Section slides: <http://webdev.slides.com/coltsteele/mysql-106#/19>

* In this section we’ll be inserting a ton of data into our Instagram database schema, then asking some questions that we’ll need to answer using the techniques we’ve learned in this class

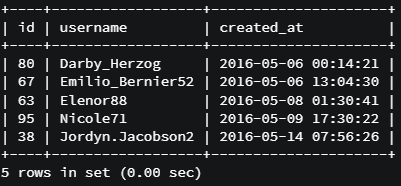
# Inserting the Dataset

* For this section we’ll use the ig\_clone\_data.sql file that the instructor has created
* It consists of thousands of likes, 21 different hashtags, a couple hundred photos, and 100 users

# Instagram Clone Challenge 1 – The 5 Most-Tenured Users

* Suppose we want to reward our users who have been around the longest. **Find the 5 oldest users**.
* This can be accomplished simply be listing out all the users, sorting them by the **created\_at** timestamp, and limiting the output to 5





* Solution code

SELECT \*

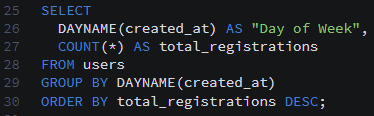
FROM users

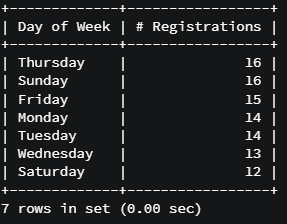
ORDER BY created\_at

LIMIT 5;

# Instagram Clone Challenge 2 – The most popular day of the week to register on

* For this exercise, we want to schedule an ad campaign that will hit the greatest number of new users. Therefore, we want to determine the most popular day of the week that people register on, so we know what day of the week to run on ad on.
* We can determine this by looking at the *users* table and using string functions to determine what day of the week they registered on. The relevant function is DAYNAME(), which returns the day of the week given a DATE, DATETIME, or TIMESTAMP. We’ll also use GROUP BY to group by the name of the day, and the COUNT() function to count how many times that name of the day occurred (and thus the number of registrations on that day).
* We will then ORDER BY that count to determine which day is most popular





* Thus, it appears that Thursday and Sunday are tied for the most popular day of the week to register on.
* Instructor solution code:

SELECT

DAYNAME(created\_at) AS day,

COUNT(\*) AS total

FROM users

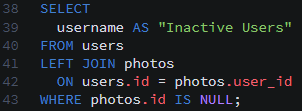
GROUP BY day

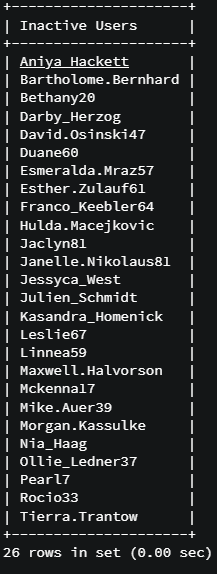
ORDER BY total DESC

LIMIT 2;

# Instagram Clone Challenge 3 - Users who have never posted a photo

* We want to send an email campaign out to our inactive users who have never posted a photo
  + Note that a user can be active on Instagram without ever posting photos. They can be “lookers” instead
* We’ll need to work with the *users* and *photos* table here. We’ll need to create a LEF JOIN that lists all users, even those that do not have any photos. Those users who have no photos will have NULL values on the join. We can use this property to select only for those users





* Instructor solution

SELECT username

FROM users

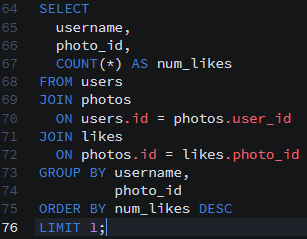
LEFT JOIN photos

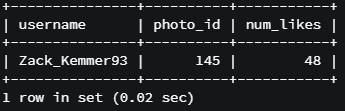
ON users.id = photos.user\_id

WHERE photos.id IS NULL;

# Instagram Clone Challenge 4 – The Most Popular Photo

* In this challenge, we want to find the user who has the most-liked photo. That person wins a prize!
* This will again require a join, this time using the *users*, *photos*, and *likes* tables. In the approach here, we join all three tables and then find the user-photo\_id combo that appears most often.





* Another approach could have been to find the photo with the most likes first (just group the *likes* table by photo\_id), then use that photo\_id and the *photos* table to find the corresponding user
* Instructor solution

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

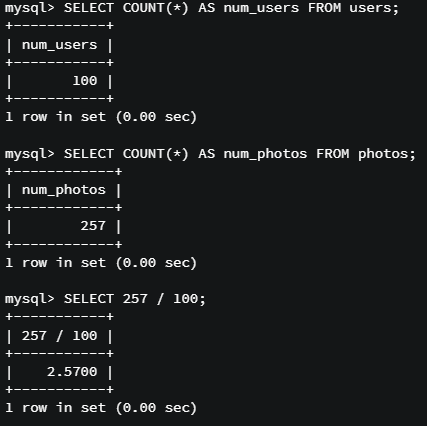
GROUP BY photos.id

ORDER BY total DESC

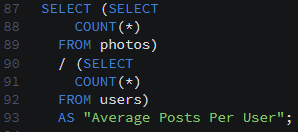
LIMIT 1;

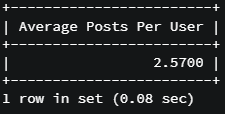
# Instagram Clone Challenge 5 – How many times does the average user post?

