Name: Nikhil Zodape

Email Id.: Nikszodape@gmail.com

Project Name: Instagram User Analytics

Using: MySQL Workbench 8.0 CE

PURPOSE:

The Instagram User Analytics Project is undertaken to leverage SQL. MySQL Workbench to provide a valuable insight into user interaction and engagement with the Instagram app. The primary aim of this project is to empower Instagram's product and marketing teams with data-driven insights. By exploring user interactions, engagement patterns and other key metrics, we can inform the decision-making process.

APPROACH:

Our approach involves using SQL queries to answer specific questions related to user behaviour and engagement on the Instagram platform,

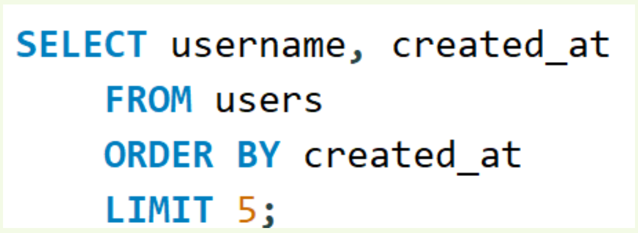
To achieve our goals, we will adopt a structured approach:

SQL TASKS:

A) Marketing Analysis:

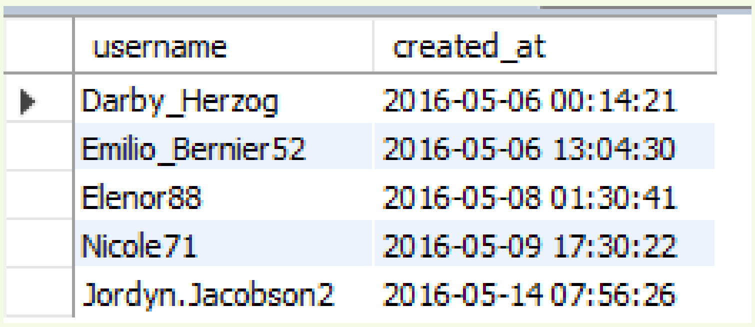
1. **Loyal User Reward:** The marketing team wants to reward the most loyal users i.e., those who have use the platform for the longer time.

   Identify the five oldest users on Instagram:
   1. **"SELECT username, created_at":** In this case we want to fetch the 'username' and 'created_at' columns.
   2. **"FROM users":** The source table is named "users".
   3. **"ORDER BY crated_at":** Then we use to sort the result in ascending order based on the values in the 'created_at' columns.
   4. **"LIMIT 5":** It restricts the number or rows returned in the result set to 5.



```
SELECT username, created_at
    FROM users
    ORDER BY created_at
    LIMIT 5;
```

OUTPUT ➔

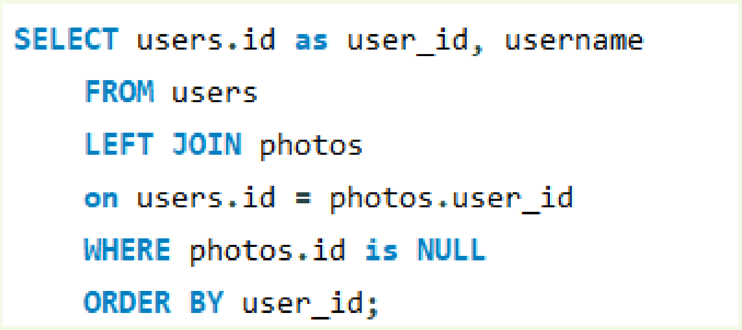


username	created_at
Darby_Herzog	2016-05-06 00:14:21
Emilio_Bernier52	2016-05-06 13:04:30
Elenor88	2016-05-08 01:30:41
Nicole71	2016-05-09 17:30:22
Jordyn.Jacobson2	2016-05-14 07:56:26

2. **INACTIVA USER ENGAGEMENT:** The team wants to encourage inactive users to start posting by sending them promotional emails.

