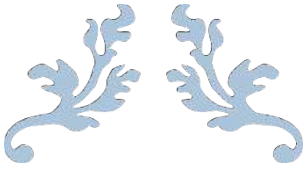


CSE 608 - DATABASE SYSTEMS PROJECT REPORT

#Tagged

KRIT GUPTA

UIN : 927001565

Texas A&M University, College Station

INDEX

PROJECT DESCRIPTION

Background

I am an avid user of various social media applications, Facebook being one of them. Facebook is an application which has various users and each user, apart from other functionalities, has many pictures on which they can post and comment. I want to target this segment through the project.
I have always felt that Facebook has a picture display interface that is not yet complete and they are evolving it to match the customer needs. Users, like me, who use other web applications like Instagram, would like to see similar features on Facebook. Facebook has recently started using 'Tags' for their pictures, wherein, similar to Instagram, a user can tag a picture with a short word or a phrase to depict the main theme. However, functionality that sorts the pictures according to different tags is lacking, wherein, clicking on a tag can provide other pictures having the same tag. One reason might be that Facebook is not focused only on displaying pictures. However, I for one, would like to see that feature.
I would prefer that on clicking on a tag, I get to see all the pictures (amongst my Facebook friends perhaps!) with the same tags – something similar to what Clustering Algorithms do! However, I want to do this with a database design.

This got me curios and I decided to try and implement it through this application. Designing and building a database which implements this functionality is the purpose of this application.
I wanted a database which has the data of my Facebook friends, their pictures with the comments, likes and tags associated with each picture and user. I would like to know which all pictures are associated with the same tag so that I can go through them at one go, instead of going to different profiles and checking out pictures with the same theme.
Since, Instagram provides this functionality, I have tried to imitate Instagram's database and have attempted to recreate a small-scale model of how, I perceive their database would be designed and organized.

In simple words, this project is trying to store data generated on Facebook (particularly related to the pictures) in an Instagram-inspired database, so that operations can be performed on it.

Functions and Services

This application has the main function of displaying all the pictures associated with the same tag and also to show the number of pictures associated with the tag. The major aim is to display the data of the Facebook friends I have and sort their pictures according to tags.

Apart from that, it has the functionality of:

- Displaying total current users
  The first line of the homepage shows the total number of current users, total pictures and total comments. The said numbers change with any new addition or deletion of user. This functionality is achieved by triggering live SQL queries with any addition or deletion.

- Adding a User
  The application only adds the username of a new user and not his/her pictures or comments. Adding them would clutter the User Interface (UI) and would need more knowledge about User Design and Experience and Web Development.
  I have simply given the option of adding a user to show that similar adding functionality can be, in future, expanded to adding pictures with their comments and tags.
  Again, adding a User will increase the total User count on the first line by 1.
  Needless to say that similar queries can be run for Updating the User and adding pictures, comments, likes and tags and updating them. Since, that was not the primary function of the project, I have tried to keep the UI simple and kept only the essentially needed functionality for the project.

- Deleting a User
  The application gives the functionality, wherein, on deleting a user, by his ID, the entire records with all his/her associated pictures, comments and tags will be deleted. 'ON DELETE CASCADE' really came in handy here!
  Of course, all users with their IDs can be seen with a button click on a separate screen. Deletion of a user updates the total count of the users, pictures and comments on the first line of the webpage.
  The reason why deletion and further functionalities are triggered by IDs and not the username is that we can have the same username, but the Database will ensure that the IDs remain unique and only the needed deletion is done.

- See all pictures by a User
  The pictures associated with a User (through their User ID) can be seen on a separate screen with a button click.

- Counting total pictures in the database
  Though the first line of the webpage displays all the pictures in the database. I have shown the functionality that through a button click, one can see the count of the total pictures currently in the database.
  This is primarily, because, this functionality can be expanded in the future to other features, if need be.

- Count total pictures associated with a tag
  The total number of pictures associated with a single tag can be seen with a button click on the same page, just below the related button.
  Of course, all the tags with their IDs can be seen first on a separate screen.

- See all pictures associated with a tag
  All the pictures associated with the same tag can be seen on another screen with a button click. All one has to do is enter the Tag ID.

Since the main functionality of the application is to display pictures with the same tag, wherein tags essentially start with the character '#', I named the application – ***'#Tagged'.***

