# Instagram User Analytics

**SQL Fundamentals**

**Project Description:** User Analysis for Instagram Product Improvement

**Project Overview:**

In this project, I aim to enhance the performance and user experience of Instagram, a popular digital platform, by conducting a comprehensive user analysis. By closely examining how users engage and interact with the software or mobile application, I will gather valuable insights to inform decision-making across marketing, product development, and management teams. These insights will be instrumental in launching effective marketing campaigns, prioritizing feature development, measuring user engagement, and driving overall business growth.

**Objectives/ Approach:**

1. Understand User Engagement: Through in-depth analysis, I will gain a deep understanding of how users engage with Instagram, including their behaviors, preferences, and patterns of interaction.

2. Derive Business Insights: By analyzing user data, I will extract meaningful insights that can inform decision-making for various teams within the organization, including marketing, product development, and management.

3. Optimize Marketing Campaigns: Leveraging the insights obtained, I will assist the marketing team in crafting targeted and impactful campaigns that resonate with Instagram users and drive user acquisition and retention.

4. Enhance Product Features: Based on user analysis, I will provide valuable recommendations to the product development team, enabling them to prioritize and build features that align with user needs and desires, ultimately improving the overall user experience.

5. Measure User Engagement: By implementing appropriate metrics and analytics, I will establish a framework to measure user engagement and track the success of Instagram, allowing us to assess the impact of product changes and enhancements over time.

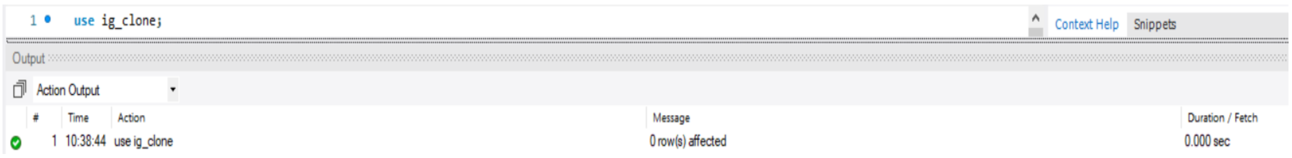
6. Continuous Improvement: Throughout the project, I will iterate on our analysis and insights, ensuring that I stay aligned with evolving user preferences and market dynamics, while continuously enhancing the Instagram platform.

## Tech-Stack Used:

For the user analysis project at Instagram, I will utilize a tech-stack that includes SQL as one of the core components. SQL (Structured Query Language) will be employed for extracting, manipulating, and analyzing data from Instagram's vast databases. With SQL, I can efficiently retrieve user engagement data, perform complex queries to uncover meaningful insights, and generate reports that will drive decision-making across marketing, product development, and management teams. SQL's versatility and poIr in handling large datasets make it an essential tool for conducting comprehensive user analysis and deriving valuable business insights for Instagram's growth and success.

## Understanding database:

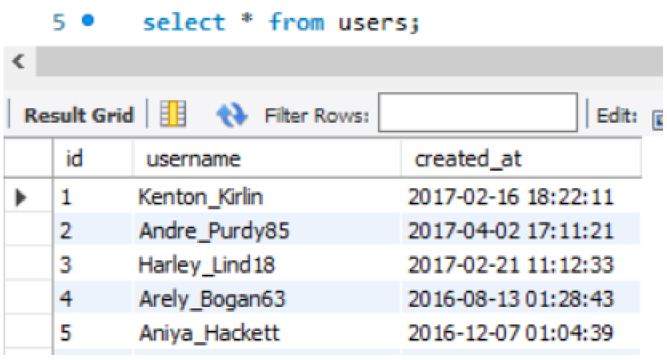
Using ‘ig_clone’ database:



```
use ig_clone;
```

```
#  Time      Action        Message            Duration / Fetch
1  10:38:44  use ig_clone  0 row(s) affected  0.000 sec
```

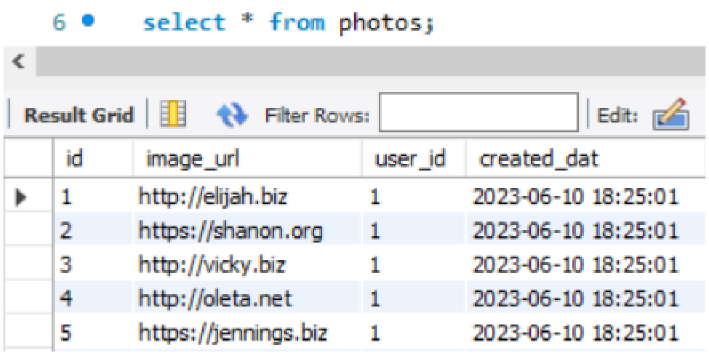
Users table: with 3 columns and 100 rows



```
select * from users;
```

| id | username | created_at |
|---|---|---|
| 1 | Kenton_Kirlin | 2017-02-16 18:22:11 |
| 2 | Andre_Purdy85 | 2017-04-02 17:11:21 |
| 3 | Harley_Lind18 | 2017-02-21 11:12:33 |
| 4 | Arely_Bogan63 | 2016-08-13 01:28:43 |
| 5 | Aniya_Hackett | 2016-12-07 01:04:39 |