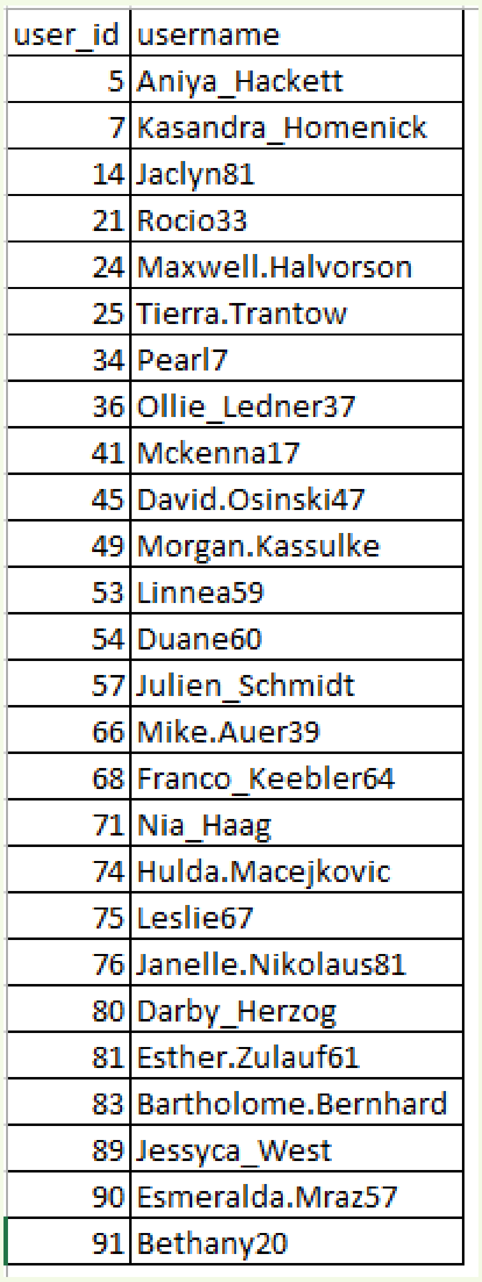
   Identifying users who have never posted a single photo on Instagram.

   1. **'Select user.id as user_id, username':** First, we will retrieve two columns from the 'users' tables i.e., 'users.id and usernames' and renaming user.id as user_id.
   2. **'FROM users' :** The source table name is users.
   3. **'LEFT JOIN photos on users.id = photos.user_id':** This is the left join operation that combines rows of 'users' tables with the rows of 'photos' table based on the condition.
   4. **'WHERE photos.id is NULL':** This where clause filters the result set to include only the rows where the 'photos.id' is Null (blank/empty).
   5. **'ORDER BY user_id':** Then we use to sort the orders in ascending order based on the values of 'user_id'.



```
SELECT users.id as user_id, username
    FROM users
    LEFT JOIN photos
    on users.id = photos.user_id
    WHERE photos.id is NULL
    ORDER BY user_id;
```

OUTPUT ➔



user_id	username
5	Aniya_Hackett
7	Kasandra_Homenick
14	Jaclyn81
21	Rocio33
24	Maxwell.Halvorson
25	Tierra.Trantow
34	Pearl7
36	Ollie_Ledner37
41	Mckenna17
45	David.Osinski47
49	Morgan.Kassulke
53	Linnea59
54	Duane60
57	Julien_Schmidt
66	Mike.Auer39
68	Franco_Keebler64
71	Nia_Haag
74	Hulda.Macejkovic
75	Leslie67
76	Janelle.Nikolaus81
80	Darby_Herzog
81	Esther.Zulauf61
83	Bartholome.Bernhard
89	Jessyca_West
90	Esmeralda.Mraz57
91	Bethany20

From all the users of Instagram we are having 26 users who have never posted a single photo on Instagram.

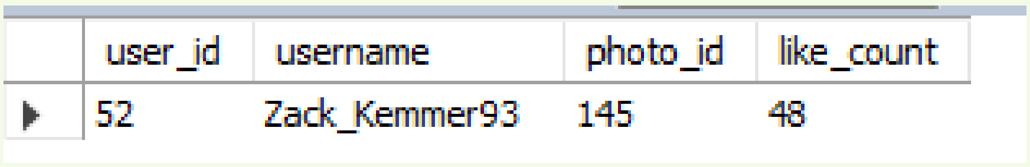
3. **Contest Winner Declaration:** The team has organized a contest where the user with the most like on a single photo wins.

   Determine the winner of the contest and provide their details to the team.