The URL of the project is: https://infinite-hamlet-44584.herokuapp.com/

DATA COLLECTION

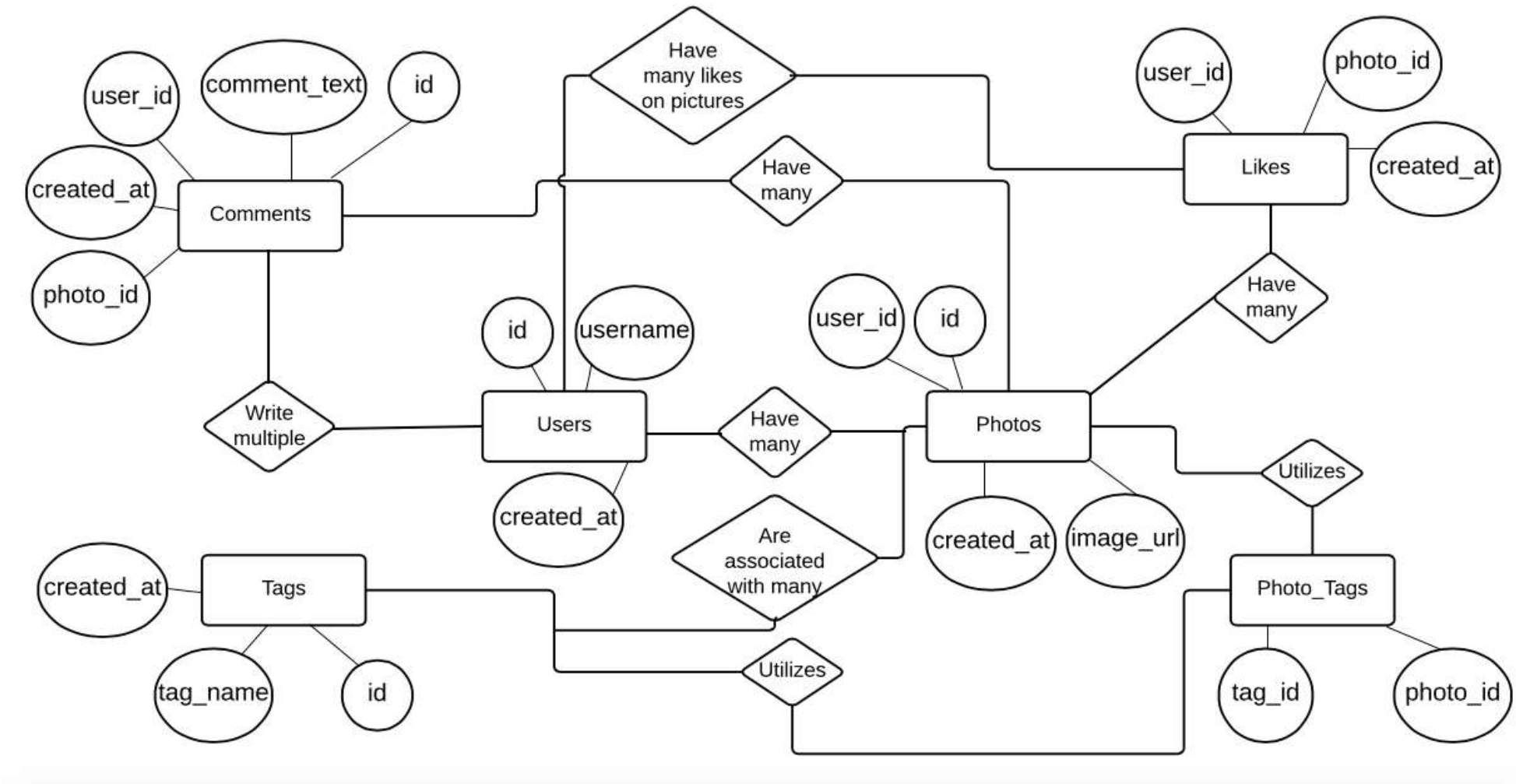
I initially wanted to web-scrape real data from my Facebook profile. I actually wrote the script in Python to scrape real data from Facebook. This was not a major challenge, since I have some experience with developing automation bots and web scraping.
However, as per my view and further discussions with the project TA, I needed to have the consent of all my Facebook friends, since I would be using their data and maybe, breaking a few laws on data security if I did otherwise.

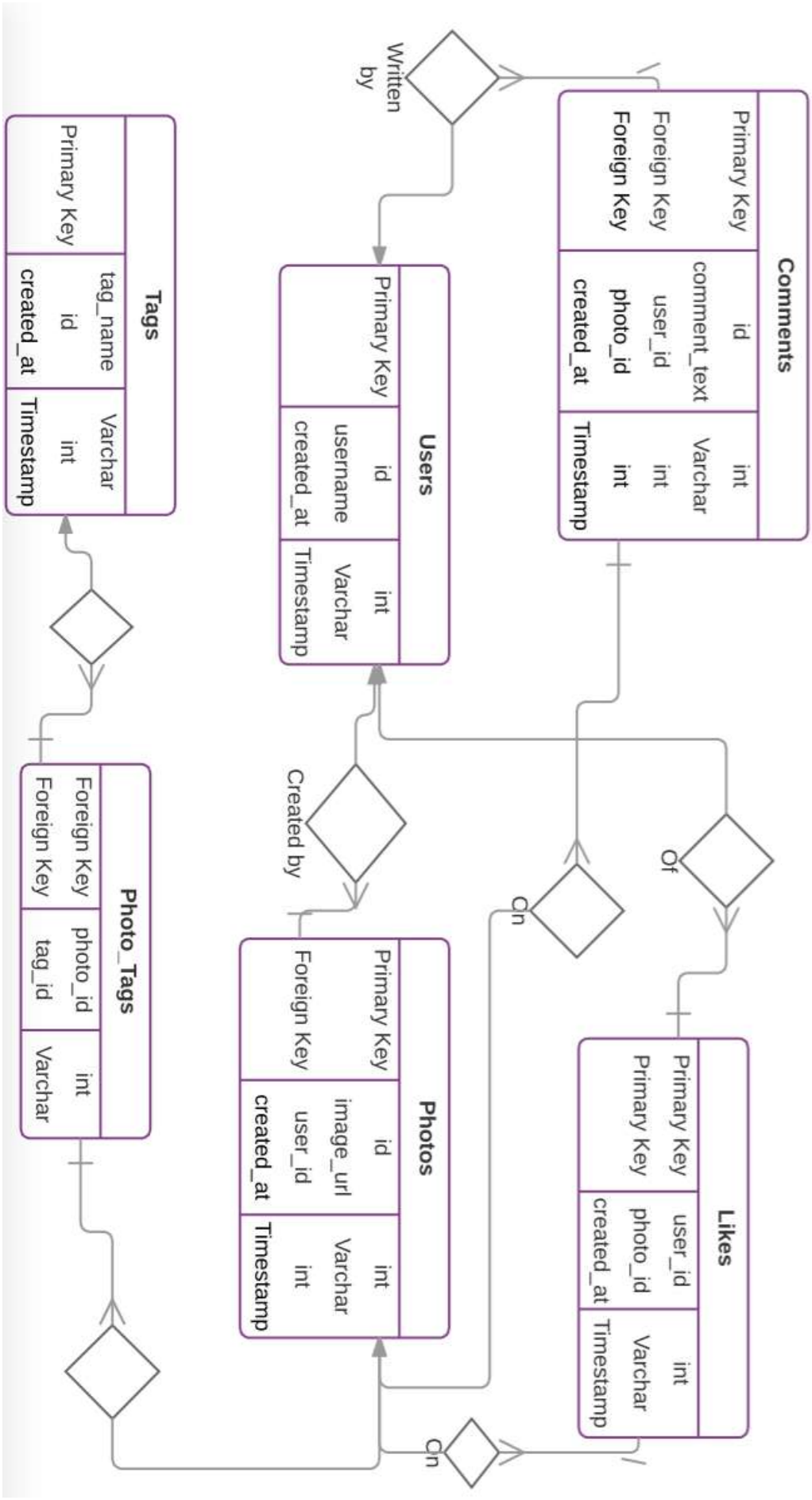
Hence, I decided to generate fake data, keeping in mind how data is generated on Facebook. I used the free library – 'Faker' which generates as much fake data as one wants with a few lines of code – a simple 'for' loop and all set to go.
The data generated is not all in English, comments are in a different language (Portuguese perhaps!). However, it fulfills my role of imitating Instagram's database with similar data as generated on Facebook. I generated 100 users with a total of 280 pictures and 7500 comments. Tags associated were 15 (I typed them in myself, just to not lose the feel of reality). I lost a few chunks of data while testing out my application's UI.