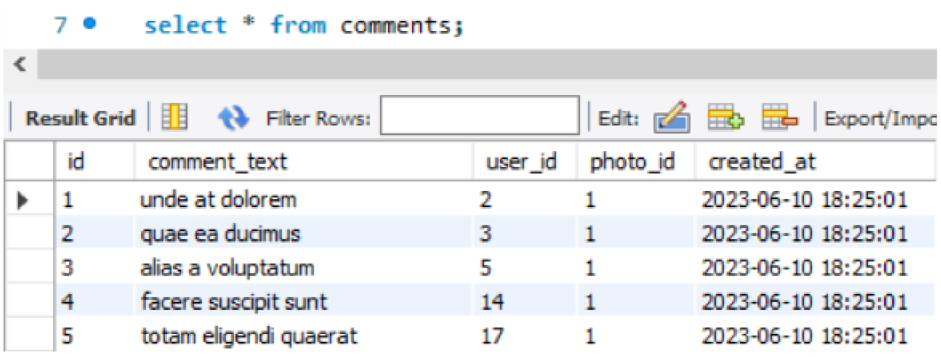
Photos table: with 4 columns and 257 rows



```
select * from photos;
```

| id | image_url | user_id | created_dat |
|---|---|---|---|
| 1 | http://elijah.biz | 1 | 2023-06-10 18:25:01 |
| 2 | https://shanon.org | 1 | 2023-06-10 18:25:01 |
| 3 | http://vicky.biz | 1 | 2023-06-10 18:25:01 |
| 4 | http://oleta.net | 1 | 2023-06-10 18:25:01 |
| 5 | https://jennings.biz | 1 | 2023-06-10 18:25:01 |

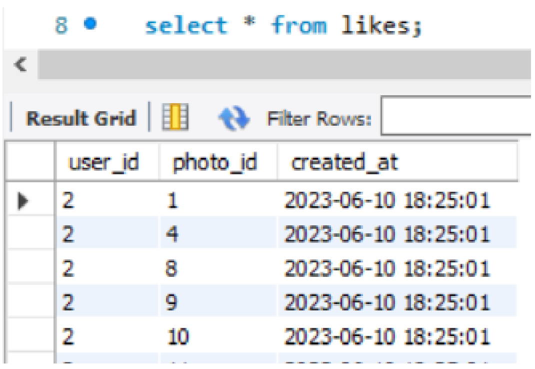
Comments table: with 5 columns and 7488 rows



```
select * from comments;
```

| id | comment_text | user_id | photo_id | created_at |
|---|---|---|---|---|
| 1 | unde at dolorem | 2 | 1 | 2023-06-10 18:25:01 |
| 2 | quae ea ducimus | 3 | 1 | 2023-06-10 18:25:01 |
| 3 | alias a voluptatum | 5 | 1 | 2023-06-10 18:25:01 |
| 4 | facere suscipit sunt | 14 | 1 | 2023-06-10 18:25:01 |
| 5 | totam eligendi quaerat | 17 | 1 | 2023-06-10 18:25:01 |

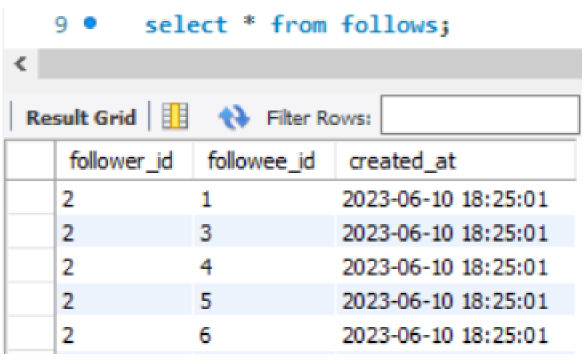
Likes table: with 3 columns and 8782 rows



```
select * from likes;
```

| user_id | photo_id | created_at |
|---|---|---|
| 2 | 1 | 2023-06-10 18:25:01 |
| 2 | 4 | 2023-06-10 18:25:01 |
| 2 | 8 | 2023-06-10 18:25:01 |
| 2 | 9 | 2023-06-10 18:25:01 |
| 2 | 10 | 2023-06-10 18:25:01 |

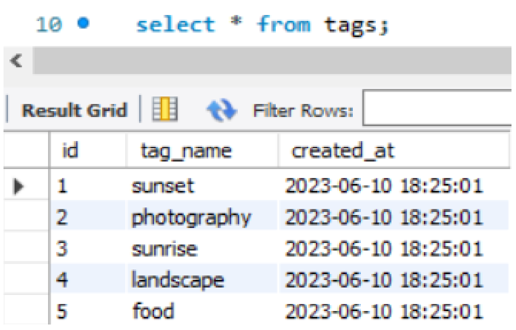
Follows table: with 3 columns and 7623 rows



```
select * from follows;
```

| follower_id | followee_id | created_at |
|---|---|---|
| 2 | 1 | 2023-06-10 18:25:01 |
| 2 | 3 | 2023-06-10 18:25:01 |
| 2 | 4 | 2023-06-10 18:25:01 |
| 2 | 5 | 2023-06-10 18:25:01 |
| 2 | 6 | 2023-06-10 18:25:01 |

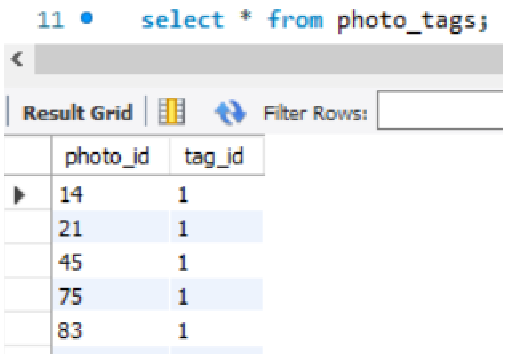
Tags table: with 3 columns and 21 rows



```
select * from tags;
```

| id | tag_name | created_at |
|---|---|---|
| 1 | sunset | 2023-06-10 18:25:01 |
| 2 | photography | 2023-06-10 18:25:01 |
| 3 | sunrise | 2023-06-10 18:25:01 |
| 4 | landscape | 2023-06-10 18:25:01 |
| 5 | food | 2023-06-10 18:25:01 |

photo_tags table: with 2 columns and 501 rows



```
select * from photo_tags;
```

