Instagram user analytics

Presented by:
Radhika N

Project Description:

The purpose of this project is to analyse user interactions and engagement with the Instagram app using SQL and MYSQL workbench. The goal is to provide valuable insights that can inform decision-making for the product team at Instagram these insights can be used to improve the user experience, develop new features, and optimize marketing campaigns.

Approach:

Understanding data: the first step is to thoroughly understand the structure of the Instagram user data available in the database. This includes identifying relevant tables, understanding the relationships between them, and familiarizing myself with the meaning of each column.

Defining objectives and key questions: By understanding the objectives and defining specific questions that need to be answered, the analysis should be aligned with business goals of Instagram such as improving user engagement enhancing user experience and identifying growth opportunities.

Before diving into analysis, I ensure that the data is clean and ready for analysis, the provided data is in a format suitable for analysis.

Query writing: using SQL queries I extract relevant data from the database to address the key questions identified earlier this may involve joining multiple tables, filtering data based on specific criteria and aggregating data over time periods or user segments.

Data analysis: I conduct various analyses to uncover insights apart from key questions to be answered. This could include analysing user engagement metrics such as likes, comments and follower growth and many other possible insights.

Tech Stack used:

MYSQL Workbench: It is a open source tool which can support multiple database systems that can perform complex calculations and data manipulation with ease.

Insights

A.Marketing Analysis

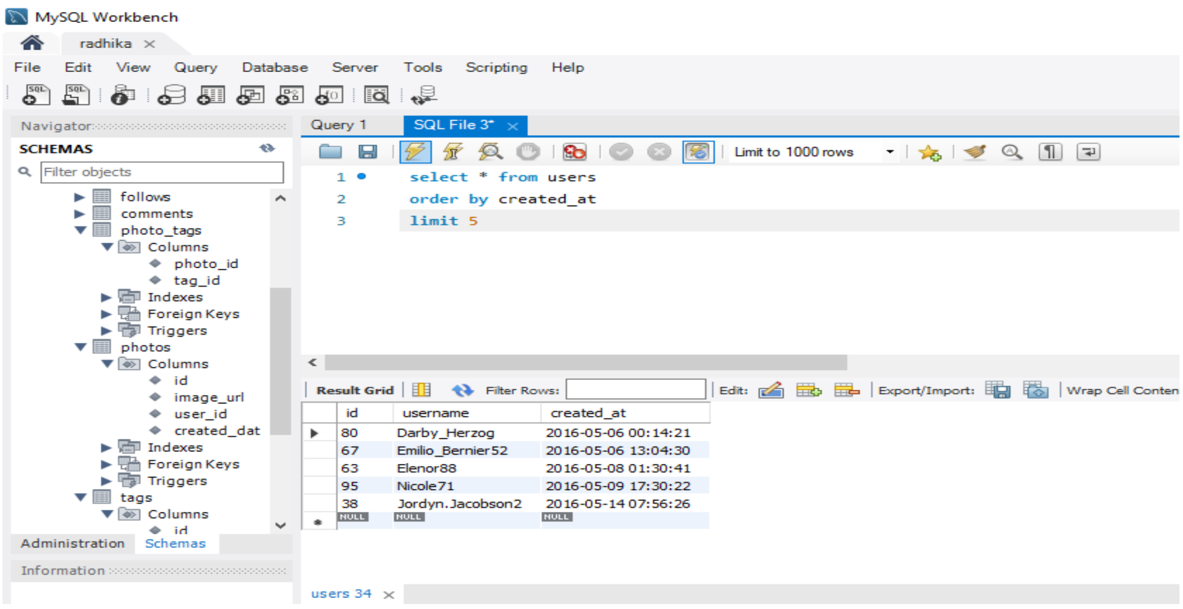
Task 1(Loyal user reward): Top 5 oldest users on Instagram

SQL Code:

select * from users
order by created_at
limit 5

the query retrieves data from table" users" by ordering the result based on created_at column and limits the rows by only top 5 users as per the earliest date in the created_at column giving us the five users who have been using the platform for longest time.