I am attaching the Python script and the related Excel file (the script can be modified to generate tab related files as well) which the script generates that has all the data scraped with the project, in case, anyone (even me perhaps) decides to takes the project further with real data from their Facebook profiles. (Annexure – I)

In future, I would like to integrate this script with the project so that every time someone opens up this application – the application automatically scrapes the data from the Facebook profile and populates the database with live data. Clearly, one has to be logged in to the affiliated Facebook profile or give the script or the application his/her Facebook Username and Password for it to scrape live data.

In the script, I have presented scraping of a local news Facebook page and not my Facebook friend list for data security and privacy reasons. I show only a single page of the related Excel file just to show the structure of the data scraped.
The data has no relevance to the project otherwise, except for if expanded in the future to live data from Facebook. The code can be easily modified to scrape the entire friend list later on.


Have
many likes
on pictures
user_id
comment_text
id
user_id
photo_id
created_at
Comments
Have
many
Likes
created_at
photo_id
Have
many
id
username
user_id
id
Write
multiple
Users
Have
many
Photos
Utilizes
created_at
Are
associated
with many
created_at
image_url
created_at
Tags
Photo_Tags
tag_name
id
Utilizes
tag_id
photo_id


Comments
Primary Key
id
int
Foreign Key
comment_text
Varchar
Foreign Key
user_id
int
photo_id
int
created_at
Timestamp
Written by
Users
Primary Key
id
username
created_at
Timestamp
int
Varchar
Tags
Primary Key
tag_name
id
created_at
Varchar
int
Timestamp
Photo_Tags
Foreign Key
Foreign Key
photo_id
tag_id
int
Varchar
Created by
Photos
Primary Key
Foreign Key
id
image_url
user_id
created_at
int
Varchar
int
Timestamp
On
Of
On
Likes
Primary Key
Primary Key
user_id
photo_id
created_at
int
Varchar
Timestamp
On

ENTITY- RELATIONSHIP (ER) DIAGRAM

The diagram is made with the online, freely available website called– LucidChart.com. I am a novice at making ER Diagrams and the one I made was looking complex, hence, I decided to make a Relational Schema Diagram as well, though it was not required, for better clarity and understanding of the model I intend to design and the database I want to build.
I have tried to replicate the the type of data that is generated on Facebook – Users, Pictures, Comments, Likes and Tags on pictures. After doing so, I have attempted to design a database, which in my opinion is similar to what Instagram might use and connect various elements.

Keeping this in mind, I made separate tables for

- Users – which will record the '*username',* Unique *'id'* (will be referenced by other tables) and '*created_at'* (Timestamp) for the User.
- Photos – which will keep the data of the pictures posted by a particular user. It has a Unique '*id'* (will be referenced by other tables) for the photo stored, an '*image_url'*, a '*created_at'* (Timestamp) and a '*user_id'* (which references the *id* of the Users) so as to tell which user posted this picture, which is used in the UI.
- Comments – This table stores the comments on a particular picture and also who wrote the that comment. This is how I portray Facebook or Instagram will be keeping tabs on who wrote a particular comment. Except for the Unique '*id'* (for comment), '*comment_text'* and '*created_at',* it has a '*user_id'* column (which references the '*id'* in the Users table) and a '*photo_id'* (which references the '*id'* in the Photos table).
- Likes – This is a feature of Facebook (Liking a picture!). I made 2 PRIMARY KEYS in this table ('*user_id'* and '*photo_id'*) to ensure the functionality that a particular user does not like a particular picture more than once. This functionality is observed both at Facebook and Instagram.
- Tags – It has a '*tag_name',* a unique *'id'* (or the tag_id) and '*created_at'*.
- Photo_Tags – This has 2 columns – '*photo_id'* and '*tag_id'* to reference the Tags and the Photos table respectively.

The ER Diagram has a different table for Users, Photos, Comments, Likes and Tags.
A separate table Photo_Tags has been made to link Tags and Photos keeping in mind the normalization. This table has only two columns, *'tag_id'* and *'photo_id'* – both FOREIGN KEYS, which reference the *'id'* in the Tags table and the *'id'* in the Photos table respectively. This will fetch the Photos attached to a particular Tag with a simple NATURAL JOIN.

TABLE NORMALIZATION

I will be honest and state that I built the schema first and made the tables before I made the ER diagram. I understand that it should have been done the other way round. It took me a couple of hours to finally decipher how I should make the various tables and how I should connect them. The connections were simple as I have a live working final product of what I was trying to replicate – Facebook and Instagram.

Tags Table

The *'Tags'* table, in the first iteration, included 4 columns and had *'photo_id'* in it, which referenced the *'id'* in the *'Photos'* table. However, keeping in acceptance to Boyce-Codd Normal Form (BCNF), I decided to split it into two tables – *'Tags'* and a separate table – *'Photo_Tags'* which was a connecting table to *'Tags'* and *'Photos'* having both its columns – *'photo_id'* and *'tag_id'* as Foreign Keys, one each for referencing either table.