   1. Start by selecting the user's ID, usernames, Photos ID and count of likes for each user and photos.
   2. Then join the 'users' table with the 'photos' table using the user ID as the common field,
   3. Afterwards, use a LEFT JOIN to join the 'likes' table with the result of the previous join based on the photo ID. This allows us to shows all the photos, event those photos are having no likes.
   4. We group the result by the user ID, username and Photo ID. This allows us to count the likes of each users photos.
   5. Later we ORDER the result in the descending order of the likes count using DESC.
   6. Lastly, we use LIMIT 1 so select only the top row which will represent the winner of the contest.

```sql
SELECT u.id AS user_id, u.username AS username,
    p.id AS photo_id, COUNT(l.user_id) AS like_count
    FROM users u
JOIN photos p ON u.id = p.user_id
LEFT JOIN likes l ON p.id = l.photo_id
    GROUP BY u.id, u.username, p.id
    ORDER BY like_count DESC
    LIMIT 1
    ;
```

OUTPUT ➔



	user_id	username	photo_id	like_count
▶	52	Zack_Kemmer93	145	48

The **user_id 52** who is **Zack_Kemmer93** is the winner of the contest who got the **highest likes** i.e, **48** on the **photo_id 145**.

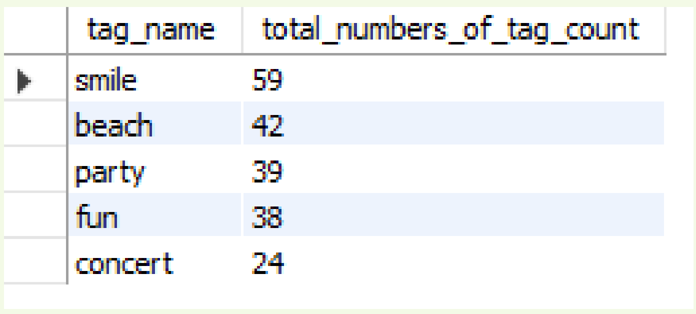
4. **Hashtag Research:** A partner brand wants to know the most popular hashtags to use in their posts to reach the most people.

   Identifying and suggesting the top five most commonly used hashtags on the platform.

   1. Firstly, we select the ‘tag_name’ from the ‘tags’ table and count the number of times each tag has been used into the ‘photo_tags’ table.
   2. Then we perform the LEFT JOIN between the ‘tags’ table (t) and the ‘photo_tags’ table (pt) using the tag ID as the common field.
   3. Then perform the GROUP BY to group the result by ‘tag_name’. This groups the tags and allows us to count how many times each tags has been used.
   4. We order the result in descending order of the tag count using “ORDER BY tag_count DESC”. The tags with the highest usage will appear first
   5. Finally, we use ‘LIMIT 5’ to select only top five most commonly used hashtags.

```sql
SELECT t.tag_name, COUNT(pt.photo_id) AS total_numbers_of_tag_count
FROM tags t
LEFT JOIN photo_tags pt ON t.id = pt.tag_id
GROUP BY t.tag_name
ORDER BY total_numbers_of_tag_count DESC
LIMIT 5;
```

OUTPUT ➔

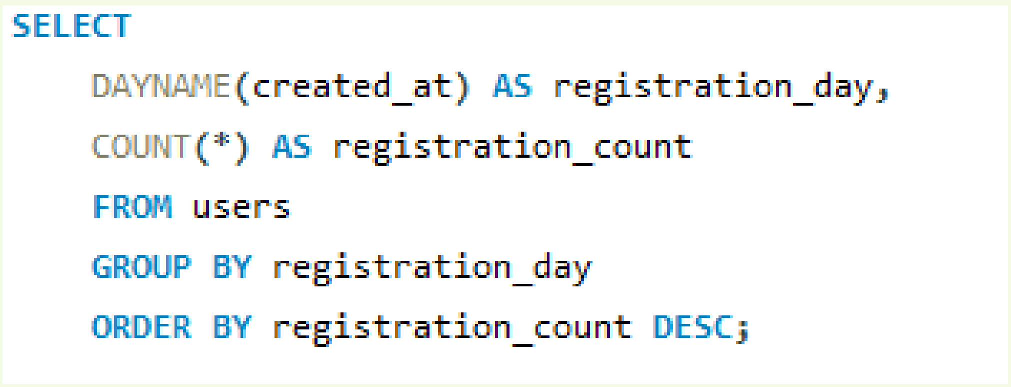


tag_name	total_numbers_of_tag_count
smile	59
beach	42
party	39
fun	38
concert	24

5. **Ad Campaign Launch:** The team wants to know the best day of the week to launch ads.

Determine the day of the week when most users register on Instagram. Provide insights on when to schedule an ad campaign.

