

# **Instagram User Analytics**

## **Project Description:**

This project uses MySQL and SQL queries to analyse Instagram's database, providing marketing insights, user engagement metrics, bot detection, and guidance for ad and investor strategies—all processed in MySQL Workbench for decision-ready reporting.

## **Project Approach:**

The project was executed using SQL, where queries were utilized to create a database from the raw data provided. Sorting and data-executing queries were then implemented to obtain the required data and insights.

## **Tech Stack Used:**

Intel Core i5

Windows 10

MySQL 8.0 CE Workbench

## **Project Insights:**

**CREATE DATABASE ig\_clone;**

**USE ig\_clone;**

**USE database** command is run to select the database that we work on.

## **Creating a table:**

Creating a table named users to save information related to users of instagram app.

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

**CREATE TABLE** command is used to create the table.

We create three columns in our table, namely

1) Id

```
id INT AUTO_INCREMENT UNIQUE PRIMARY KEY,
```

The query receives a unique integer value as input. We define the ID column as the primary key to reference other foreign key tables.

Primary key values are unique and distinct.

2) username

```
username VARCHAR(255) NOT NULL,
```

The query receives a string as input. NOT NULL specifies the input value can't be null.

3) created\_at

```
created_at TIMESTAMP DEFAULT NOW()
```

This column specifies the date and time on which the user id was created.

Similarly, we create our other tables for

Photos uploaded on the app

```
CREATE TABLE photos(
```

```
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    image_url VARCHAR(355) NOT NULL,
```

```
    user_id INT NOT NULL,
```

```
    created_at TIMESTAMP DEFAULT NOW(),
```

```
    FOREIGN KEY(user_id) REFERENCES users(id)  
);
```

**Foreign Key** is used to reference the primary key which in our case is the id column.

Database for all comments on the app

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

For likes

```
CREATE TABLE likes(  
    user_id INT NOT NULL,  
    photo_id INT 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)  
);
```

For follows

```
CREATE TABLE follows(  
    ...  
);
```

```

follower_id INT NOT NULL,
followee_id INT NOT NULL,
created_at TIMESTAMP DEFAULT NOW(),
FOREIGN KEY (follower_id) REFERENCES users(id),
FOREIGN KEY (followee_id) REFERENCES users(id),
PRIMARY KEY(follower_id,followee_id)
);

```

For hashtags

```

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

```

Then I have create photo\_tags table as junction table

```

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

```

## **Inserting values in table columns:**

Inserting values in table/columns we have created, we use

**INSERT INTO tablename (column1, column2,..column n) VALUES**

**(expression1,expression2,.....expression n);**

where tablename is the table we want to add values to, column refers to the columns in the table and expressions are the values that we want to add.

## Result:

To visualize the data in tables we use query

**SELECT \* FROM tablename;**

To visualize data from users table we use command

**select \* from users :**

we get output as

```
73
74 • INSERT INTO photo_tags(photo_id, tag_id) VALUES (1, 18), (1, 17), (1, 21), (1, 13), (1, 19), (2, 4), (2, 3), (2, 20), (2, 2), (3, 8), (4, 12), (4, 11), (4, 21), (4, 13), (5, 15), (5, 14), (5, 17)
75
76
77     OUTPUTS
78
79 • USE ig_clone;
80
81 • SELECT *
82     FROM users;
83
84     ## A) Marketing Analysis
85     # 1. Loyal User Reward: 5 Oldest Users
86
87 • SELECT *
88     FROM users
89     ORDER BY created_at ASC
90     LIMIT 5;
```

| id | username            | created_at          |
|----|---------------------|---------------------|
| 1  | Kenton_John         | 2017-02-16 18:22:11 |
| 2  | Aleksa_Pundt5       | 2017-02-16 18:22:11 |
| 3  | Yolka_Wendy3        | 2017-02-21 11:12:23 |
| 4  | Arelis_Bogart63     | 2016-08-13 01:28:43 |
| 5  | Arvia_Hackett       | 2016-12-07 01:04:39 |
| 6  | Travon_Walter       | 2017-04-30 13:26:14 |
| 7  | Kassandra_Homnick   | 2016-12-12 06:30:08 |
| 8  | Shawn_Schamberger11 | 2016-09-19 10:45:55 |
| 9  | Gurni3              | 2016-06-24 19:36:11 |
| 10 | Predney_McClure     | 2016-08-07 16:25:49 |
| 11 | Justine_Gaynor27    | 2017-05-04 16:32:16 |
| 12 | Derekd5             | 2017-01-19 01:04:14 |
| 13 | Alexandro35         | 2017-03-29 17:09:02 |
| 14 | JefreiR1            | 2017-03-06 21:26:18 |