Likes Table

In the *'Likes'* table, I had to ensure the functionality that a User does not like the picture more than once. This was essential as, on social media, a picture's worth or credibility is measured by how many likes the picture has received. So, if one allows multiple likes on the picture by the same person, this valuation will be seriously hit!

I achieved this by defining both the *'user_id'* and *'photo_id'* columns as PRIMARY KEYS, which will ensure that a User gets to like a picture only once.

Also, there is no *'id'* column in the *'Likes'* table, as I never needed to reference it any where else.

The sole purpose of the *'Likes'* table is to display this thinking, though, the table does not hold a major significance for the project.

I have attached a screen shot of the sample brainstorming session (rough ideas generated in the process) about the structure of the tables in the database for reference. **(Annexure – II)**

From the Relational Schema Diagram, one can see that:

- An attribute does not have multiple values.
- Every non-key attribute is dependent on the Primary Key.
- No non-key attribute is transitively dependent on the Primary Key.

USER INTERFACE

Backend – Node.js

I used Node.js for developing the backend of the application. Though, I am learning Ruby on Rails for another course and developing a web application for that class, I decided to use Node.js for this application, just to learn the basics of the framework. I had no previous experience of Node.js before this project.

Development Environment – Cloud 9

For the IDE, I used the free Cloud 9 – and developed my entire application there; generated the fake data there; made the entire backend with the database and the routes, models and views; and connected all the views of the application to the backend.

Frontend – HTML and CSS

I developed the frontend of the application using HTML for structuring and CSS for beautifying! I built the entire code on Cloud 9 and connected it to the backend on Cloud 9 only.

Deployment

I deployed my application on Heroku. The reason for doing this was that I wanted to expand this application even after the class and would like to actually use this personally in future. CSNET might pull down my application once I graduate. Heroku gives me the ability to continuously update and use the application without any fear and for free.
I used the ClearDB add-on for making my MySQL database and connecting it to the server. Clear DB provides a 5MB free database with the Heroku application. Since, my Facebook friend list is quite small, 5MB would be more than enough for the purpose.

Usage of the application

The application is a very simple web app, essentially made to support my imitation of Instagram's database. The data is generated keeping in mind the kind of data that is generated on Facebook.
Simply put, it is about storing data generated on Facebook in an Instagram styled database.
The application sorts the images based on the tag associated with it and gives the user a list of images associated with the ID.

I did not build this system for a wide demography, though it can be scaled if needed. The project is more personal in the sense that I built it for solving a personal problem that I face – seeing all the Tagged pictures in one place on Facebook, rather than going on to different profiles.
I have intentionally kept the UI simple, focused on a single problem. This was primarily because of two reasons – one, because the UI would have become very cluttered and disorganized if I added more functionality like – Update and add pictures, etc. The other reason is that, I currently lack the knowledge about UI design, User Experience Design and Web Development to build a UI that can have added functionalities and also, keep the UI neat and user-friendly.

Having stated that, I would like to comment that building additional functionalities in the UI will be quite simple and most probably, only repetitive of the functionalities I have provided as of now. Some cases in point are – Adding a picture, adding a comment or updating a picture or a comment through the UI. These are all simple SQL queries, which are no different than Adding a new User – something that has already been provided as an example.

I have tried to incorporate sample functionalities – Adding a new User, Counting the total pictures, Seeing all the pictures by a User – just to portray that I can currently include such functionalities in the project and similar functionalities can be included in the future if the need arises.
I presently did not see any need for adding these functionalities as of now and hence, did not include them in the project.
The usage is simple with a single web-page layout.

Though I have given a major description of the UI in the Project Description section (pages 3-5) of the report, I would like to give an overview once again.
I would like to, in future, include the web-scraping Python script in the project, which automatically scrapes the data live from Facebook and feed it to the database. For now, I have made peace with fake generated data, generated through the library 'Faker'. I have ensured that the data is consistent with the one generated on Facebook.

The first line displays the count of all the live users, pictures and comments present in the database.
Adding a user is done with a text field (takes in the username) and a button. The database, automatically generates a Unique User_ID for the new user.
All the users in the database can be seen with their IDs on a separate page.
All pictures posted by a User can be seen with a text field (takes in the unique User_ID) and a button on a separate page.
Deleting a user is achieved with a text field (takes in the unique User_ID) and a button. Deleting a user deletes his/her photos, comments, likes and tags.
Despite giving a total count of the pictures on the first line, I have tried to give the same functionality with a button click as well.

All Tags can be seen alongside their Tag_ID on a separate page.
The count of the pictures associated with a Tag (through the unique Tag_ID) can be seen with a button click on the same page, just below the text field.
All the pictures associated with a Tag (through the unique Tag_ID) can be seen on a separate page with a button click.

DISCUSSION

Future expansion

The project has been an immense learning process. Keeping in sync with the ideology that 'Learning is never-ending' I have tried to keep the project unfinished (something inspired from Facebook!) and would like to include other functionalities in the future.

- I would like to integrate the web-scraping Python script, so that I can scrape live data from Facebook and work on that, rather than on fake generated data.
- I would like to make the UI more appealing and simplified.
- Possibly, utilize Data Analytics in the future to give more intelligent statistics about the data – like what tags are in most use currently, and which friend posted the maximum number of pictures and wrote the maximum comments in the last week.
- I would like to build a system that tackles automation bots – which make fake profiles and like and comment on pictures only to increase the value of the picture. (In social media, a picture's value or worth is calculated, not by its aesthetic credibility, but by the number of likes and comments on it.) The system would take into account the activity levels of all the profiles, and try to predict which profiles are fake and automated by bots, rather than real people.

Difficulties and Learnings

I faced a number of difficulties in the process.