1. We use the 'DAYNAME' function to extract the day of the week from the 'created_at' timestamp for each user's registration date.
2. Then count the number of registrations for each day of the week.
3. The "GROUP BY" clause groups the results by the day of the week, so we can count registrations for each day.
4. We use the "ORDER BY" clause to sort the results in descending order of registration count, so the day with the most registrations will appear first.



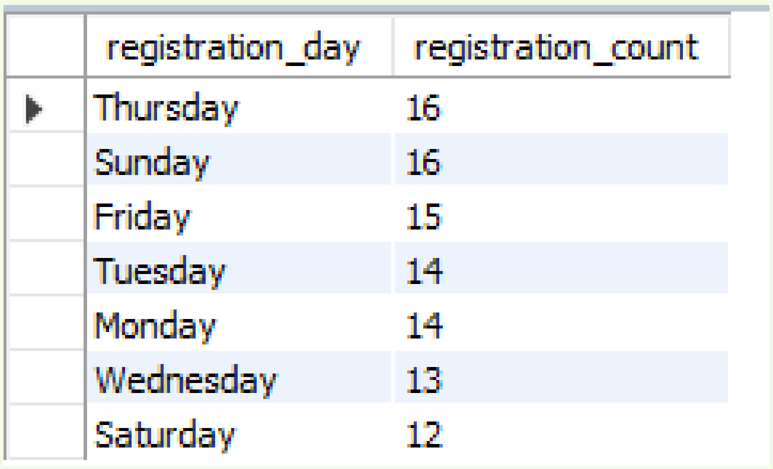
```
SELECT
    DAYNAME(created_at) AS registration_day,
    COUNT(*) AS registration_count
    FROM users
    GROUP BY registration_day
    ORDER BY registration_count DESC;
```

OUTPUT ➔

On Thursday and Sunday, we are getting most of the new users on Instagram.

So, the best day to do the ads campaign will be on Thursday and Sunday.

From all the insight we will be telling marketing team to do the ad campaign on Sunday after all we will be having more audience to represent on Sunday as most of the peoples were having weekend of on Sunday so they will be gathering in public places for the enjoyment.



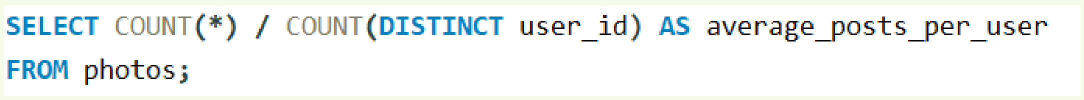
	registration_day	registration_count
▶	Thursday	16
	Sunday	16
	Friday	15
	Tuesday	14
	Monday	14
	Wednesday	13
	Saturday	12

B) Investor Metrics:

1. **User Engagement:** Investors want to know if users are still active and posting on Instagram or if they are making fewer posts.

   A. Calculating the average numbers of posts per user on Instagram.

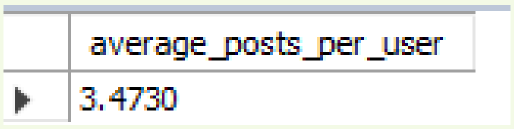
   1. We use the 'COUNT (*)' function to count the total numbers of records on the 'photos' table. This is representing the total number of posts on Instagram.
   2. We use the 'COUNT (DISTINCT user_id)' function to count the number of distinct/unique user ID in the 'photos' table. This is representing the total number of unique users who have posted on Instagram.
   3. We will divide the total number of posts by the total number of unique users to calculate the average number of post per user.



```
SELECT COUNT(*) / COUNT(DISTINCT user_id) AS average_posts_per_user
FROM photos;
```

OUTPUT ➔

The average numbers of posts per user on Instagram is 3.



	average_posts_per_user
▶	3.4730

   B. Also providing the total numbers of phots on Instagram divided by the total number of users.

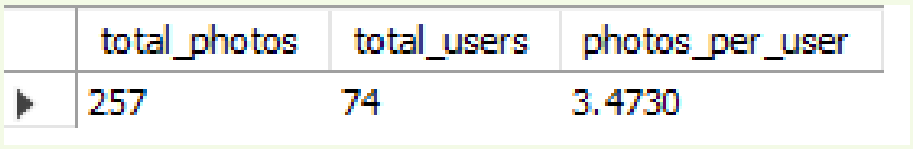
   1. We use the 'COUNT (*)' function to count the total numbers of records in the 'photos' table, which represents the total number of photos on Instagram.
   2. We use the 'COUNT (DISTINCT user_id)' function to count the number of distinct/unique user ID in the 'photos' table. Which representing the total number of unique users on Instagram.