Output

| Action | Time     | Action  | Message              | Duration / Fetch      |
|--------|----------|---|----------------------|-----------------------|
| 1      | 17:15:58 | USE ig_clone  | 0 row(s) affected    | 0.032 sec             |
| 2      | 17:33:58 | SELECT * FROM users ORDER BY created_at ASC LIMIT 5 | 5 row(s) returned    | 2.562 sec / 0.000 sec |
| 3      | 17:36:38 | SELECT * FROM users LIMIT 0, 1000                   | 1000 row(s) returned | 0.000 sec / 0.000 sec |

# SQL Tasks and Queries:

## A. Marketing Analysis

### 1. Loyal User Reward: 5 Oldest Users

**Objective** The marketing team wants to reward the most loyal users, i.e., those who have been using the platform for the longest time.

**Objective:** Identify the five oldest users on Instagram from the provided database.

#### SQL Code:

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

#### Explanation:

We select columns username and created\_at from users table and order the entries from created\_at in descending order. We limit the number of rows to 5.

#### Output:

The screenshot shows the SQL Server Management Studio interface with two panes. The top pane displays the SQL script for the 'Marketing Analysis' task, specifically the query to find the 5 oldest users. The bottom pane shows the results grid with 5 rows of user data and the execution plan and statistics at the bottom.

| ID | username         | created_at          |
|----|------------------|---------------------|
| 80 | Darby.Herzog     | 2016-05-09 00:14:21 |
| 67 | Emilio.Banner52  | 2016-05-09 13:04:30 |
| 63 | Eleanor88        | 2016-05-08 01:30:41 |
| 95 | Nicole71         | 2016-05-09 17:30:22 |
| 38 | Jordyn.Jacobson2 | 2016-05-14 07:36:26 |

Action Output

| # | Time     | Action  | Message             | Duration / Fetch      |
|---|----------|---|---------------------|-----------------------|
| 1 | 17:15:58 | USE Ig_clone  | 0 row(s) affected   | 0.032 sec             |
| 2 | 17:33:58 | SELECT * FROM users ORDER BY created_at ASC LIMIT 5 | 5 row(s) returned   | 2.562 sec / 0.000 sec |
| 3 | 17:36:38 | SELECT * FROM users LIMIT 0, 1000                   | 100 row(s) returned | 0.000 sec / 0.000 sec |
| 4 | 17:41:07 | SELECT * FROM users ORDER BY created_at ASC LIMIT 5 | 5 row(s) returned   | 0.000 sec / 0.000 sec |

### **Conclusion:**

Justina.Gaylord27, Travon.Waters, Milford\_Gliechner42, Hailee26, Maxwell.Halvorson are the first five users of instagram app.

## **2. Inactive User Engagement: Users Who Never Posted a Photo**

The team wants to encourage inactive users to start posting by sending them promotional emails.

**Objective:** Identify users who have never posted a photo to target with re-engagement campaigns.

### **SQL Code:**

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

### **Explanation:**

Here we use **JOINT** clauses to join photos on users. To find records where users id having null join records in photos are given as output.

The **WHERE** clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

### **LEFT JOIN**

The LEFT JOIN command returns all rows from the left table, and the matching rows from the right table.

## Output:

The screenshot shows the SQL Server Management Studio interface with three queries run against the Instagram User Analytics database. The first query retrieves the 5 oldest users. The second query finds users who have never posted a photo. The third query identifies the contest winner based on the most likes on a single photo.

```
85  # 1. Loyal User Reward: 5 Oldest Users
86
87 • SELECT *
88   FROM users
89   ORDER BY created_at ASC
90   LIMIT 5
91
92  # 2. Inactive User Engagement: Users Who Never Posted a Photo.
93
94 • SELECT username
95   FROM users
96   LEFT JOIN photos
97     ON users.id = photos.user_id
98   WHERE photos.id IS NULL;
99
100 # 3. Contest Winner: Most Likes On a Single Photo
101
< 102 • SELECT username, photos.id, photos.image_url, COUNT(likes.user_id) AS total>
```