- I had no previous experience of Node.js for developing the backend. Though I am learning Ruby on Rails and developing a web application with it for another class, I decided to use Node.js in order to learn another language and get my hands dirty with it.
- It was really challenging to integrate Node.js with MySQL because Node.js supports PostgreSQL and integrating MySQL needed an add-on – ClearDB, provided on Heroku.
- ClearDB kept on disconnecting the server connection in the initial tries, if I kept the project idle for some time. It was raising a Connection error. I dealt with the error by catching it and writing code for what to do if an error is raised.
- Finding the library to generate fake and good data was a challenge. I considered several libraries before I decided to use Faker with Node.js to generate the data.
- Auto_Increment for the PRIMARY_KEY in ClearDB increases by 10 and not by 1. This was really confusing for me in the first try and it was offsetting my data, which I had prepared with an Auto_Increment of 1. I could not change it as well - @@auto_increment did not give me the authorization to update the Auto_Increment. Hence, I had to write a small additional script to change the offset for all my tables which have an ID tag after inserting the data in the ClearDB Database on Heroku.
- With the help of MySQL Workbench, I had to manually update the 'user_id' entries in the Photos table, along with Likes and Tags to make some meaningful relations. Assigning

random numbers to these entries was creating difficulty to create good relations amongst the different tables. This one of the most time consuming endeavor in the entire project.

- Through this app, I got to learn the basics of Node.js and connecting it to ClearDB server on Heroku.
- I learnt how much I need to learn about User Experience design for building better and scalable applications.

ANNEXURE I

(I have deliberately put the script, font and layout of the code as that found on stackoverflow, since it is now drilled in our cognitive systems that this signifies code!)

```
    import urllib2
    import json
    import datetime
    import csv
    import time



    app_id = "490911914596812"
    app_secret = "80ec487685fcaf3f5ff4442a6e89811b"

    access_token = app_id + "|" + app_secret
    #access_token =
"EAAGZBe1ZAJGcwBAJdRMZADofSu3Pdmv61ExmI5VndSvj2dhwZCZAkChTDftycRLcJWuloJkaZC26oFuqaU9
hZC"


    page_id = '13310147298'


    def testFacebookPageData(page_id, access_token):
    # construct the URL string
    #print "inside func"
    base = "https://graph.facebook.com/v2.10"
    node = "/" + page_id
    parameters = "/?access_token=%s" % access_token
    url = base + node + parameters
    print url

    # retrieve data
    req = urllib2.Request(url)
    response = urllib2.urlopen(req)
    print response
    data = json.loads(response.read())

    print json.dumps(data, indent=4, sort_keys=True)

testFacebookPageData(page_id, access_token)


def request_until_succeed(url):
    req = urllib2.Request(url)
    success = False
    while success is False:
        try:
            response = urllib2.urlopen(req)
            if response.getcode() == 200:
                success = True
        except Exception, e:
            print e
            time.sleep(5)

            print "Error for URL %s: %s" % (url, datetime.datetime.now())
```