3. We divide the total number of photos by the total number of unique users to calculate the average number of photos per user.

```
SELECT COUNT(*) AS total_photos, COUNT(DISTINCT user_id) AS total_users,
    COUNT(*) / COUNT(DISTINCT user_id) AS photos_per_user
    FROM photos;
```

OUTPUT ➔

The total numbers of photos on Instagram are 257 and the total users are 74 so the photos posted per user on Instagram will be 3 per user.



	total_photos	total_users	photos_per_user
▶	257	74	3.4730

2. **Bots and Fake Accounts:** Investors want to know if the platform is crowded with fake and dummy accounts.

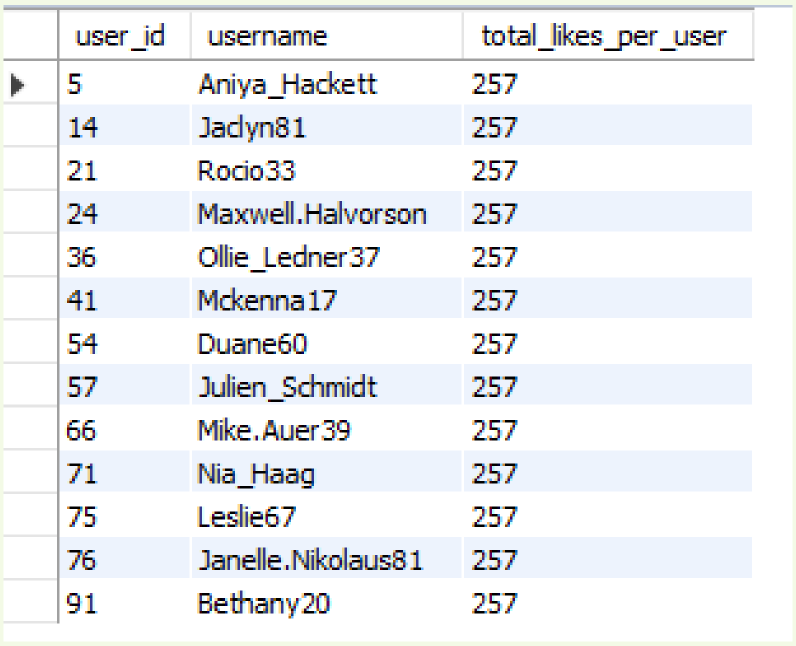
   Identify users (potential bots) who have liked every single photo on the site, as this is not typically possible for a normal user.

   1. 'SELECT u.id AS user_id, u.username, COUNT(*) AS total_likes_per_user': This part of the query selects the user's ID, username, and the count of likes given by that users. Also, we use aliases for the 'users' is 'u' and 'likes' is 'l' tables to the make the query more readable.
   2. 'FROM users u INNER JOIN likes l ON u.id = l.user_id': This part of the query perform an inner join between the 'users' and 'likes' tables using the user ID as the common field. It links users to their likes.
   3. 'GROUP BY u.id, u.username': The result of the join in then grouped by user ID and username. This groups the results by individual users.
   4. 'HAVING total_likes_per_user = (SELECT COUNT(*) FROM photos)': The Having clause filters the grouped results. It checks if the count of likes given by each user (total_likes_per_user) is equal to the total number of photos in the 'photos' table. This is how users who have liked every single photo on the site are identified.

```sql
SELECT u.id AS user_id, u.username, COUNT(*) AS total_likes_per_user
    FROM users u
INNER JOIN likes l ON u.id = l.user_id
    GROUP BY u.id, u.username
    HAVING total_likes_per_user = (SELECT COUNT(*) FROM photos);
```

OUPTUT ➔

From our calculation we got at least 13 users whom we can consider that those accounts are fake and dummy accounts.



user_id	username	total_likes_per_user
5	Aniya_Hackett	257
14	Jaclyn81	257
21	Rocio33	257
24	Maxwell.Halvorson	257
36	Ollie_Ledner37	257
41	Mckenna17	257
54	Duane60	257
57	Julien_Schmidt	257
66	Mike.Auer39	257
71	Nia_Haag	257
75	Leslie67	257
76	Janelle.Nikolaus81	257
91	Bethany20	257