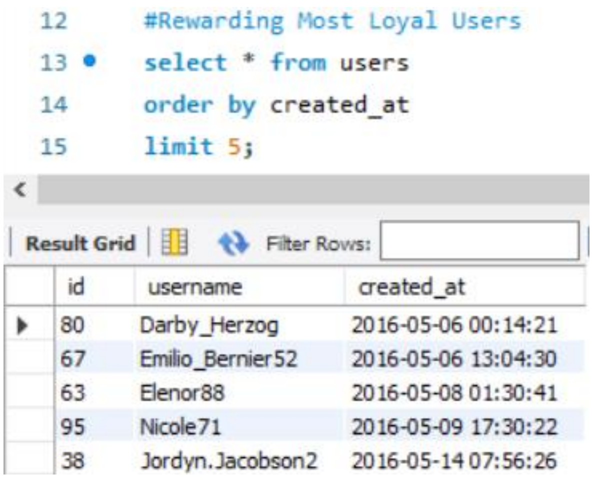
| | photo_id | tag_id |
|---|---|---|
| ▶ | 14 | 1 |
| | 21 | 1 |
| | 45 | 1 |
| | 75 | 1 |
| | 83 | 1 |

**Report:**

**A) Marketing: The marketing team wants to launch some campaigns, and they need your help with the following**

**1. Rewarding Most Loyal Users:** People who have been using the platform for the longest time.

My Task: Find the 5 oldest users of the Instagram from the database provided.



```
#Rewarding Most Loyal Users
select * from users
order by created_at
limit 5;
```

| | id | username | created_at |
|---|---|---|---|
| ▶ | 80 | Darby_Herzog | 2016-05-06 00:14:21 |
| | 67 | Emilio_Bernier52 | 2016-05-06 13:04:30 |
| | 63 | Elenor88 | 2016-05-08 01:30:41 |
| | 95 | Nicole71 | 2016-05-09 17:30:22 |
| | 38 | Jordyn.Jacobson2 | 2016-05-14 07:56:26 |

Those are the 5 oldest users of the Instagram.

The code provided is a SQL query that selects all columns from the "users" table and orders the results by the "created_at" column in ascending order. It then limits the output to the first 5 rows.

Here's an explanation of the code:

SELECT : This statement selects all columns from the "users" table. The asterisk () is a wildcard that represents all columns in the table.

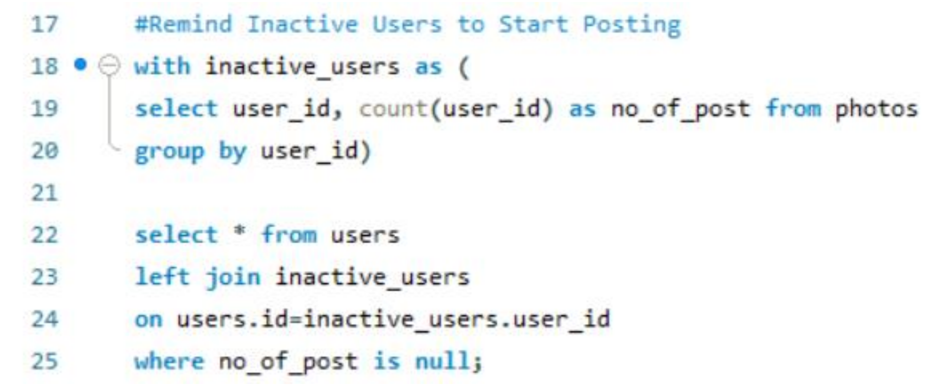
FROM users: This specifies the table named "users" from which the data will be retrieved. The table name follows the FROM keyword.

ORDER BY created_at: This clause orders the results based on the "created_at" column. The data will be sorted in ascending order by default. If you want to sort in descending order, you can use ORDER BY created_at DESC.

LIMIT 5: This clause restricts the number of rows returned by the query to 5. In this case, it will return the first 5 rows according to the specified order.

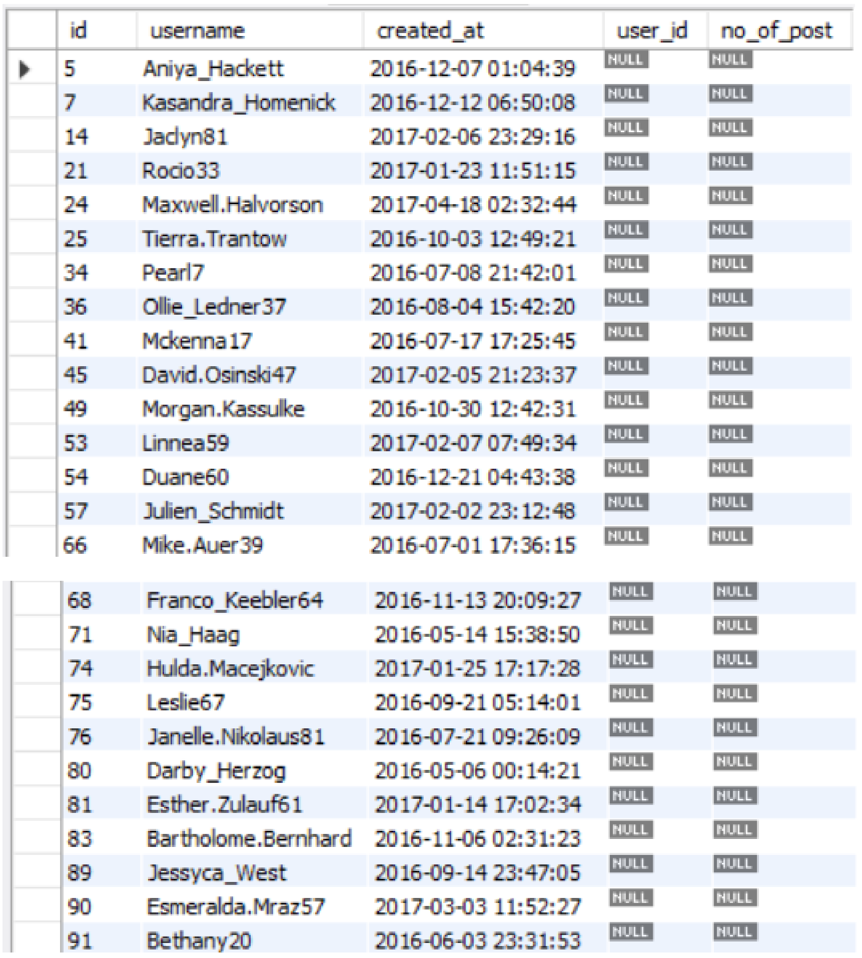
2. **Remind Inactive Users to Start Posting**: By sending them promotional emails to post their 1st photo.