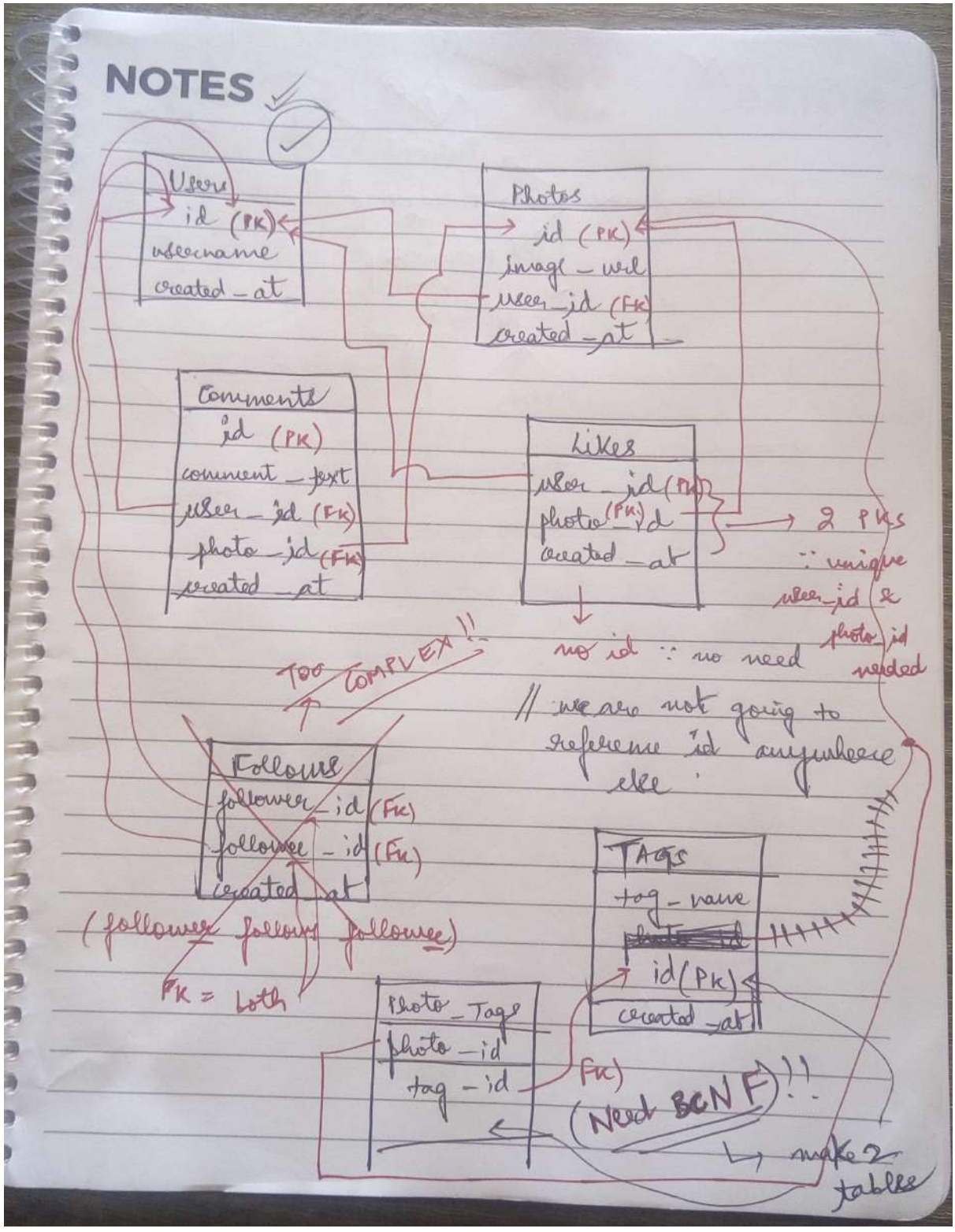
```
    return response.read()


def testFacebookPageFeedData(page_id, access_token):
    # construct the URL string
    base = "https://graph.facebook.com/v2.10"
    node = "/" + page_id + "/feed"  # changed
    parameters = "/?access_token=%s" % access_token
    url = base + node + parameters
    #print url

    # retrieve data
    data = json.loads(request_until_succeed(url))

    #print json.dumps(data, indent=4, sort_keys=True)


testFacebookPageFeedData(page_id, access_token)


def getFacebookPageFeedData(page_id, access_token, num_statuses):
    # construct the URL string
    base = "https://graph.facebook.com"
    node = "/" + page_id + "/feed"
    parameters =
"/?fields=message,link,created_time,type,name,id,likes.limit(1).summary(true),comment
s.limit(1).summary(true),shares&limit=%s&access_token=%s" % (
    num_statuses, access_token)  # changed
    url = base + node + parameters
    print url

    # retrieve data
    data = json.loads(request_until_succeed(url))

    return data


test_status = getFacebookPageFeedData(page_id, access_token, 1)["data"][0]
#print json.dumps(test_status, indent=4, sort_keys=True)


def processFacebookPageFeedStatus(status):
    # The status is now a Python dictionary, so for top-level items,
    # we can simply call the key.

    # Additionally, some items may not always exist,
    # so must check for existence first

    status_id = status['id']
    status_message = '' if 'message' not in status.keys() else
status['message'].encode('utf-8')
    link_name = '' if 'name' not in status.keys() else status['name'].encode('utf-8')
    status_type = status['type']
    status_link = '' if 'link' not in status.keys() else status['link']

    # Time needs special care since a) it's in UTC and
    # b) it's not easy to use in statistical programs.
```

```
    status_published = datetime.datetime.strptime(status['created_time'], '%Y-%m-
%dT%H:%M:%S+0000')
    status_published = status_published + datetime.timedelta(hours=-5)  # EST
    status_published = status_published.strftime('%Y-%m-%d %H:%M:%S')  # best time
format for spreadsheet programs

    # Nested items require chaining dictionary keys.

    num_likes = 0 if 'likes' not in status.keys() else
status['likes']['summary']['total_count']
    num_comments = 0 if 'comments' not in status.keys() else
status['comments']['summary']['total_count']
    num_shares = 0 if 'shares' not in status.keys() else status['shares']['count']

    # return a tuple of all processed data
    return (status_id, status_message, link_name, status_type, status_link,
            status_published, num_likes, num_comments, num_shares)

processed_test_status = processFacebookPageFeedStatus(test_status)
print processed_test_status


def scrapeFacebookPageFeedStatus(page_id, access_token):
    with open('%s_facebook_statuses.csv' % page_id, 'wb') as file:
        w = csv.writer(file)
        w.writerow(["status_id", "status_message", "link_name", "status_type",
"status_link",
                    "status_published", "num_likes", "num_comments", "num_shares"])

        has_next_page = True
        num_processed = 0  # keep a count on how many we've processed
        scrape_starttime = datetime.datetime.now()

        print "Scraping %s Facebook Page: %s\n" % (page_id, scrape_starttime)

        statuses = getFacebookPageFeedData(page_id, access_token, 100)

        while has_next_page:
            for status in statuses['data']:
                w.writerow(processFacebookPageFeedStatus(status))

                # output progress occasionally to make sure code is not stalling
                num_processed += 1
                if num_processed % 1000 == 0:
                    print "%s Statuses Processed: %s" % (num_processed,
datetime.datetime.now())

            # if there is no next page, we're done.
            if 'paging' in statuses.keys():
                statuses =
json.loads(request_until_succeed(statuses['paging']['next']))
            else:
                has_next_page = False

        print "\nDone!\n%s Statuses Processed in %s" % (num_processed,
datetime.datetime.now() - scrape_starttime)


scrapeFacebookPageFeedStatus(page_id, access_token)
```

Output Result

	A	B	C	D	E	F	G	H	I
1	status_id	status_message	link_name	status_type	status_link	status_published	num_likes	num_comments	num_shares
2	13310147298_1015	In case you missed y	Actor Harry Dean Sta	link	http://on-ajc.com/2fi	2017-09-17 23:50:39	0	1	1
3	13310147298_1015	Georgia Tech studer	5 students robbed ne	link	http://on-ajc.com/2fi	2017-09-17 23:35:13	0	0	1
4	13310147298_1015	Some of Georgia's b	Which companies re	link	http://on-ajc.com/2fi	2017-09-17 23:19:32	6	1	0
5	13310147298_10155749904982299		Exclusive: Mom of G	link	http://on-ajc.com/2fs	2017-09-17 23:03:11	4	2	4
6	13310147298_1015	An argument among	Argument among 20	link	http://on-ajc.com/2x	2017-09-17 22:45:13	2	0	5
7	13310147298_1015	After serving close to	How the Atlanta Bra	link	http://on-ajc.com/2xl	2017-09-17 22:28:12	146	3	31
8	13310147298_10155749789667299		Couple stressed over	link	http://on-ajc.com/2fi	2017-09-17 22:10:56	4	0	5
9	13310147298_10155749764842299		Opinion: Is Trump a	link	http://on-ajc.com/2xl	2017-09-17 21:53:02	9	1	2
10	13310147298_1015	Is this the city you ca	This city was ranked	link	http://on-ajc.com/2x	2017-09-17 21:35:17	16	1	5
11	13310147298_10155749700497299		Atlanta family is one	link	http://on-ajc.com/2fi	2017-09-17 21:17:35	246	8	41
12	13310147298_10155749674402299		Mother could face c	link	http://on-ajc.com/2fs	2017-09-17 21:01:16	19	8	18
13	13310147298_1015	"The girls are essenti	Dress-wearing' studer	link	http://on-ajc.com/2f	2017-09-17 20:44:33	2	1	0
14	13310147298_1015	He said his "blood w	Nature photographer	link	http://on-ajc.com/2xl	2017-09-17 20:28:02	23	1	17
15	13310147298_1015	Pro tip: Always look t	A huge snake found	link	http://on-ajc.com/2f	2017-09-17 20:10:01	37	11	41
16	13310147298_1015	ESPN host Jemele H	Here's why the White	link	http://on-ajc.com/2xl	2017-09-17 19:52:19	22	17	0
17	13310147298_1015	A bride and groom to	Couple invites Obam	link	http://on-ajc.com/2xl	2017-09-17 19:37:01	157	11	5
18	13310147298_10155749524722299		Fixer Upper' couple	link	http://on-ajc.com/2fi	2017-09-17 19:21:18	34	24	4
19	13310147298_1015	Letting the grandpar	Study: Grandparents	link	http://on-ajc.com/2fi	2017-09-17 19:05:05	14	1	3