The top 5 oldest users with their ID username and the time they created their account has been displayed in the results.



```
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
NULL	NULL	NULL

Task 2(Inactive user engagement): people who have not posted a single picture

SQL Code:

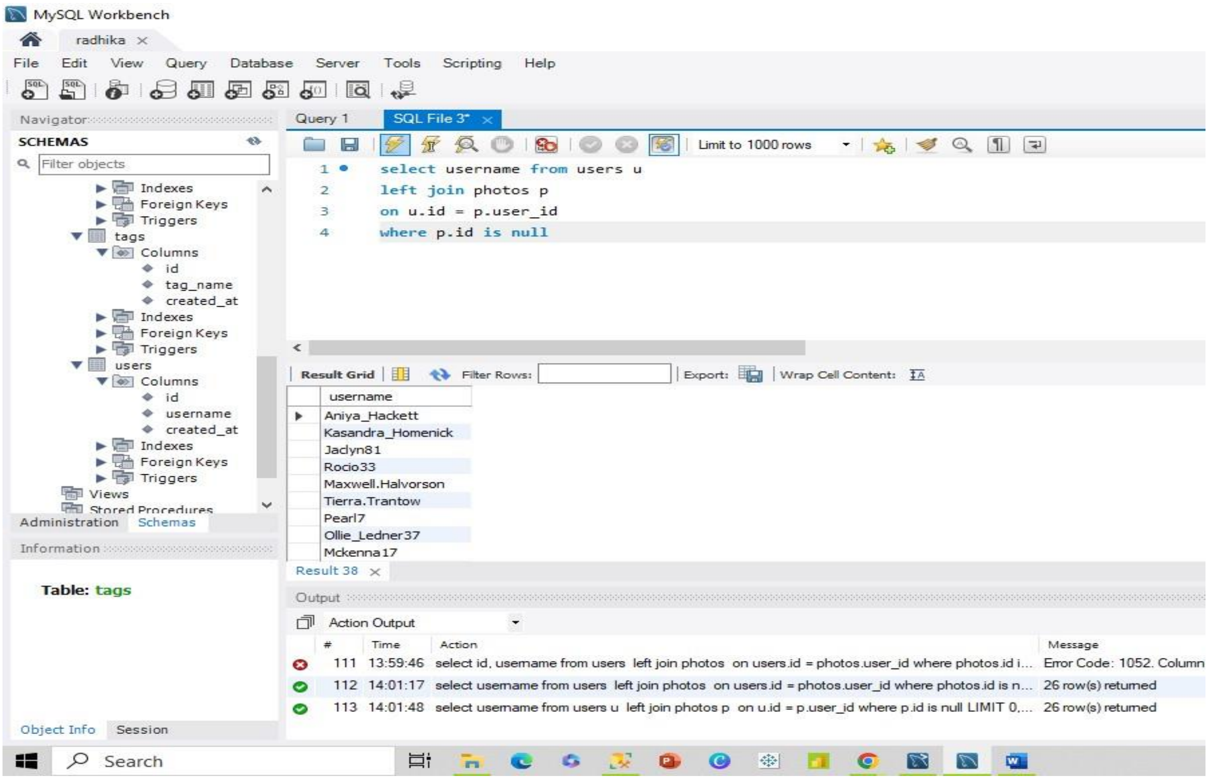
select username from users u
left join photos p
on u.id = p.user_id
where p.id is null

Query retrieves rows from users table where there is no corresponding entry in photos table. The left join displays only entries from left table "users" where there is no corresponding match in the right table "photos" by "where p.id is null" statement

"Where" is a clause in SQL used to filter records according to a condition.

All the users who have not posted a single post has been displayed

P is the alias given to photos, which we can use to address "photos" table anywhere in the query



```
select username from users u
left join photos p
on u.id = p.user_id
where p.id is null
```

username
Aniya_Hackett
Kasandra_Homenick
Jaclyn81
Rocio33
Maxwell.Halvorson
Tierra.Trantow
Pearl7
Ollie_Ledner37
Mckenna17

```
#    Time      Action                                                                                        Message
111  13:59:46  select id, username from users  left join photos  on users.id = photos.user_id where photos.id i...  Error Code: 1052. Column
112  14:01:17  select username from users  left join photos  on users.id = photos.user_id where photos.id is n...  26 row(s) returned
113  14:01:48  select username from users u  left join photos p  on u.id = p.user_id where p.id is null LIMIT 0,...  26 row(s) returned
```

Task 3(Contest winner declaration): user with a single photo having most likes

SQL Code:

select * from photos
join (select photo_id, count(likes.created_at) a from likes
group by 1
order by 2 desc) sub
on photos.id = sub.photo_id
order by sub.a desc

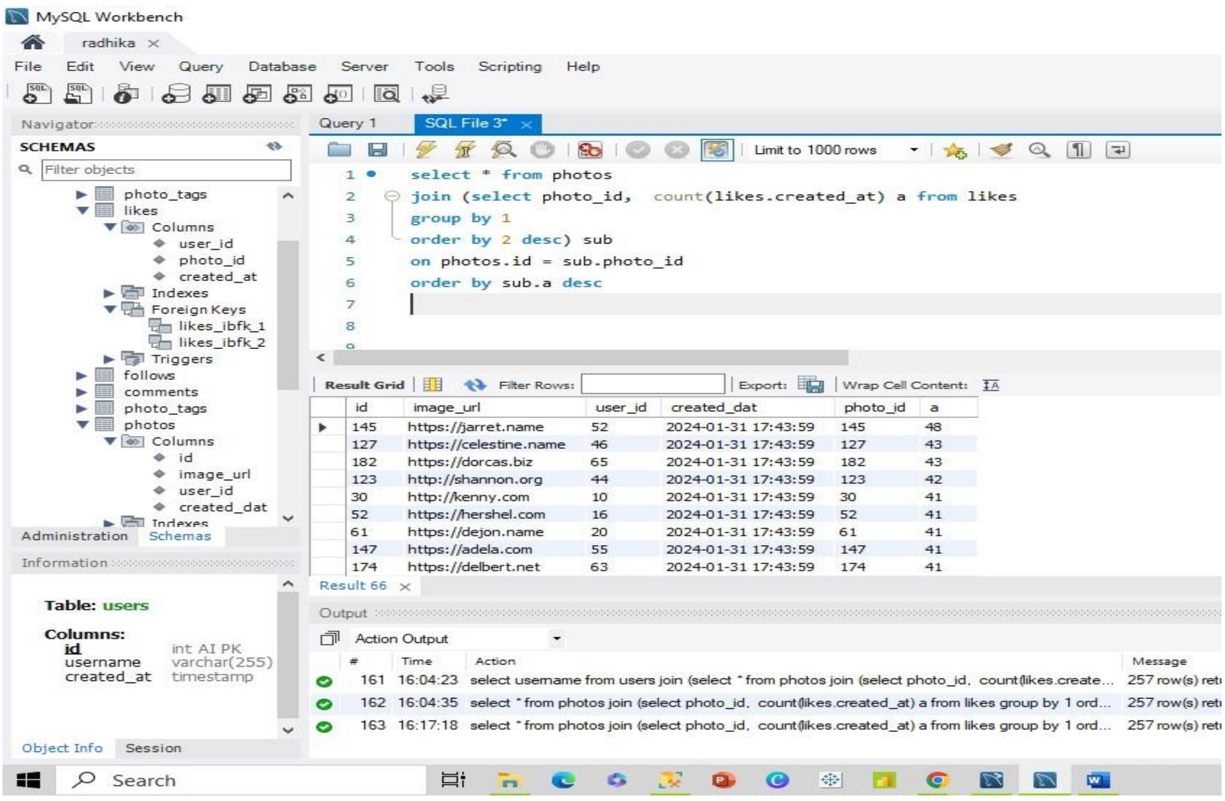
the query retrieves data from the "photos" table and sorts the results based on the number of likes each photo has received with the photo having the highest number of likes appearing first.

The subquery selects the "photo_id" column from the "likes" table and counts the number of likes for each photo. It groups the results by "photo_id" and orders them by the count of likes in descending order. The main query then joins the result of this subquery with the "photos" table based on the "photo_id" column.

User_id "52" has got 45 number of likes for his photo with the photo_id "145"

"Group by" is a clause in SQL used group a column into distinct values, it is most often used with aggregations to show one value per grouped field or combination of fields.

"Order by" sorts the result as per specified field, by default it sorts in ascending, "desc" is used for sorting in descending order.