My Task: Find the users who have never posted a single photo on Instagram.



```
#Remind Inactive Users to Start Posting
with inactive_users as (
select user_id, count(user_id) as no_of_post from photos
group by user_id)

select * from users
left join inactive_users
on users.id=inactive_users.user_id
where no_of_post is null;
```



| id | username | created_at | user_id | no_of_post |
|---|---|---|---|---|
| 5 | Aniya_Hackett | 2016-12-07 01:04:39 | NULL | NULL |
| 7 | Kasandra_Homenick | 2016-12-12 06:50:08 | NULL | NULL |
| 14 | Jaclyn81 | 2017-02-06 23:29:16 | NULL | NULL |
| 21 | Rocio33 | 2017-01-23 11:51:15 | NULL | NULL |
| 24 | Maxwell.Halvorson | 2017-04-18 02:32:44 | NULL | NULL |
| 25 | Tierra.Trantow | 2016-10-03 12:49:21 | NULL | NULL |
| 34 | Pearl7 | 2016-07-08 21:42:01 | NULL | NULL |
| 36 | Ollie_Ledner37 | 2016-08-04 15:42:20 | NULL | NULL |
| 41 | Mckenna17 | 2016-07-17 17:25:45 | NULL | NULL |
| 45 | David.Osinski47 | 2017-02-05 21:23:37 | NULL | NULL |
| 49 | Morgan.Kassulke | 2016-10-30 12:42:31 | NULL | NULL |
| 53 | Linnea59 | 2017-02-07 07:49:34 | NULL | NULL |
| 54 | Duane60 | 2016-12-21 04:43:38 | NULL | NULL |
| 57 | Julien_Schmidt | 2017-02-02 23:12:48 | NULL | NULL |
| 66 | Mike.Auer39 | 2016-07-01 17:36:15 | NULL | NULL |
| 68 | Franco_Keebler64 | 2016-11-13 20:09:27 | NULL | NULL |
| 71 | Nia_Haag | 2016-05-14 15:38:50 | NULL | NULL |
| 74 | Hulda.Macejkovic | 2017-01-25 17:17:28 | NULL | NULL |
| 75 | Leslie67 | 2016-09-21 05:14:01 | NULL | NULL |
| 76 | Janelle.Nikolaus81 | 2016-07-21 09:26:09 | NULL | NULL |
| 80 | Darby_Herzog | 2016-05-06 00:14:21 | NULL | NULL |
| 81 | Esther.Zulauf61 | 2017-01-14 17:02:34 | NULL | NULL |
| 83 | Bartholome.Bernhard | 2016-11-06 02:31:23 | NULL | NULL |
| 89 | Jessyca_West | 2016-09-14 23:47:05 | NULL | NULL |
| 90 | Esmeralda.Mraz57 | 2017-03-03 11:52:27 | NULL | NULL |
| 91 | Bethany20 | 2016-06-03 23:31:53 | NULL | NULL |

26 users never posted a single photo on Instagram.

The code provided is a SQL query that involves a subquery and a left join. It selects all columns from the "users" table and left joins it with a subquery named "inactive_users". The subquery calculates the number of posts made by each user in the "photos" table. The main query filters the results by selecting only the rows where the "no_of_post" column from the subquery is null.

Here's an explanation of the code:

Subquery "inactive_users":

WITH inactive_users AS (...): This syntax allows you to define a temporary table or view called "inactive_users" within the query. The subquery inside the parentheses calculates the number of posts made by each user in the "photos" table.

SELECT user_id, COUNT(user_id) AS no_of_post FROM photos GROUP BY user_id: This subquery selects the "user_id" column from the "photos" table and uses the COUNT function to count the number of occurrences of each "user_id". The result is grouped by "user_id" to obtain the count of posts made by each user.