ANNEXURE II


NOTES
Users
id (PK)
username
created_at
Photos
id (PK)
image_url
user_id (FK)
created_at
Comments
id (PK)
comment_text
user_id (FK)
photo_id (FK)
created_at
Likes
user_id (PK)
photo_id (FK)
created_at
2 PKs
∵ unique user_id & photo_id needed
no id ∵ no need
// we are not going to reference id anywhere else
TOO COMPLEX!!
Follows
follower_id (FK)
followee_id (FK)
created_at
(follower followee followee)
FK = both
Tags
tag_name
id (PK)
created_at
Photo_Tags
photo_id
tag_id
FK)
(Need BCNF)!!
↳ make 2 tables

NODE.JS CODE

```
var express = require('express');
var app = express();
var mysql = require('mysql')
var bodyParser = require('body-parser');
const sortBy = require('sort-array')


app.set('port', (process.env.PORT || 5000));
app.set("view engine", "ejs");
app.use(bodyParser.urlencoded({extended: true}));
app.use(express.static(__dirname + '/public'));



var db_config = {
  host: 'us-cdbr-iron-east-05.cleardb.net',
  user: 'b09777f1d9817f',
  password: '********', //Password starred for security reasons
  database: 'heroku_77726c8f6b9784b'
};


var connection;

function handleDisconnect() {
  connection = mysql.createConnection(db_config); // Recreate the connection, since
                                                  // the old one cannot be reused.

  connection.connect(function(err) {              // The server is either down
    if(err) {                     // or restarting (takes a while sometimes).
      console.log('error when connecting to db:', err);
      setTimeout(handleDisconnect, 2000); // We introduce a delay before attempting
to reconnect,
    }              // to avoid a hot loop, and to allow our node script to
  });             // process asynchronous requests in the meantime.
                 // If you're also serving http, display a 503 error.
  connection.on('error', function(err) {
    console.log('db error', err);
    if(err.code === 'PROTOCOL_CONNECTION_LOST') { // Connection to the MySQL server
is usually
      handleDisconnect();       // lost due to either server restart, or a
    } else {                    // connnection idle timeout (the wait_timeout
      throw err;               // server variable configures this)
    }
  });
}

handleDisconnect();

var count2 = 252;
var count3 = 7216;
var count4 = 0;
//SELECT COUNT(*) AS count2 FROM photos

app.get("/", function(req, res){
  var q_users = "SELECT COUNT(*) AS count1 FROM users";
```

```
  connection.query(q_users, function(err,results, fields){
    if(err) throw err;
    var count1 = results[0].count1;
    //res.send(results);
    res.render("home", {data1: count1, data2: count2, data3: count3, data4: count4});
  });
});

app.post('/adduser', function(req,res){
  var person = { username: req.body.username};

  connection.query('INSERT INTO users SET ?', person, function(err, result){
    if(err) throw err;
    res.redirect("/");

  });
});


app.post('/deleteuserid', function(req,res){
  var id = req.body.userid;

  connection.query('DELETE FROM users WHERE id = ?', id, function(err, result){
    if(err) throw err;
    //res.send(result);
    res.redirect("/countpictures");

  });
});

app.get("/countpictures", function(req, res){

  connection.query('SELECT COUNT(*) AS count2 FROM photos', function(err, results){
    if(err) throw err;
    count2 = results[0].count2;
    res.redirect("/countcomments");
  });
});


app.get("/countcomments", function(req, res){

  connection.query('SELECT COUNT(*) AS count3 FROM comments', function(err, results){
    if(err) throw err;
    count3 = results[0].count3;
    res.redirect("/");
  });
});

app.post('/countpictures', function(req,res){


  connection.query('SELECT COUNT(*) AS count2 FROM photos', function(err, results){
    if(err) throw err;
    count2 = results[0].count2;
    res.redirect("/");

  });
```

```
});

app.post('/counttaggedpictures', function(req,res){

  var tag_id = req.body.tagid;

  connection.query('SELECT COUNT(*) AS count4 FROM photo_tags WHERE tag_id = ?',
tag_id, function(err, results){
    if(err) throw err;
    count4 = results[0].count4;
    res.redirect("/");

  });
});

app.post("/seealltags", function(req, res){
  var q_users = "SELECT id AS id, tag_name AS tagss FROM tags";

  connection.query(q_users, function(err,results, fields){
    if(err) throw err;
    var disp_results = sortBy(results, 'id');
    res.render("todo", {supplies: disp_results});

  });
});

app.post("/seeallusers", function(req, res){
  var q_users = "SELECT id AS id, username AS username FROM users";

  connection.query(q_users, function(err,results, fields){
    if(err) throw err;
    var disp_results = sortBy(results, 'id');
    res.render("allusers", {supplies: disp_results});

  });
});

app.post('/findpictures', function(req,res){

  var tag_id = req.body.allpictagid;

  connection.query('SELECT id, image_url AS pics FROM photos JOIN photo_tags ON
photos.id = photo_tags.photo_id WHERE tag_id = ?', tag_id, function(err, results){
    if(err) throw err;
    var disp_results = sortBy(results, 'id');
    res.render("pictures", {supplies: disp_results});

  });
});


app.post('/userpic', function(req,res){

  var tag_id = req.body.userpics;

  connection.query('SELECT id, image_url FROM photos WHERE user_id = ?', tag_id,
function(err, results){
    if(err) throw err;
    var disp_results = sortBy(results, 'id');
```

```
    res.render("userpics", {supplies: disp_results});

  });
});

app.listen(app.get('port'), function() {
  console.log('Node app is running on port', app.get('port'));
});
```