```
select * from photos
join (select photo_id,  count(likes.created_at) a from likes
group by 1
order by 2 desc) sub
on photos.id = sub.photo_id
order by sub.a desc
```

	id	image_url	user_id	created_dat	photo_id	a
▶	145	https://jarret.name	52	2024-01-31 17:43:59	145	48
	127	https://celestine.name	46	2024-01-31 17:43:59	127	43
	182	https://dorcas.biz	65	2024-01-31 17:43:59	182	43
	123	http://shannon.org	44	2024-01-31 17:43:59	123	42
	30	http://kenny.com	10	2024-01-31 17:43:59	30	41
	52	https://hershel.com	16	2024-01-31 17:43:59	52	41
	61	https://dejon.name	20	2024-01-31 17:43:59	61	41
	147	https://adela.com	55	2024-01-31 17:43:59	147	41
	174	https://delbert.net	63	2024-01-31 17:43:59	174	41

Result 66

```
#    Time      Action                                                                                          Message
161  16:04:23  select username from users join (select * from photos join (select photo_id,  count(likes.create...  257 row(s) ret
162  16:04:35  select * from photos join (select photo_id,  count(likes.created_at) a from likes group by 1 ord...   257 row(s) ret
163  16:17:18  select * from photos join (select photo_id,  count(likes.created_at) a from likes group by 1 ord...   257 row(s) ret
```

Table: users

Columns:

id	int AI PK
username	varchar(255)
created_at	timestamp

Task 4(Hashtag Research): Most frequently used hashtag

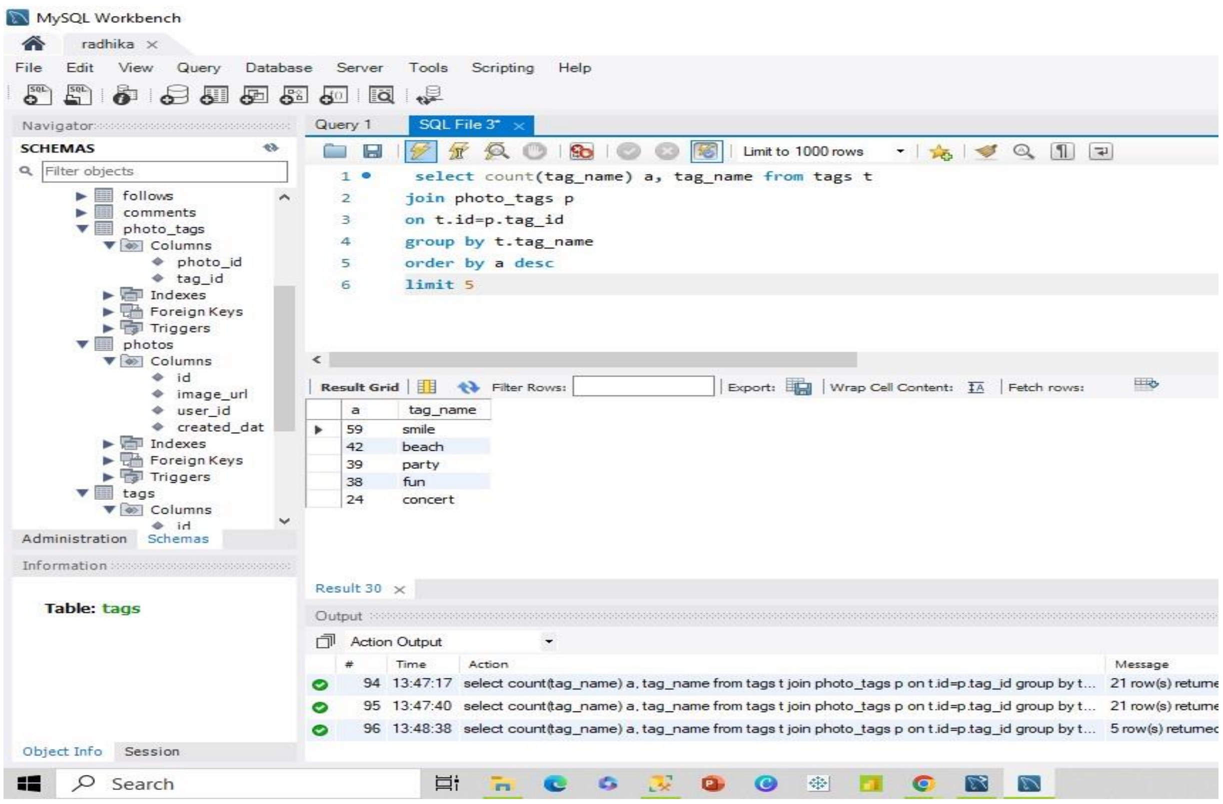
SQL Code:

select count(tag_name) a, tag_name from tags t
join photo_tags p on t.id=p.tag_id
group by t.tag_name
order by a desc
limit 5

The query retrieves the top 5 most frequently used tags in the dataset along with the count of their occurences, providing insights into popular topics or themes among users photo uploads

smile is the top used hashtag with count 59 and beach being 2 most frequently used hashtag with count 42.

Count is an aggregate function used in SQL to count no of rows as per group by statement