Main query:

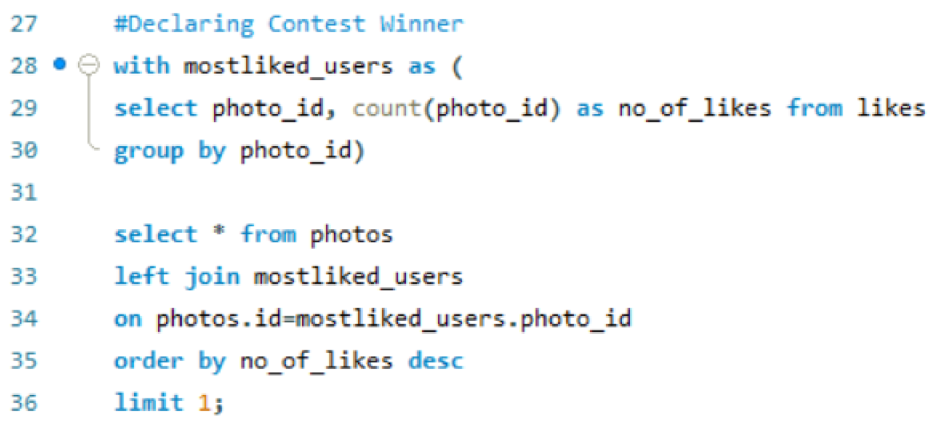
SELECT * FROM users: This statement selects all columns from the "users" table.

LEFT JOIN inactive_users ON users.id = inactive_users.user_id: This performs a left join between the "users" table and the "inactive_users" subquery. It matches rows based on the equality of "users.id" and "inactive_users.user_id".

WHERE no_of_post IS NULL: This condition filters the results and selects only the rows where the "no_of_post" column (from the subquery) is null, indicating that the user has not made any posts.

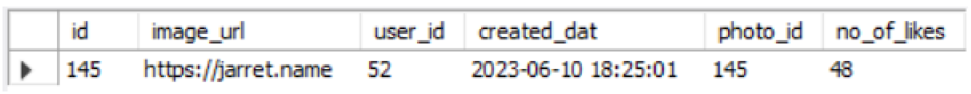
3. **Declaring Contest Winner**: The team started a contest and the user who gets the most likes on a single photo will win the contest now they wish to declare the winner.

My Task: Identify the winner of the contest and provide their details to the team



```
#Declaring Contest Winner
with mostliked_users as (
select photo_id, count(photo_id) as no_of_likes from likes
group by photo_id)

select * from photos
left join mostliked_users
on photos.id=mostliked_users.photo_id
order by no_of_likes desc
limit 1;
```



| id | image_url | user_id | created_dat | photo_id | no_of_likes |
|---|---|---|---|---|---|
| 145 | https://jarret.name | 52 | 2023-06-10 18:25:01 | 145 | 48 |

The user_id 52 is the winner and the name of the user is ‘Zack_Kemmer93’.

The code provided is a SQL query that involves a subquery and a left join. It selects all columns from the "photos" table and left joins it with a subquery named "mostliked_users". The subquery calculates the number of likes received by each photo in the "likes" table. The main query then orders the results based on the "no_of_likes" column from the subquery in descending order and limits the output to only one row.

Here's an explanation of the code:

Subquery "mostliked_users":

WITH mostliked_users AS (...): This syntax allows you to define a temporary table or view called "mostliked_users" within the query. The subquery inside the parentheses calculates the number of likes received by each photo in the "likes" table.

SELECT photo_id, COUNT(photo_id) AS no_of_likes FROM likes GROUP BY photo_id: This subquery selects the "photo_id" column from the "likes" table and uses the COUNT function to count the number of occurrences of each "photo_id". The result is grouped by "photo_id" to obtain the count of likes received by each photo.

Main query:

SELECT * FROM photos: This statement selects all columns from the "photos" table.

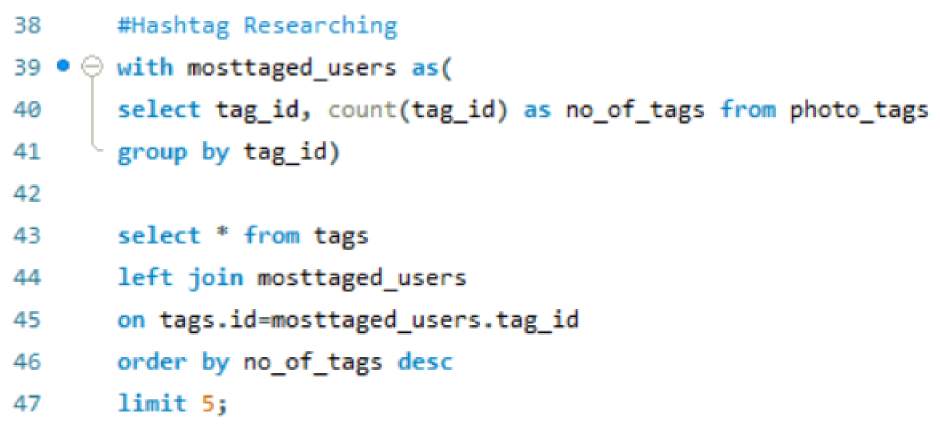
LEFT JOIN mostliked_users ON photos.id = mostliked_users.photo_id: This performs a left join between the "photos" table and the "mostliked_users" subquery. It matches rows based on the equality of "photos.id" and "mostliked_users.photo_id".

ORDER BY no_of_likes DESC: This clause orders the results based on the "no_of_likes" column (from the subquery) in descending order, meaning that photos with the highest number of likes will appear first.

LIMIT 1: This clause limits the output to only one row, effectively retrieving the photo with the highest number of likes.

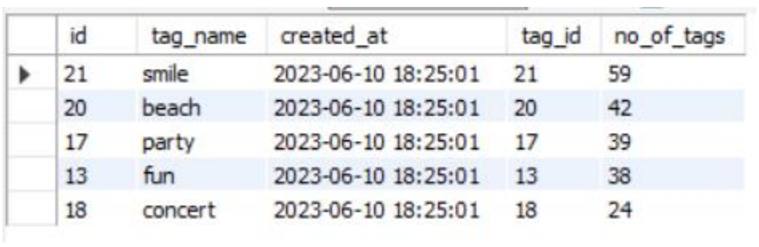
4. **Hashtag Researching**: A partner brand wants to know, which hashtags to use in the post to reach the most people on the platform.