| username          | photos.id | photos.image_url | COUNT(likes.user_id) |
|-------------------|-----------|------------------|----------------------|
| Aniva_Hodnett     |           |                  | 0                    |
| Keesha_Homnick    |           |                  | 0                    |
| Jadyn81           |           |                  | 0                    |
| Rocio33           |           |                  | 0                    |
| Maxwell_Halvorson |           |                  | 0                    |
| Terry_Trautwol    |           |                  | 0                    |
| Pearl7            |           |                  | 0                    |
| Ollie_Jedder37    |           |                  | 0                    |
| Makenna17         |           |                  | 0                    |
| David_Osredrik7   |           |                  | 0                    |
| Morgan_Kasule     |           |                  | 0                    |
| Liane59           |           |                  | 0                    |
| Duanell           |           |                  | 0                    |
| Adam_Schmidt      |           |                  | 0                    |

Action Output

| # | Time     | Action   | Message             | Duration / File |
|---|----------|--|---------------------|-----------------|
| 1 | 17:15:58 | USE [dbo]  | 0 row(s) affected   | 0.032 sec       |
| 2 | 17:23:58 | SELECT * FROM users ORDER BY created_at ASC LIMIT 5  | 5 row(s) returned   | 2.562 sec / 0   |
| 3 | 17:26:38 | SELECT * FROM users LIMIT 0, 1000  | 100 row(s) returned | 0.000 sec / 0   |
| 4 | 17:41:07 | SELECT * FROM users ORDER BY created_at ASC LIMIT 5  | 5 row(s) returned   | 0.000 sec / 0   |
| 5 | 17:43:49 | SELECT username FROM users LEFT JOIN photos ON users.id = photos.user_id WHERE photos.id IS NULL LIMIT 0, 1000 | 26 row(s) returned  | 0.141 sec / 0   |

## Conclusion:

The result table displays the usernames of IDs who have never uploaded a single photo on Instagram.

## 3. Contest Winner: Most Likes On a Single Photo

The team wants to encourage inactive users to start posting by sending them promotional emails.

**Objective:** Determine the photo with the most likes to identify the contest winner.

### SQL Code:

```
SELECT username, photos.id, photos.image_url, COUNT(likes.user_id) 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 ;

```

### Explanation:

SELECT command is used to show selected columns.

COUNT(\*) is used to count the number of entries in the photos table.

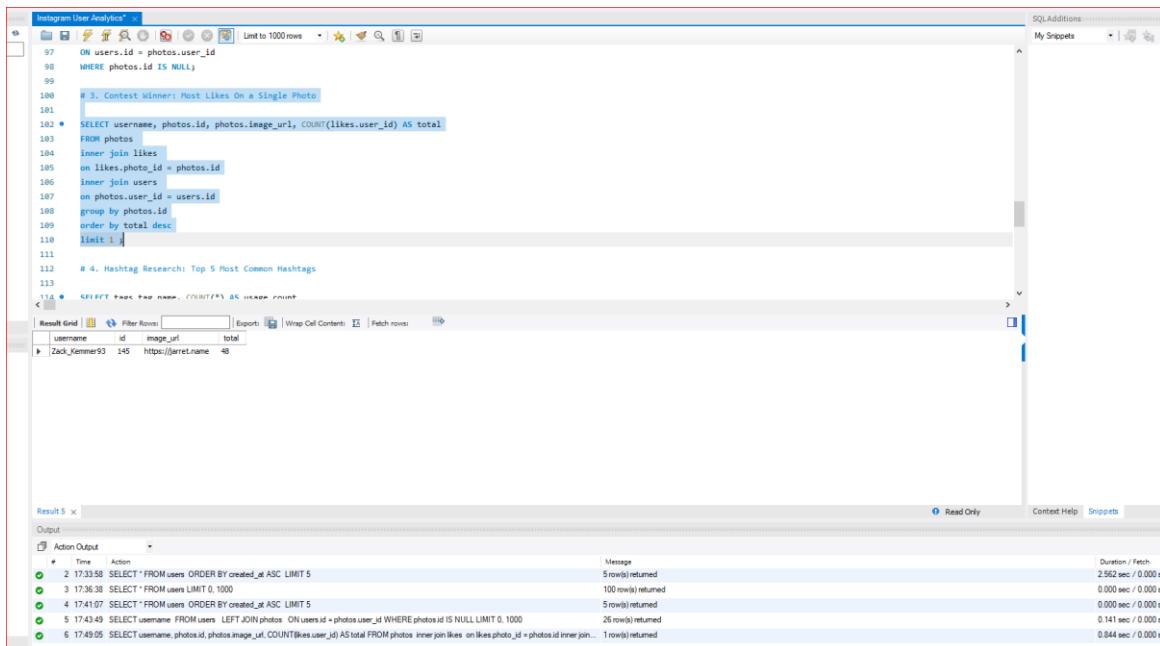
### INNER JOIN

The INNER JOIN clause in SQL is used to combine multiple tables and fetch records that have the same values in the common columns.