```
select count(tag_name) a, tag_name from tags t
join photo_tags p
on t.id=p.tag_id
group by t.tag_name
order by a desc
limit 5
```

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

```
Result 30
Output
Action Output
 #   Time      Action                                                                                          Message
 94  13:47:17  select count(tag_name) a, tag_name from tags t join photo_tags p on t.id=p.tag_id group by t...  21 row(s) returne
 95  13:47:40  select count(tag_name) a, tag_name from tags t join photo_tags p on t.id=p.tag_id group by t...  21 row(s) returne
 96  13:48:38  select count(tag_name) a, tag_name from tags t join photo_tags p on t.id=p.tag_id group by t...  5 row(s) returnec
```

Task 5(AD Campaign launch): Best day of the week to launch ads

SQL Code:

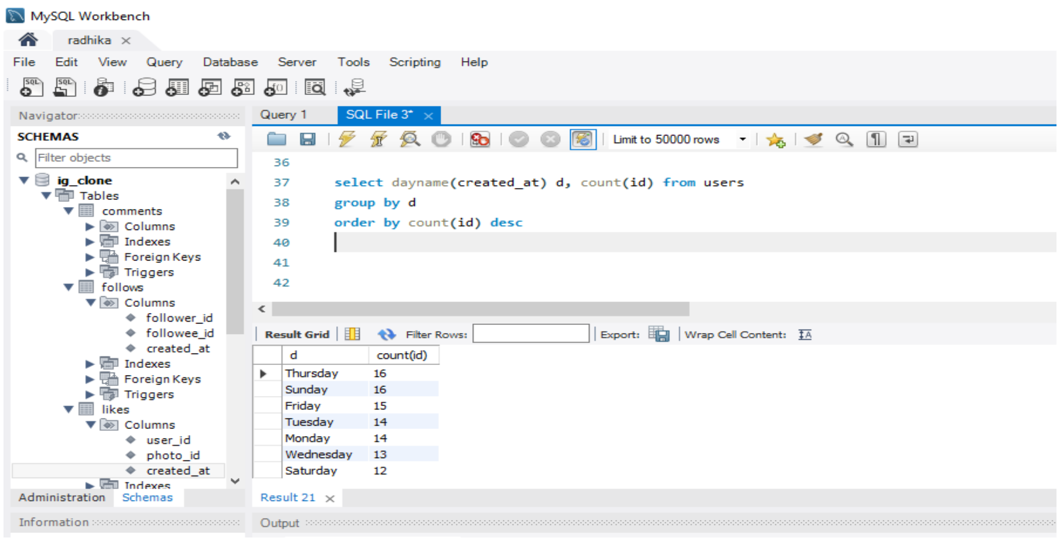
select dayname(created_at) d, count(id) from users
group by d
order by count(id) desc

query retrieves the weekday with count of users giving the distribution of user registration over a week of time.

dayname function gives the name of the weekday of the date given

(0-Monday,1-Tuesday, ………, 6-Sunday)

Thursday and Sunday are best days to launch ad campaign since registration on the platform is highest



```
select dayname(created_at) d, count(id) from users
group by d
order by count(id) desc
```

d	count(id)
Thursday	16
Sunday	16
Friday	15
Tuesday	14
Monday	14
Wednesday	13
Saturday	12

Result 21

B. Investor Metrics

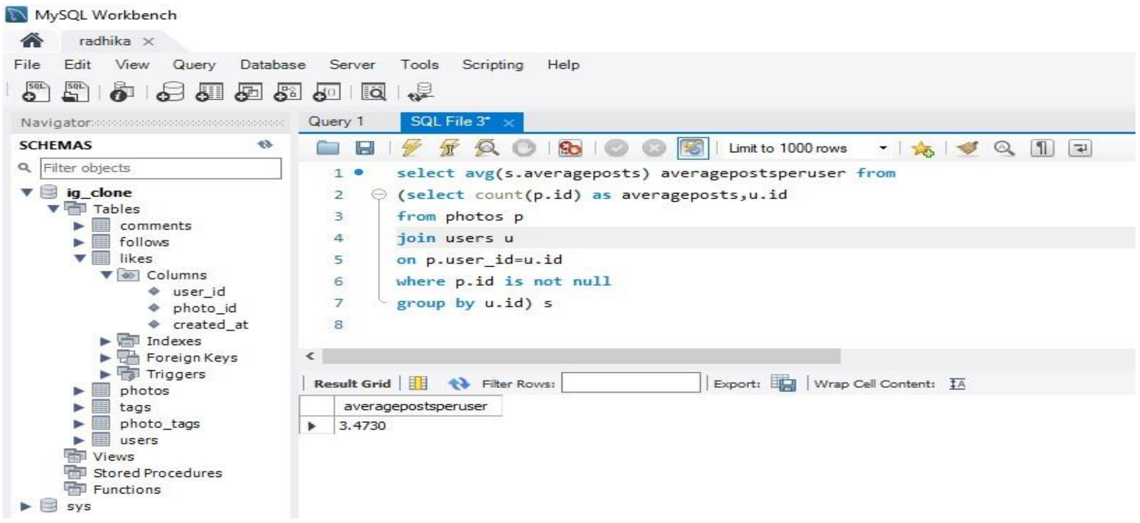
Task1(User engagement): average number of posts per user on Instagram

SQL Code:

select avg(s.averageposts) averagepostsperuser from
(select count(p.id) as averageposts,u.id from photos p
join users u
on p.user_id = u.id
group by u.id) s

the query calculates the average number of posts per user by counting the total number of posts for each user and then finding the average across all users. This provides insight into user engagement and activity levels on the platform.

Average posts per user is 3.4730