My Task: Identify and suggest the top 5 most commonly used hashtags on the platform.



```
#Hashtag Researching
with mosttaged_users as(
select tag_id, count(tag_id) as no_of_tags from photo_tags
group by tag_id)

select * from tags
left join mosttaged_users
on tags.id=mosttaged_users.tag_id
order by no_of_tags desc
limit 5;
```



| id | tag_name | created_at | tag_id | no_of_tags |
|---|---|---|---|---|
| 21 | smile | 2023-06-10 18:25:01 | 21 | 59 |
| 20 | beach | 2023-06-10 18:25:01 | 20 | 42 |
| 17 | party | 2023-06-10 18:25:01 | 17 | 39 |
| 13 | fun | 2023-06-10 18:25:01 | 13 | 38 |
| 18 | concert | 2023-06-10 18:25:01 | 18 | 24 |

From the column tag_name the observations are the most commonly used tags.

The code provided is a SQL query that involves a subquery and a left join. It selects all columns from the "tags" table and left joins it with a subquery named "mosttaged_users". The subquery calculates the number of times each tag has been used in the "photo_tags" table. The main query then orders the results based on the "no_of_tags" column from the subquery in descending order and limits the output to the top 5 tags with the highest usage count.

Here's an explanation of the code:

Subquery "mosttaged_users":

WITH mosttaged_users AS (...): This syntax allows you to define a temporary table or view called "mosttaged_users" within the query. The subquery inside the parentheses calculates the number of times each tag has been used in the "photo_tags" table.

SELECT tag_id, COUNT(tag_id) AS no_of_tags FROM photo_tags GROUP BY tag_id: This subquery selects the "tag_id" column from the "photo_tags" table and uses the COUNT function to count the number of occurrences of each "tag_id". The result is grouped by "tag_id" to obtain the count of tag usage.

Main query:

SELECT * FROM tags: This statement selects all columns from the "tags" table.

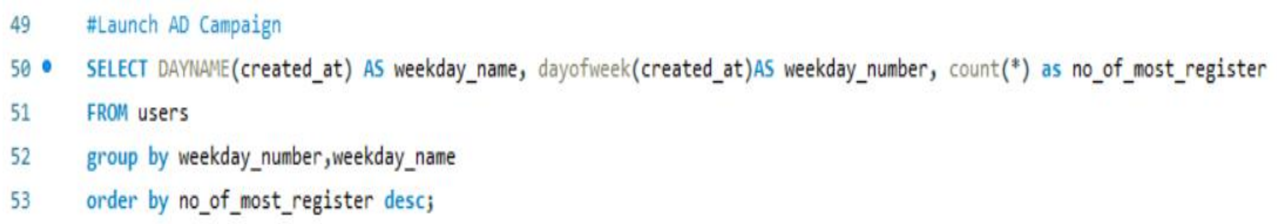
LEFT JOIN mosttaged_users ON tags.id = mosttaged_users.tag_id: This performs a left join between the "tags" table and the "mosttaged_users" subquery. It matches rows based on the equality of "tags.id" and "mosttaged_users.tag_id".

ORDER BY no_of_tags DESC: This clause orders the results based on the "no_of_tags" column (from the subquery) in descending order, meaning that tags with the highest usage count will appear first.

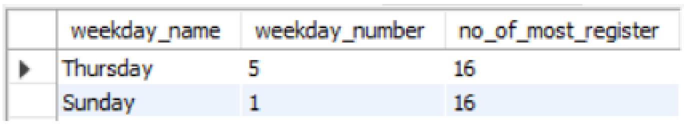
LIMIT 5: This clause limits the output to only 5 rows, effectively retrieving the top 5 tags with the highest usage count.

5. **Launch AD Campaign:** The team wants to know, which day would be the best day to launch ADs.

My Task: What day of the week do most users register on? Provide insights on when to schedule an ad campaign.



```
#Launch AD Campaign
SELECT DAYNAME(created_at) AS weekday_name, dayofweek(created_at)AS weekday_number, count(*) as no_of_most_register
FROM users
group by weekday_number,weekday_name
order by no_of_most_register desc;
```



| weekday_name | weekday_number | no_of_most_register |
|---|---|---|
| Thursday | 5 | 16 |
| Sunday | 1 | 16 |

Sunday and Thursday will the best days to launch Ads.

The code provided is a SQL query that retrieves the weekday name and number from the "created_at" column of the "users" table. It then counts the number of registrations that occurred on each weekday and groups the results by both the weekday number and name. Finally, the query orders the results in descending order based on the count of registrations.

Here's an explanation of the code:

SELECT DAYNAME(created_at) AS weekday_name: This statement extracts the weekday name (e.g., Monday, Tuesday) from the "created_at" column using the DAYNAME function. It aliases the result as "weekday_name" for clarity in the output.

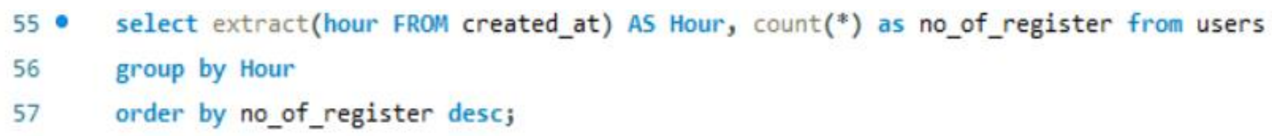
dayofweek(created_at) AS weekday_number: This statement retrieves the weekday number (1 to 7, where 1 represents Sunday) from the "created_at" column using the DAYOFWEEK function. It aliases the result as "weekday_number" for clarity in the output.

count(*) AS no_of_most_register: This statement uses the COUNT function to count the number of occurrences (registrations) for each combination of weekday number and name. It aliases the result as "no_of_most_register" for clarity in the output.