First inner join is applied as a likes on likes (photos\_id) column. Another is applied as users on photos(user\_id) column.

The result is given in descending order using GROUP BY and DESC. Entries are limited to 1

### Output:



The screenshot shows the SQLAdditions interface with the following details:

- SQL Editor:** Contains the following SQL code:

```

97 ON users.id = photos.user_id
98 WHERE photos.id IS NULL;
99
100 # 3. Contest Winner: Most Likes On a Single Photo
101
102 • SELECT username, photos.id, photos.image_url, COUNT(likes.user_id) AS total
103   FROM photos
104   INNER JOIN likes
105     ON likes.photo_id = photos.id
106   INNER JOIN users
107     ON photos.user_id = users.id
108   GROUP BY photos.id
109   ORDER BY total DESC
110   LIMIT 1 ;
111
112 # 4. Hashtag Research: Top 5 Host Common Hashtags
113
114 SELECT hashtags.name, COUNT(*) AS hashtag_count
    
```
- Result Grid:** Displays the output of the query:

| username      | id  | image_url          | total |
|---------------|-----|--------------------|-------|
| Zack_Kemmer93 | 145 | https://arret.name | 48    |
- Action Output:** Shows the execution log with the following rows:

| Time       | Action   | Message             | Duration / Fetch      |
|------------|--|---------------------|-----------------------|
| 2 17:33:58 | SELECT * FROM users ORDER BY created_at ASC LIMIT 5  | 5 row(s) returned   | 2.562 sec / 0.000 sec |
| 3 17:36:38 | SELECT * FROM users LIMIT 0,1000   | 100 row(s) returned | 0.000 sec / 0.000 sec |
| 4 17:41:07 | SELECT * FROM users ORDER BY created_at ASC LIMIT 5  | 5 row(s) returned   | 0.000 sec / 0.000 sec |
| 5 17:43:49 | SELECT username FROM users LEFT JOIN photos ON users.id = photos.user_id WHERE photos.id IS NULL LIMIT 0,1000  | 26 row(s) returned  | 0.141 sec / 0.000 sec |
| 6 17:49:05 | SELECT username, photos.id, photos.image_url, COUNT(likes.user_id) AS total FROM photos INNER JOIN likes ON likes.photo_id = photos.id INNER JOIN users ON users.id = photos.user_id ORDER BY total DESC LIMIT 1 | 1 row(s) returned   | 0.844 sec / 0.000 sec |

## Conclusion:

User Zack\_Kemmer93 is the winner of the contest with 48 likes which is most number of likes on a single post.

## 4. Hashtag Research:

A partner brand wants to know the most popular hashtags to use in their posts to reach the most people.

**Objective:** Identify the top 5 most used hashtags to guide marketing campaigns.

### SQL Code:

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

### Explanation:

In COUNT(tag\_name) AS most\_popular query, AS is used as an alias to give name to created aliases.

### Output:

The screenshot shows the SQL Server Management Studio interface. The top pane displays the SQL script with comments and numbered steps. The bottom pane shows the results grid and the execution plan details.

**Result Grid:**

| tag_name | usage_count |
|----------|-------------|
| smile    | 59          |
| beach    | 42          |
| party    | 42          |
| fun      | 38          |
| concert  | 24          |