```
select avg(s.averageposts) averagepostsperuser from
(select count(p.id) as averageposts,u.id
from photos p
join users u
on p.user_id=u.id
where p.id is not null
group by u.id) s
```

```
averagepostsperuser
3.4730
```

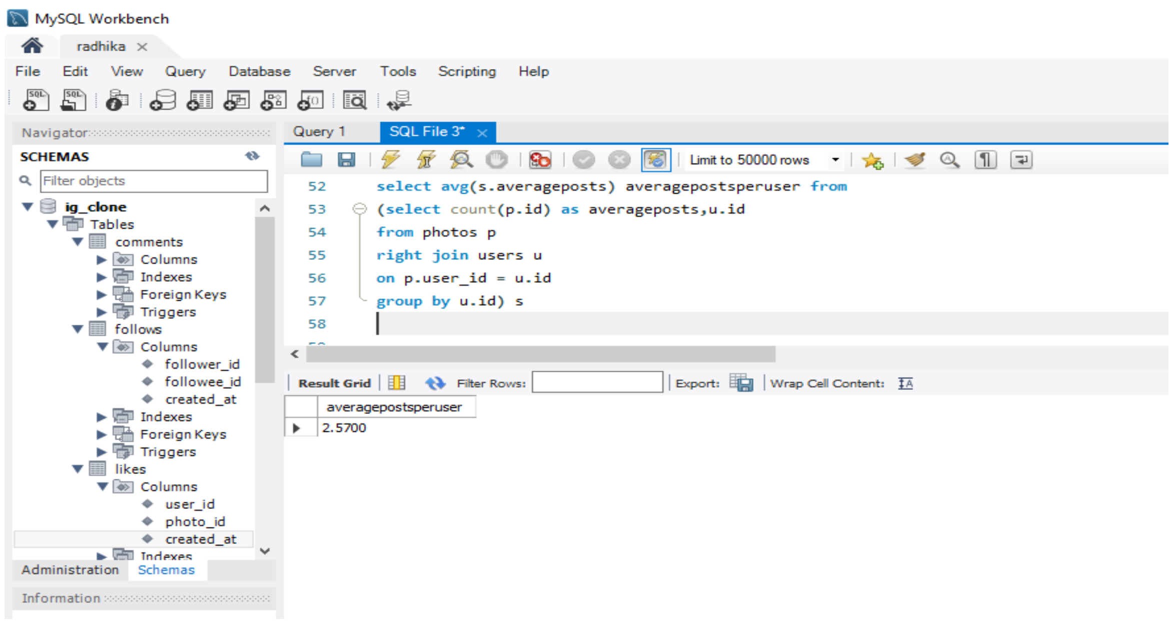
select avg(s.averageposts) averagepostsperuser from
(select count(p.id) as averageposts,u.id from photos p
right join users u
on p.user_id = u.id
group by u.id) s

the query calculates the average number of posts per user by counting the total number of posts for each user including null values that is users who have not posted any photos.

Both the queries calculate the same thing but the difference is with kind of joins being used. A simple **join** would only consider user id's that are present in both the tables whereas by using **right join** the user id's from the right table "users" is included even though there is no match for it in left table "photos". the match for these users id's will be null values in the result table. As a result

Average posts per user is 2.57

which corresponds to the "total number of photos divided by total number of users on Instagram" which actually gives us the average photos per user.**This shows the importance of understanding the uses of kinds of joins in SQL, how the results would be affected by simply altering the kind of joins we use.**

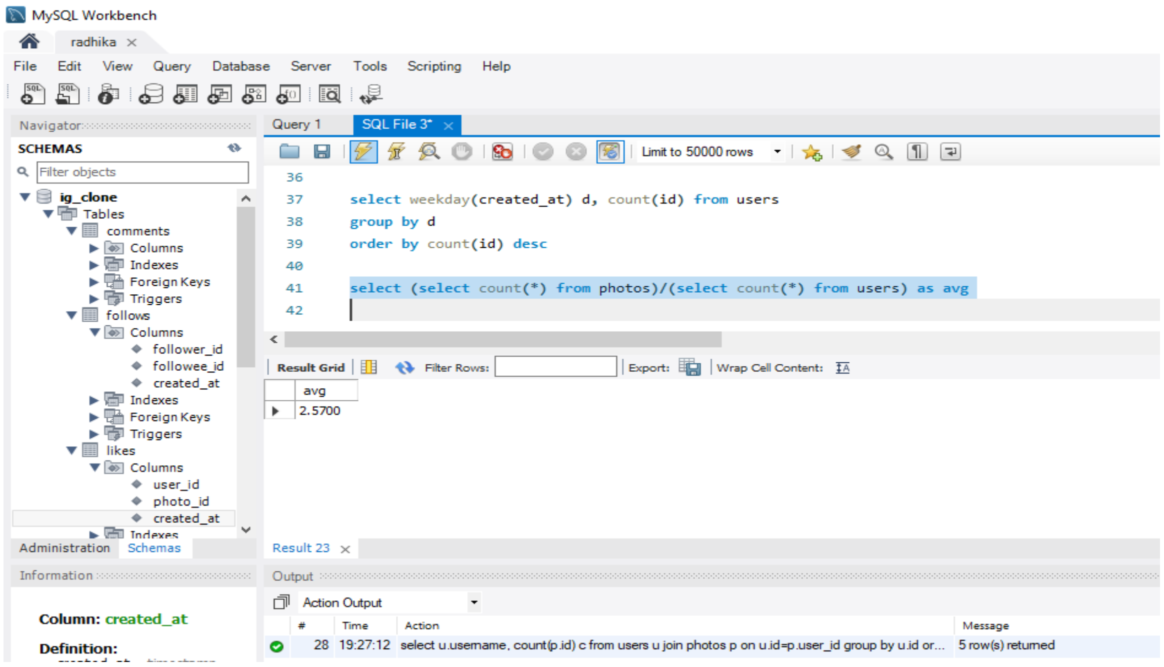