HTML CODE

```
<link href="https://fonts.googleapis.com/css?family=Roboto:100,300,400"
rel="stylesheet">
<link rel="stylesheet" href="/app.css">


<h1 style="color:#66B2FF"><em><b>#Tagged</b></em></h1>
<p class="lead" style="color:#66B2FF">(CSE 608: Database Systems Project)</p>
<br>
<br>
<br>

<p class="lead">We currently have <strong><%=data1%></strong>
users in the database, with a total of <strong><%=data2%></strong> pictures and
<strong><%=data3%></strong> comments! </p>
<br>
<br>
<br>
<br>

<form method="POST" action='/adduser'>
 <p class="lead">Add a new user</p>
 <input type="text" name = "username" class="form" placeholder="Enter the Username"
style="margin-left: 10px">
 <button style="margin-left: 15px">Enter a new User</button>
</form>
<br>
<br>

<p class="lead">See all users with their IDs</p>
<form method="POST" action='/seeallusers'>
 <button type="submit" formaction"/seeallusers" style="margin-left: 40px; margin-top:
10px">See all users</button>
</form>
<br>
<br>


<form method="POST" action='/userpic'>
 <p class="lead">See all pictures by a User</p>
 <input type="text" name = "userpics" class="form" placeholder="Enter the User ID"
style="margin-left: 10px">
 <button style="margin-left: 15px">Enter the User ID</button>
</form>
<br>
<br>

<form method="POST" action='/deleteuserid'>
 <p class="lead">Delete a User</p>
 <p class="lead">(If deleted, the entire record with all the pictures, likes,
comments and tags of the user will be deleted!)</p>
 <input type="text" name = "userid" class="form" placeholder="Enter the ID"
style="margin-left: 10px">
 <button style="margin-left: 15px">Delete User</button>
</form>
<br>
<br>
```

```
<form method="POST" action='/countpictures'>
 <p class="lead">Count total pictures in the database</p>
 <button type="submit" formaction"/countpictures" style="margin-left: 40px; margin-
top: 10px">Count total pictures</button>
</form>
<br>
<br>

<form method="POST" action='/seealltags'>
 <p class="lead">Find the number of pictures tagged with a common tag</p>
 <p class="lead">(See all tags with their Tag IDs)</p>
 <button type="submit" formaction"/seealltags" style="margin-left: 40px; margin-top:
10px">See all available Tags</button>
</form>

<form method="POST" action='/counttaggedpictures'>
 <input type="text" name = "tagid" class="form" placeholder="Enter the tag ID"
style="margin-left: 10px">
 <button style="margin-left: 15px">Count Pictures</button>
 <p class="lead">Currently <strong><%=data4%></strong> pictures are tagged with this
tag.</p>
</form>
<br>
<br>

<form method="POST" action='/findpictures'>
 <p class="lead">Find the pictures having the entered tag</p>
 <input type="text" name = "allpictagid" class="form" placeholder="Enter the tag ID"
style="margin-left: 10px">
 <button style="margin-left: 15px">Find Pictures</button>
</form>
```

SQL SCHEMA CREATION CODE

(Created and inserted directly in the database through MySQL workbench)

```
CREATE TABLE users (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE photos (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    image_url VARCHAR(255) NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id)
);

CREATE TABLE comments (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    comment_text VARCHAR(255) NOT NULL,
    photo_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    FOREIGN KEY(user_id) REFERENCES users(id)
);

CREATE TABLE likes (
    user_id INTEGER NOT NULL,
    photo_id INTEGER NOT NULL,
    created_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 tags (
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  tag_name VARCHAR(255) UNIQUE,
  created_at TIMESTAMP DEFAULT NOW()
);

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

DATA INSERTION CODE (SQL QUERIES IN NODE.JS)

```
var mysql = require('mysql');
var faker = require('faker');


var connection = mysql.createConnection({
  host     : ' us-cdbr-iron-east-05.cleardb.net',
  user     : ' b09777f1d9817f ',
  database : 'heroku_77726c8f6b9784b',
  password: '********', //Password starred for security reasons

});

//Inserting Users

var data = [];
for(var i = 0; i < 100; i++){
    data.push([
        faker.name.firstName + '_' + faker.name.lastName
        faker.date.past()
    ]);
}

var q = 'INSERT INTO users (username, created_at) VALUES ?';

connection.query(q, [data], function(err, result) {
  console.log(err);
  console.log(result);
});

// Inserting Images

var data = [];
for(var i = 0; i < 280; i++){
    data.push([
        faker.image.imageUrl,
        faker.date.past()
    ]);
}

var q = 'INSERT INTO photos (image_url, created_at) VALUES ?';

connection.query(q, [data], function(err, result) {
  console.log(err);
  console.log(result);
});

//Inserting into comments
var data = [];
for(var i = 0; i < 7500; i++){
    data.push([
        faker.lorem.words,
        faker.random.uuid[1,100],
        faker.random.uuid[1,280],
        faker.date.past()
    ]);
```

```
}

var q = 'INSERT INTO comments (comment_text, user_id, photo_id, created_at) VALUES
?';

connection.query(q, [data], function(err, result) {
  console.log(err);
  console.log(result);
});
connection.end();
```