Ariel Uribe, John Kim
CS 460 final report
March 5, 2023
Prof. Kollios

SCHEMA:

```
drop database if exists photoshare;
create database if not exists photoshare;
use photoshare;
drop table if exists registeredUser cascade;
drop table if exists albums cascade;
drop table if exists photo_in_album cascade;
drop table if exists comments cascade;
drop table if exists tags cascade;

create table registeredUser(
userID int not null auto_increment,
fName varchar(20) not null,
lName varchar(20) not null,
fullName varchar(40) not null,
email varchar(255) UNIQUE,
password varchar(255),
gender char(6),
DOB DATE,
hometown varchar(20),
contributionScore int default 0,
primary key(userID)
);

create table albums(
albumName char(20),
DOC datetime default current_timestamp,
albumID int not null auto_increment,
ownerID int,
primary key (albumID),
foreign key (ownerID) references registeredUser(userID)
);

create table friendship(
userID int,
friendID int,
check(userID <> friendID),
primary key(userID, friendID),
foreign key(userID) references registeredUser(userID),
foreign key(friendID) references registeredUser(userID)
);

 create table photo_in_album(
 pID int auto_increment not null,
 albumID int,
 caption text,
 photoBinary longblob,
 userID int,
 likes INTEGER default 0,
 primary key(pID),
 foreign key (userID) references registeredUser(userID),
  foreign key (albumID) references albums(albumID) on delete Cascade
 );

 create table comments(
 commentID int not null auto_increment,
 contents text not null,
 commentOwner varchar(40),
 commentDate datetime default current_timestamp,
 primary key(commentID)
 );

 create table comment_under_photo(
 commentID int,
 pID int ,
 primary key(pID, commentID),
 foreign key (pID) references photo_in_album(pID) on delete cascade,
 foreign key (commentID) references comments(commentID)
 );

 create table tags(
tagTitle varchar(20) not null,
primary key(tagTitle)
 );
 create table hasTag(
pID int,
tagTitle varchar(20) not null,
primary key (pID, tagTitle),
foreign key (pID) references photo_in_album(pID) on delete cascade,
foreign key (tagTitle) references tags(tagTitle)
 );

 create table likesPhoto(
pID int,
userID int,
primary key(pID, userID),
foreign key(pID) references photo_in_album(pID) on delete cascade,
foreign key(userID) references registeredUser(userID)
);
```

ADDITIONAL COMMENTS:

- implemented a comment constraint trigger in python rather than the SQL to prevent the user from commenting under their own photos
- updated the data types of a lot of the tables to match the given in the PA1 answer key; i.e. auto increment, etc.
- added a check constraint to prevent users from being friends with themselves
- added user_id attribute in the photos
- updated the relationship between users and albums
- removed our userLeftComment table bc it's repetitive and unnecessary

USE CASES:

- **BECOMING A REGISTERED USER:** implemented fully with no assumptions
    - takes in all inputs from form requests in the html and then inserts it into the table for registered users
- **ADDING AND LISTING FRIENDS:** assumed users would not double friend someone, prevented users from friending themselves, assumed that the user would have the email of the friend that they are searching for, simply listed out the full name of all friends, assumed 1 way activity for friends (to be mutual, they would add each other)

- adding friends: button hyper references to an add_friend function inside python file which then requests the userID and friend ID and inserts them if they arent friends already using helper functions
- finding friends: hyperreferences button refers to an find_friend function which finds the user id of the friend that the user inputs using their email and the function returns their id in the html
- listing friends: button hyper ref to a show_friend function that takes in their userID and uses helper functions to find their associated friends in the sql and returns all their info to be displayed

- **USER ACTIVITY:** implemented fully with no assumptions
  - simply increments the contribution score inside the uploading photos, and adding comments functions and displays the top 3 users with the highest scores
- **PHOTO AND ALBUM BROWSING:** implemented fully with no assumptions
  - routes to the html page which allows users to select what photos and albums they can search through and view by selection queries and display htmlsx
- **PHOTO AND ALBUM CREATING:** implemented fully
  - Routed to the html page like usual. User needs to be signed in to access.
  - Takes input album name from html page and stores in the database through a query in python when the post method is used.
- **VIEWING PHOTOS BY TAG:** implemented fully
  - displays hyperlinks to each tag which then passes the tag to a helper function that displays the photos by querying for its metadata based on input tag
- **VIEWING ALL PHOTOS BY TAG:** implemented fully with no assumptions
  - hyper refs to app route all tagged photos to get the respective tags and then tags are passed to another python function that find the photo and displays it for them
- **VIEWING MOST POPULAR TAGS:** implemented fully with no assumptions
  - same thing except the sql query selects from all tags the 3 most popular to be displayed with their respective photos
- **PHOTO SEARCH:could not get the multiple tag search down; single tag search works**
  - takes in a search box input which the input is then broken down into individual tags if multiple and passed to their respective htmls, 1 for multiple 1 for single tag search and it displays using the displayer function in python
- **LEAVING COMMENTS:** assumed users would leave a comment to view other comments, displayed their id rather than other identifiers, added python constraint to make sure the user would not comment on their own photos
  - gets pID, userID, commentID form form requesting on the html and then inserts and creates on the respective tables to be displayed later in a nother html template with a displayer function
- **LIKE:** assumed that users would not double like a photo

- same thing, but the likes counter is incremented in its respective table which is later displayed in another html template
- **SEARCH ON COMMENTS:** implemented fully with no assumptions
  - takes in a search box input which then passes in the comment contents into a helper html and function which displays all of the comments that match the contents through query
- **FRIEND RECOMMENDATION:** assumed that we shouldnt return the friends or the user itself, assumed that we should return their ids, assumed user should be signed in to get recommendations because of the nature of friends
  - uses get_friend helper to find friends then their friend of friends which returns a sorted output of friends ordered by appearance in highest to lowest frequency
- **YOU-MAY-ALSO-LIKE:** implemented fully
  - queries for the top tags and searches for the respective photos of each category that best match it using our helper functions

all implemented fully minus the photo search multiple tag search