**Action Output:**

| Time | Action  | Message            | Duration / Feed |
|------|---|--------------------|-----------------|
| 4    | 17:41:07 SELECT * FROM users ORDER BY created_at ASC, LIMIT 5   | 5 row(s) returned  | 0.000 sec / 0.0 |
| 5    | 17:43:49 SELECT username FROM users LEFT JOIN photos ON users.id = photos.user_id WHERE photos.id IS NULL LIMIT 0, 1000   | 26 row(s) returned | 0.141 sec / 0.0 |
| 6    | 17:49:05 SELECT username, photos.id, photos.image_url, COUNT(users.id) AS total FROM photos INNER JOIN users ON users.id = photos.user_id WHERE photos.id IS NOT NULL LIMIT 0, 1000 | 1 row(s) returned  | 0.844 sec / 0.0 |
| 7    | 17:54:04 SELECT username FROM users LEFT JOIN photos ON users.id = photos.user_id WHERE photos.id IS NULL LIMIT 0, 1000   | 26 row(s) returned | 0.015 sec / 0.0 |
| 8    | 17:59:59 SELECT tag.tag_name, COUNT(*) AS usage_count FROM photo_tags JOIN tags ON photo_tags.tag_id = tags.id GROUP BY tag.id ORDER BY usage_count DESC                            | 5 row(s) returned  | 1.594 sec / 0.0 |

## Conclusion:

Smile, Beach, Party, Fun, Concert are the most popular hashtags used by users on instagram in a descending order.

## 5. Ad Campaign Launch: Most Popular Registration Day

A partner brand wants to know the most popular hashtags to use in their posts to reach the most people.

**Objective:** Determine which day of the week has the most user registrations to optimize ad campaign timing.

### SQL Code:

```
SELECT DAYNAME(created_at) AS day, COUNT(*) AS total
FROM users
GROUP BY day
ORDER BY total DESC
LIMIT 1;
```

### Explanation:

The DATE\_FORMAT() function formats a date as specified.

%W is a parameter used to give weekday names in full (Sunday to Saturday).

### Output:

The screenshot shows a SQL editor interface with the following details:

- Query Editor:** Displays the SQL code for finding the most popular registration day. It includes comments (# 5. Ad Campaign Launch: Most Popular Registration Day) and a section for investor metrics.
- Result Grid:** Shows the output of the query. The columns are "day" and "total". The result is: Thursday 16.
- Action Output:** Shows the execution log with 9 entries, each detailing a query and its duration (e.g., 0.141 sec / 0.00). The log includes:
  - 17:43:49 SELECT username FROM users LEFT JOIN photos ON users.id = photos.user\_id WHERE photos.id IS NULL LIMIT 0, 1000
  - 17:49:05 SELECT username, photos.id, photo.image\_url, COUNT(likes.user\_id) AS total FROM photos INNER JOIN likes ON likes.photo\_id = photos.id INNER JOIN users ON users.id = photos.user\_id WHERE photos.id IS NULL LIMIT 0, 1000
  - 17:54:04 SELECT username FROM users LEFT JOIN photos ON users.id = photos.user\_id WHERE photos.id IS NULL LIMIT 0, 1000
  - 17:59:59 SELECT tag.tag\_name, COUNT(\*) AS usage\_count FROM photo\_tags JOIN tag ON photo\_tags.tag\_id = tag.id GROUP BY tag.id ORDER BY usage\_count DESC LIMIT 1
  - 18:17:24 SELECT DAYNAME(created\_at) AS day, COUNT(\*) AS total FROM users GROUP BY day ORDER BY total DESC LIMIT 1

## Conclusion:

16 registrations which is the most out of all weeks has been done on a thursday. Hence, the ad campaign should be scheduled on a thursday.

## B. Investor Metrics

### 1. User Engagement: Average Posts Per User

Investors want to know if users are still active and posting on Instagram or if they are making fewer posts.

**Objective:** Calculate the average number of posts per user to assess platform engagement.

#### SQL Code:

```
SELECT
(SELECT COUNT() FROM photos) / (SELECT COUNT() FROM users) AS Average.
```

#### Explanation:

Nested query has been used where an independent query is nested inside a dependent query.

Execution of outer query is dependent on inner query. / is used as a mathematical operator to perform division.