```
select avg(s.averageposts) averagepostsperuser from
(select count(p.id) as averageposts,u.id
from photos p
right join users u
on p.user_id = u.id
group by u.id) s
```

```
averagepostsperuser
2.5700
```

Below Query retrieves the “total number of photos divided by total number of users on Instagram”
result is 2.57

SQL Code:
select (select count(*) from photos)/(select count(*) from users) as avg
the result is equal to the previous query with the right join



```
select weekday(created_at) d, count(id) from users
group by d
order by count(id) desc

select (select count(*) from photos)/(select count(*) from users) as avg
```

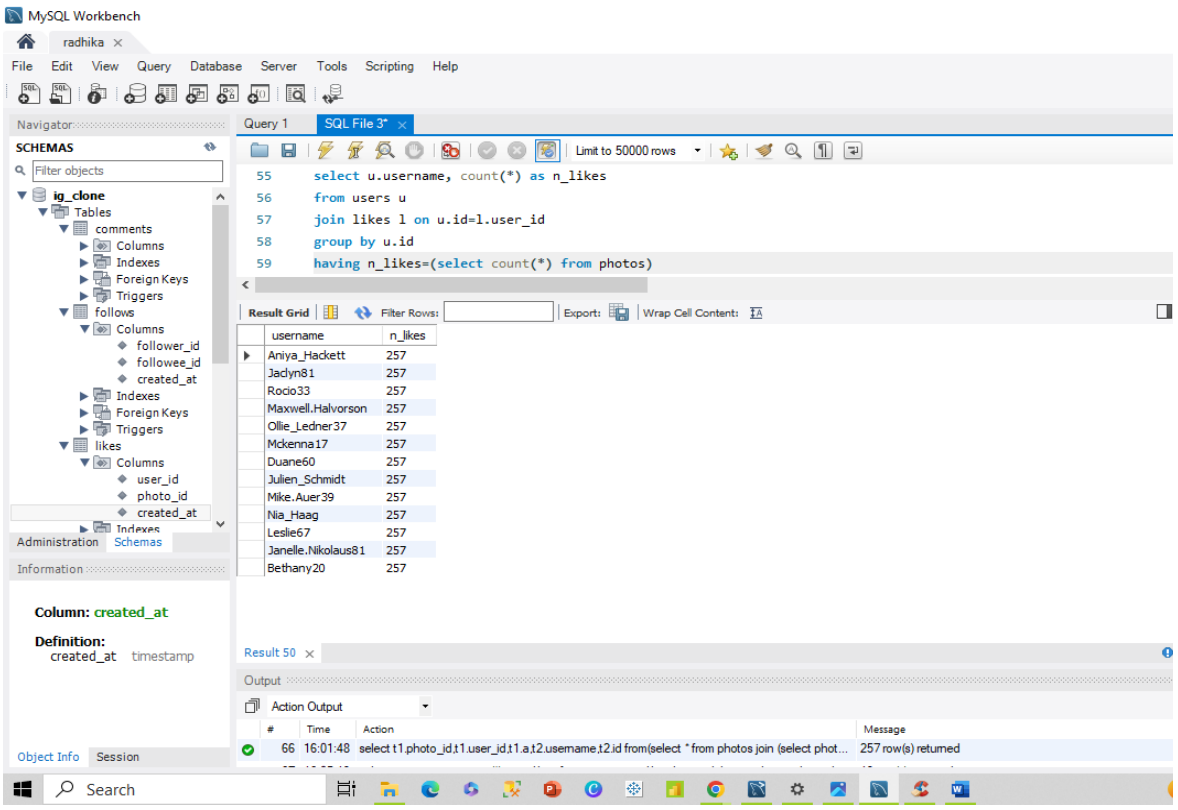
```
avg
2.5700
```

