-end part, this database can do wonders, as we can see the users, post, likes, comments.

- Future Enhancements:-
  1. Integrate with the front-end part, and make it attractive and effective.
  2. Add new tables like chats, activities of the users so that the user can communicate with the other user without using any other application.