#### Output:

| Time        | Action   | Message            | Duration / Fetch      |
|-------------|--|--------------------|-----------------------|
| 6 17:49:05  | SELECT username, photo_id, photo.image_url, COUNT(bree.user_id) AS total FROM photos INNER JOIN likes ON likes.photo_id = photo.id INNER JOIN users AS bree ON users.id = bree.user_id | 1 row(s) returned  | 0.84 sec / 0.000 sec  |
| 7 17:50:04  | SELECT username FROM users LEFT JOIN photo ON users.id = photo.user_id WHERE photo.id IS NULL LIMIT 0, 1000  | 26 row(s) returned | 0.015 sec / 0.000 sec |
| 8 17:50:59  | SELECT tag_name, COUNT(*) AS usage_count FROM photo_tags JOIN tags ON photo_tags.tag_id = tags.id GROUP BY tag_id ORDER BY usage_count DESC LIMIT 0, 10                                | 5 row(s) returned  | 1.594 sec / 0.000 sec |
| 9 18:17:24  | SELECT DAYNAME(created_at) AS day, COUNT(*) AS total FROM users GROUP BY day ORDER BY total DESC LIMIT 0, 1000   | 1 row(s) returned  | 0.782 sec / 0.000 sec |
| 10 18:20:14 | SELECT (SELECT COUNT(*) FROM photos) / (SELECT COUNT(*) FROM users) AS Average LIMIT 0, 1000   | 1 row(s) returned  | 0.125 sec / 0.000 sec |

**Conclusion:**

Total number of photos divided by the total number of users on instagram is 2.5700.

## 2. Bots and Fake Accounts: Users Who Like Every Photo

Investors want to know if the platform is crowded with fake and dummy accounts.

**Objective:** Identify potential bot accounts by finding users who have liked every single photo on the platform.

**SQL Code:**

```
select username, count(*) as num_likes  
from users  
join likes  
on users.id = likes.user_id  
group by users.id  
having num_likes = (select count(*) from photos);
```

**Explanation:**

The HAVING clause was introduced in SQL to allow the filtering of query results based on aggregate functions and groupings.

## Output:

The screenshot shows a MySQL Workbench session titled "My Snippets". The SQL editor contains two queries. The first query, starting at line 129, calculates average posts per user. The second query, starting at line 135, identifies users who have liked every photo. The results of the second query are displayed in a table with columns "username" and "num\_likes". The output section shows the execution log with 11 rows of SQL statements and their execution times.

| username         | num_likes |
|------------------|-----------|
| Anya_Hadlett     | 257       |
| Rose33           | 257       |
| Maxwell_Harvors  | 257       |
| Cole_Lewis37     | 257       |
| Morgan17         | 257       |
| Duane60          | 257       |
| JeanneDorff      | 257       |
| HollieAuer20     | 257       |
| Nic_Keog         | 257       |
| Leslie7          | 257       |
| Janelle_Niklaus1 | 257       |
| Bethany20        | 257       |

| Action | Time     | Message   | Duration / Fetch                           |
|--------|----------|---|--|
| 7      | 17:48:04 | SELECT `username` FROM `users` LEFT JOIN `photos` ON `users`.`id` = `photos`.`user_id` WHERE `photos`.`id` IS NULL LIMIT 0, 1000                              | 26 row(s) returned<br>0.015 sec / 0.00 sec |
| 8      | 17:48:59 | SELECT `tag_id`, COUNT(`tag_id`) AS `count` FROM `photo_tags` GROUP BY `tag_id` ORDER BY `count` DESC LIMIT 1   | 5 rows returned<br>1.054 sec / 0.00 sec    |
| 9      | 18:17:24 | SELECT DAYNAME(`created_at`) AS day, COUNT(*) AS total FROM `users` GROUP BY day ORDER BY total DESC LIMIT 1  | 1 row(s) returned<br>0.782 sec / 0.00 sec  |
| 10     | 18:25:14 | SELECT (SELECT COUNT(*) FROM `photos`) / (SELECT COUNT(*) FROM `users`) AS Average LIMIT 0, 1000  | 1 row(s) returned<br>0.125 sec / 0.00 sec  |
| 11     | 18:25:42 | select `username`, count(*) as num_likes from users join likes on users.id = likes.user_id group by users.id having num_likes = (select count(*) from photos) | 13 row(s) returned<br>0.203 sec / 0.00 sec |

## Conclusion:

Multiple IDs that have liked all the photos on the app could potentially be bots as its humanly not possible. The result table displays such IDs.

## Results

The SQL-driven analysis provided:

- Actionable marketing targets (top loyal users, inactive users, and contest winners)
- Data-backed campaign scheduling recommendations
- Popular hashtag statistics for brand collaboration
- Investor-ready metrics on user engagement and platform authenticity

This project improved proficiency in relational analysis, query design, and business reporting, and provided insights for future strategic decisions.

## Drive Link:

<https://drive.google.com/drive/folders/1Fr42vlrvfD3k4FDfRjk4QzBrqAeRPL0L?usp=sharing>