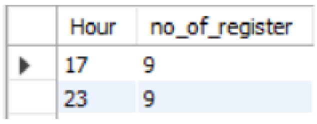
FROM users: This specifies the table named "users" from which the data will be retrieved.

GROUP BY weekday_number, weekday_name: This clause groups the results based on both the weekday number and name, ensuring that the counts are aggregated for each unique combination.

ORDER BY no_of_most_register DESC: This clause orders the results in descending order based on the count of registrations, from highest to lowest.



```
select extract(hour FROM created_at) AS Hour, count(*) as no_of_register from users
group by Hour
order by no_of_register desc;
```



| Hour | no_of_register |
|---|---|
| 17 | 9 |
| 23 | 9 |

17 hour and 23 hour of a day represent 5PM and 11PM respectively. We can schedule an ad campaign on those times

The code provided is a SQL query that extracts the hour component from the "created_at" column of the "users" table. It then counts the number of registrations that occurred during each hour of the day, groups the results by hour, and finally orders the results in descending order based on the count of registrations.

Here's an explanation of the code:

SELECT EXTRACT(HOUR FROM created_at) AS Hour: This statement uses the EXTRACT function to retrieve the hour component from the "created_at" column. It aliases the result as "Hour" for clarity in the output.

COUNT(*) AS no_of_register: This statement uses the COUNT function to count the number of occurrences (registrations) for each hour.

FROM users: This specifies the table named "users" from which the data will be retrieved.

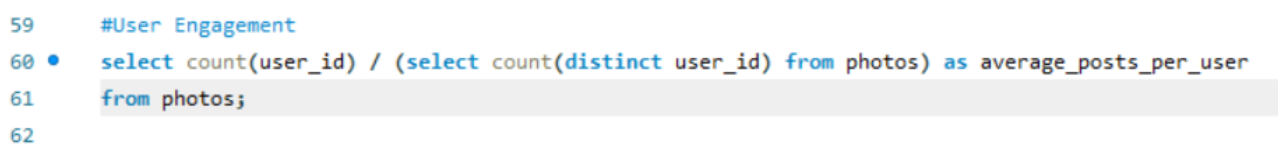
GROUP BY Hour: This clause groups the results based on the extracted hour component, ensuring that the counts are aggregated for each unique hour.

ORDER BY no_of_register DESC: This clause orders the results in descending order based on the count of registrations, from highest to lowest.

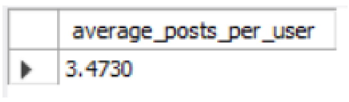
**B) Investor Metrics: Our investors want to know if Instagram is performing well and is not becoming redundant like Facebook, they want to assess the app on the following grounds**

**1. User Engagement:** Are users still as active and post on Instagram or they are making fewer posts

My Task: Provide how many times does average user posts on Instagram. Also, provide the total number of photos on Instagram/total number of users.

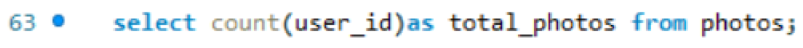


```
#User Engagement
select count(user_id) / (select count(distinct user_id) from photos) as average_posts_per_user
from photos;
```

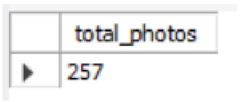


| | average_posts_per_user |
|---|---|
| ▶ | 3.4730 |

3.4730 times does average user posts on Instagram.

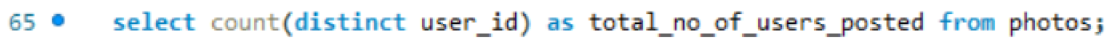


```
select count(user_id)as total_photos from photos;
```

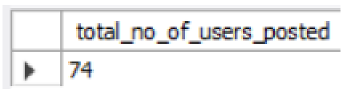


| | total_photos |
|---|---|
| ▶ | 257 |

257 is the total number of photos on Instagram.



```
select count(distinct user_id) as total_no_of_users_posted from photos;
```



| | total_no_of_users_posted |
|---|---|
| ▶ | 74 |

There is actually 100 users but only 74 of them are posted photos.

The code provided consists of three separate SQL queries that operate on the "photos" table.

Here's an explanation of each query:

SELECT count(user_id) / (SELECT count(DISTINCT user_id) FROM photos) AS average_posts_per_user FROM photos;

This query calculates the average number of posts per user in the "photos" table. Here's the breakdown:

SELECT count(user_id): This selects the count of rows where the "user_id" column is not null, representing the total number of posts.

/: This is the division operator.

(SELECT count(DISTINCT user_id) FROM photos): This subquery calculates the count of distinct user IDs in the "photos" table, representing the total number of unique users who have posted.

AS average_posts_per_user: This aliases the result of the division as "average_posts_per_user" for clarity in the output.

SELECT count(user_id) AS total_photos FROM photos;

This query counts the total number of photos in the "photos" table. Here's the breakdown:

SELECT count(user_id): This selects the count of rows where the "user_id" column is not null, representing the total number of photos.

AS total_photos: This aliases the result as "total_photos" for clarity in the output.