```
#   Time      Action                                                                                     Message
28  19:27:12  select u.username, count(p.id) c from users u join photos p on u.id=p.user_id group by u.id or...   5 row(s) returned
```

Task 2(Bots and fake account): user who have liked every photo on Instagram

SQL Code:

select u.username, count(*) as n_likes
from users u
join likes l on u.id=l.user_id
group by u.id
having n_likes = (select count(*) from photos)
The query identifies users who have liked every photo on the platform and calculates the number of likes for each users and comparing it to a subquery "total number of photo on the platform" which indicates that each photo has been liked by a particular user which could to be fake account



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

username	n_likes
Aniya_Hackett	257
Jaclyn81	257
Rocio33	257
Maxwell.Halvorson	257
Ollie_Ledner37	257
Mckenna17	257
Duane60	257
Julien_Schmidt	257
Mike.Auer39	257
Nia_Haag	257
Leslie67	257
Janelle.Nikolaus81	257
Bethany20	257

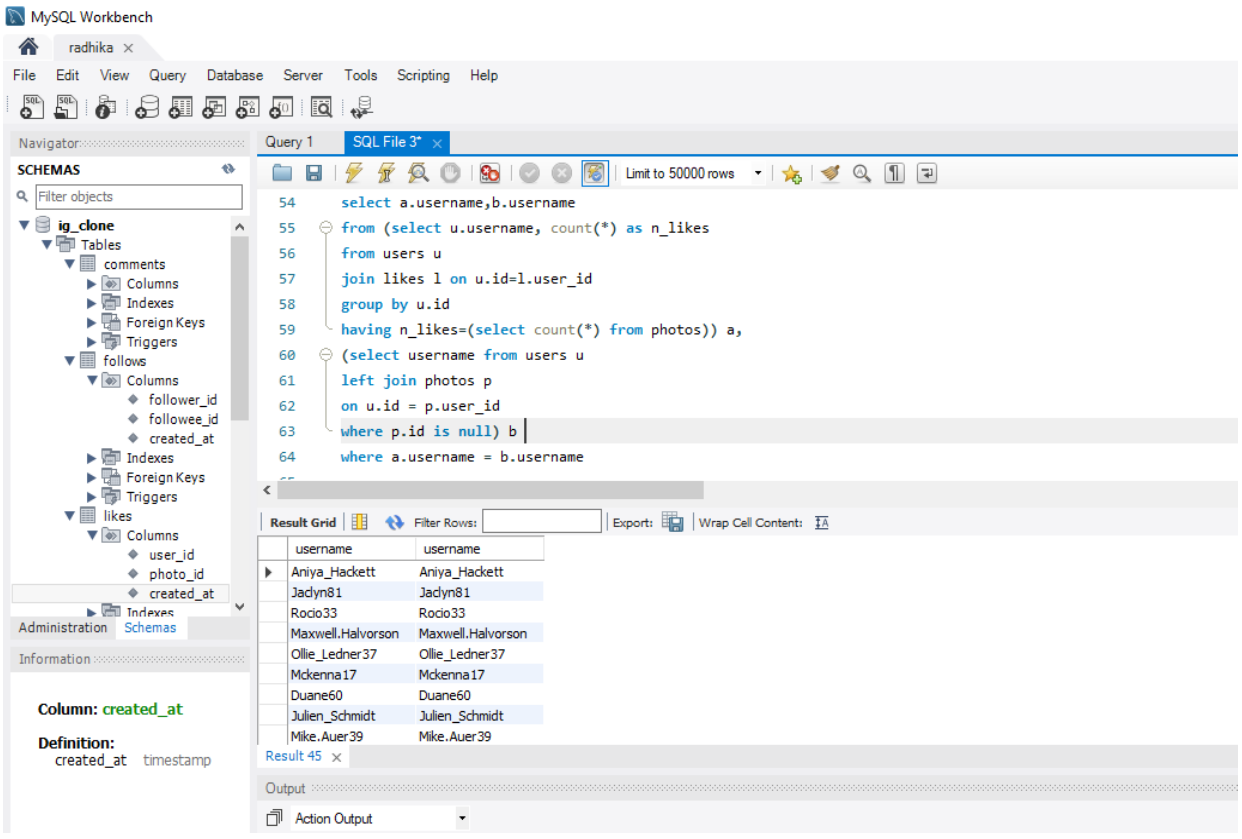
#	Time	Action	Message
66	16:01:48	select t1.photo_id,t1.user_id,t1.a,t2.username,t2.id from(select * from photos join (select phot...	257 row(s) returned

Column: created_at
Definition:
created_at timestamp

Possibility of a fake account could also be users who have not posted a single photo on the platform, by combining the results of the above query and 2nd query from marketing analysis, we can get the common people to have not posted a single photo and have liked every photo on the platform making it a possible fake account. This can be achieved by using both queries as subqueries "a", "b" and combining them through where statement equating it to identical column "username" from both queries.

Below query retrieves pairs of usernames where users have liked every photo on the platform and also have not uploaded any photo themselves showing the bot behaviour or possible fake account. Both subqueries are the individual queries of "user who have not posted a single photo" and "user who have liked every photo on the platform.