* To find this, we simply divide the total number of photos by the total number of users.
* One way to do this with separate lines:



* Another approach is to use subqueries, which is a bit faster and can be done with one line:





* Looks like there are about 2.57 posts per user. But this is skewed significantly because
* Instructor solution code

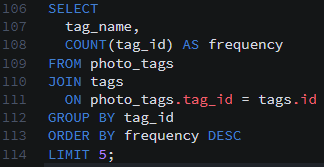
SELECT (SELECT Count(\*)

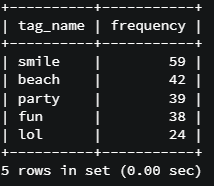
FROM photos) / (SELECT Count(\*)

FROM users) AS avg;

# Instagram Clone Challenge 6 – Determine the top 5 most commonly used hashtags

* This should be simple enough. Remember that we have a table of *tags* and a table of *photo\_tags*. So what we’ll do is to first perform a GROUP BY within photo\_tags and count the number of times each tag\_id appears. Then we’ll JOIN with the *tags* table to determine the actual text of those tags. We’ll sort them and determine to top 5 most-used hashtags



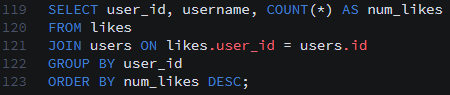


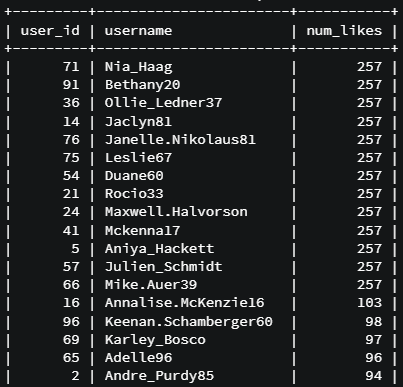
* Instructor solution code

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;

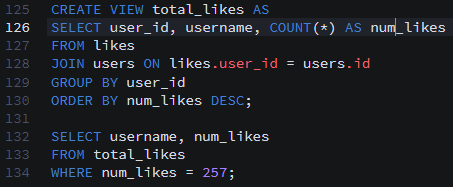
# Instagram Challenge 7 – The Bot Problem

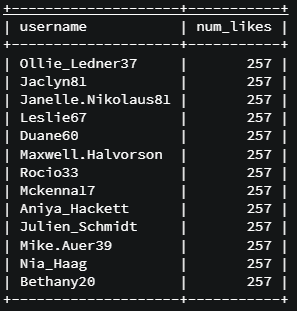
* In this challenge, we need to find all users who have liked EVERY SINGLE PHOTO on in our database. These are most likely bots
* To solve this, we’ll make use of a few key things:
  + We know exactly how many photos are in our database: 257
  + We also know that a user can only like a given photo one time. Thus, users who like every photo will have 257 likes
* With that in mind, all we need to do is perform a GROUP BY on user\_id within the *likes* table, then COUNT the number of instances for each user, with each instance being of like from that user. We can sort the table by number of likes, and then observe which users have 257 likes.



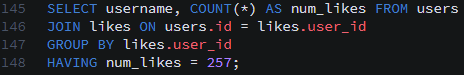


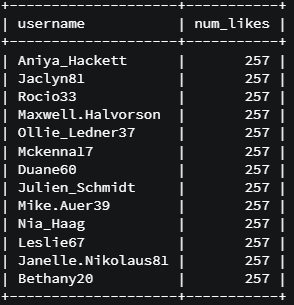
* What if we wanted to instead directly isolate these bots using a WHERE statement? We can’t do that in our approach above, because WHERE clauses come before GROUP BY. In this case though, we want to filter our data that has already been grouped. One way to do this is to create a VIEW from the table above. From there, we can make a SELECT using the WHERE clause to only select for users with 257 likes.
  + A VIEW is like a table, but it’s not in memory. It’s just in code. And you can manipulate this view without touching the original tables at all



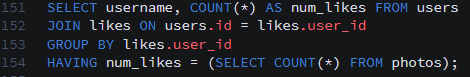


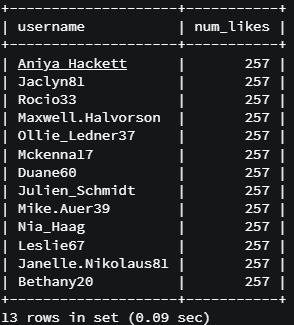
* A third approach is to use the **HAVING** clause (we didn’t learn this yet). HAVING acts like WHERE and allows us to filter our grouped/aggregated data. This allows us to perform our filtering in one SQL command.
  + Documentation on HAVING: https://www.w3schools.com/sql/sql\_having.asp





* + However, we still have the issue of having hard-coded 257. This wouldn’t work if we don’t know how many photos we have in our database. To fix this and make it more dynamic, we just use a subquery where we COUNT the number of photos in the *photos* table. Simple! This will allow us to filter for users who have likes every photo, irrespective of how many photos we actually have in our database.





* Instructor solution

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);