SELECT count(DISTINCT user_id) AS total_no_of_users_posted FROM photos;

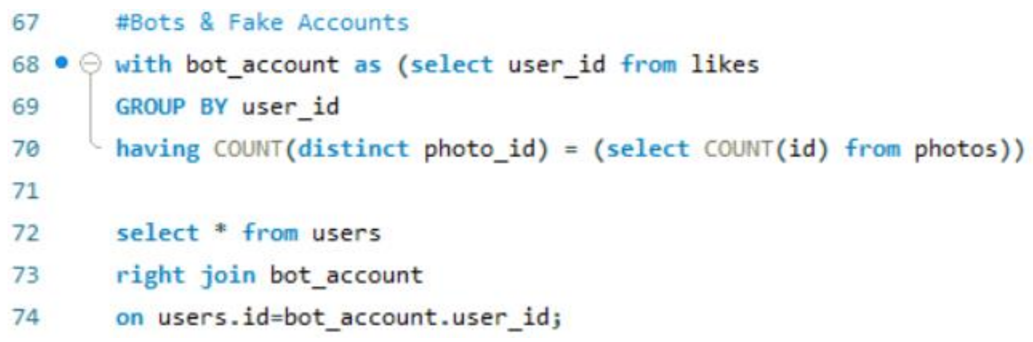
This query counts the total number of unique users who have posted photos in the "photos" table. Here's the breakdown:

SELECT count(DISTINCT user_id): This selects the count of distinct user IDs in the "photos" table, representing the total number of unique users who have posted.

AS total_no_of_users_posted: This aliases the result as "total_no_of_users_posted" for clarity in the output.

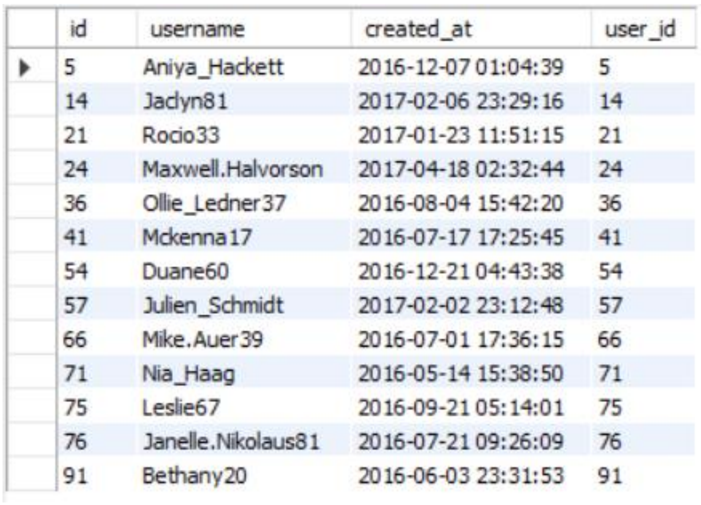
**2. Bots & Fake Accounts:** The investors want to know if the platform is crowded with fake and dummy accounts

My Task: Provide data on users (bots) who have liked every single photo on the site (since any normal user would not be able to do this).



```
#Bots & Fake Accounts
with bot_account as (select user_id from likes
GROUP BY user_id
having COUNT(distinct photo_id) = (select COUNT(id) from photos))

select * from users
right join bot_account
on users.id=bot_account.user_id;
```



| id | username | created_at | user_id |
|---|---|---|---|
| 5 | Aniya_Hackett | 2016-12-07 01:04:39 | 5 |
| 14 | Jaclyn81 | 2017-02-06 23:29:16 | 14 |
| 21 | Rocio33 | 2017-01-23 11:51:15 | 21 |
| 24 | Maxwell.Halvorson | 2017-04-18 02:32:44 | 24 |
| 36 | Ollie_Ledner37 | 2016-08-04 15:42:20 | 36 |
| 41 | Mckenna17 | 2016-07-17 17:25:45 | 41 |
| 54 | Duane60 | 2016-12-21 04:43:38 | 54 |
| 57 | Julien_Schmidt | 2017-02-02 23:12:48 | 57 |
| 66 | Mike.Auer39 | 2016-07-01 17:36:15 | 66 |
| 71 | Nia_Haag | 2016-05-14 15:38:50 | 71 |
| 75 | Leslie67 | 2016-09-21 05:14:01 | 75 |
| 76 | Janelle.Nikolaus81 | 2016-07-21 09:26:09 | 76 |
| 91 | Bethany20 | 2016-06-03 23:31:53 | 91 |

Those are the users who liked every single photo on the site.

The code provided is a SQL query that involves a subquery and a right join. It selects all columns from the "users" table and performs a right join with a subquery named "bot_account". The subquery identifies users who have liked all the photos in the "photos" table, indicating potential bot accounts. Here's an explanation of the code:

Subquery "bot_account":

WITH bot_account AS (...): This syntax allows you to define a temporary table or view called "bot_account" within the query. The subquery inside the parentheses identifies users who have liked all the photos in the "photos" table, suggesting potential bot accounts.

SELECT user_id FROM likes GROUP BY user_id HAVING COUNT(DISTINCT photo_id) = (SELECT COUNT(id) FROM photos): This subquery selects the "user_id" column from the "likes" table and groups the results by "user_id". The HAVING clause ensures that only users who have liked all the distinct "photo_id" values (count matching the total number of photos in the "photos" table) are included in the results.

Main query:

SELECT * FROM users: This statement selects all columns from the "users" table.

RIGHT JOIN bot_account ON users.id = bot_account.user_id: This performs a right join between the "users" table and the "bot_account" subquery. It matches rows based on the equality of "users.id" and "bot_account.user_id".

**Conclusion:**

Through this user analysis project, we aim to leverage data-driven insights to drive the growth and success of Instagram. By understanding user engagement, providing actionable recommendations, and continuously measuring user satisfaction, we will empower Instagram's marketing, product, and management teams to make informed decisions that enhance the platform, improve user experiences, and ultimately drive business growth.