```
select a.username,b.username
from(select u.username, count(*) as n_likes
from users u
join likes l on u.id=l.user_id
group by u.id
having n_likes=(select count(*) from photos)) a,
(select username from users u
left join photos p
on u.id = p.user_id
where p.id is null) b
where a.username = b.username
```



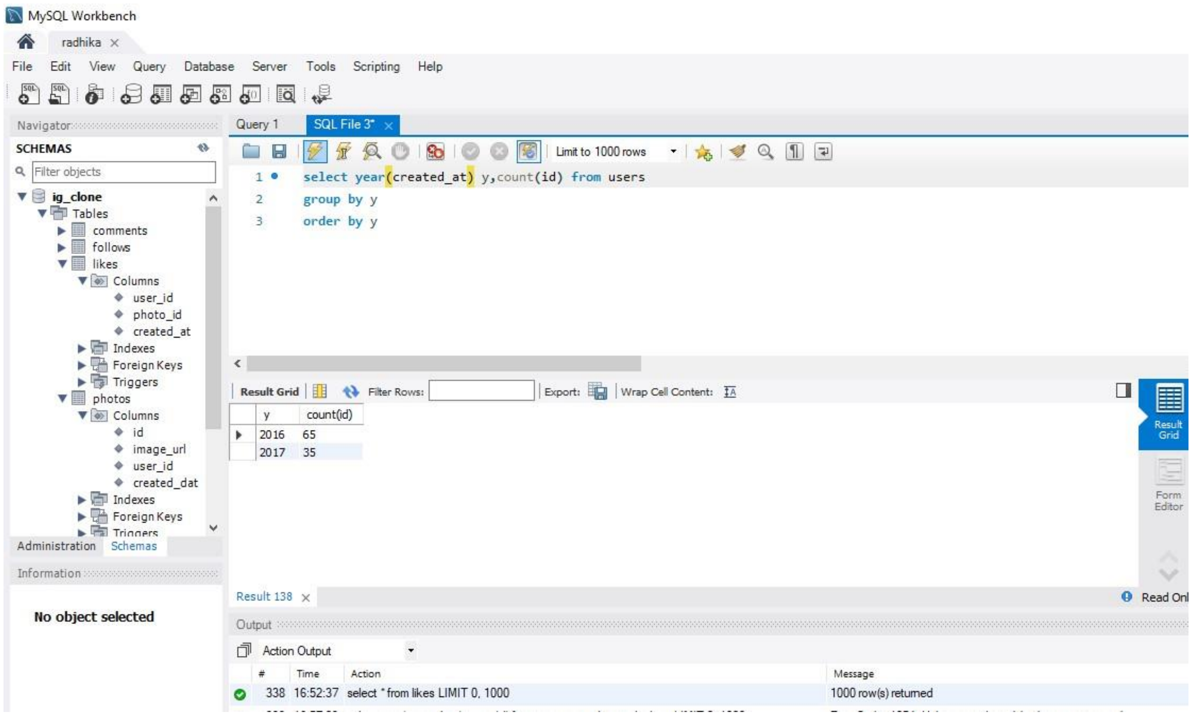
```
select a.username,b.username
from (select u.username, count(*) as n_likes
from users u
join likes l on u.id=l.user_id
group by u.id
having n_likes=(select count(*) from photos)) a,
(select username from users u
left join photos p
on u.id = p.user_id
where p.id is null) b
where a.username = b.username
```

username	username
Aniya_Hackett	Aniya_Hackett
Jaclyn81	Jaclyn81
Rocio33	Rocio33
Maxwell.Halvorson	Maxwell.Halvorson
Ollie_Ledner37	Ollie_Ledner37
Mckenna17	Mckenna17
Duane60	Duane60
Julien_Schmidt	Julien_Schmidt
Mike.Auer39	Mike.Auer39

Other Key Insights

1] No of users registered over the years

SQL Code:
select year(created_at) y, count(id) c from users
group by y
order by y
query provides insights into the distribution of user registrations across different years, helping identify trends or patterns in user sign-ups over time.
We can see there has been decrease in the no of sign-ups over the year which is almost half of the previous year.



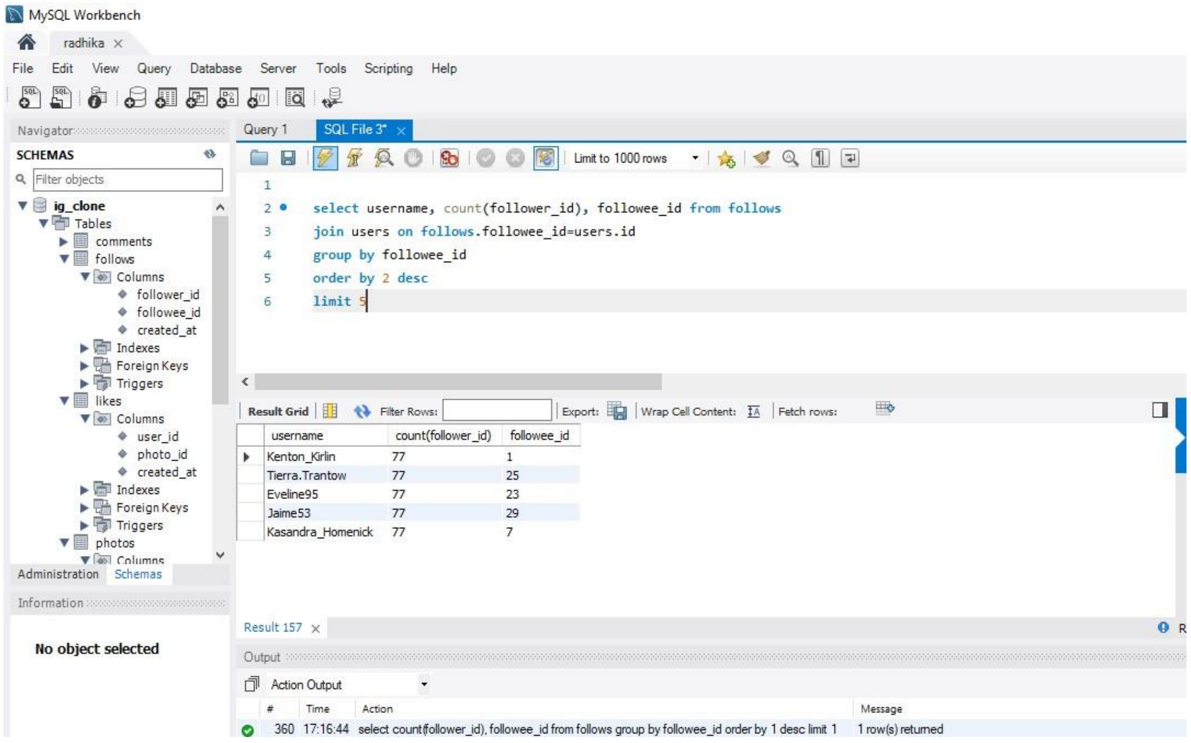
```
select year(created_at) y,count(id) from users
group by y
order by y
```

y	count(id)
2016	65
2017	35

```
#    Time      Action                            Message
338  16:52:37  select * from likes LIMIT 0, 1000  1000 row(s) returned
```

2] top 5 users with more follower count

SQL Code:
select username, count(follower_id), followee_id from follows
join users on follows.followee_id=users.id
group by followee_id
order by 2 desc
limit 5
query retrieves the usernames of users who have the most followers, along with the count of their followers
this could indicate the user engagement on the platform that they might be posting consistently and grown their follower account.



```
select username, count(follower_id), followee_id from follows
join users on follows.followee_id=users.id
group by followee_id
order by 2 desc
limit 5
```

username	count(follower_id)	followee_id
Kenton_Kirlin	77	1
Tierra.Trantow	77	25
Eveline95	77	23
Jaime53	77	29
Kasandra_Homenick	77	7

```
#    Time      Action                                                                          Message
360  17:16:44  select count(follower_id), followee_id from follows group by followee_id order by 1 desc limit 1   1 row(s) returned
```

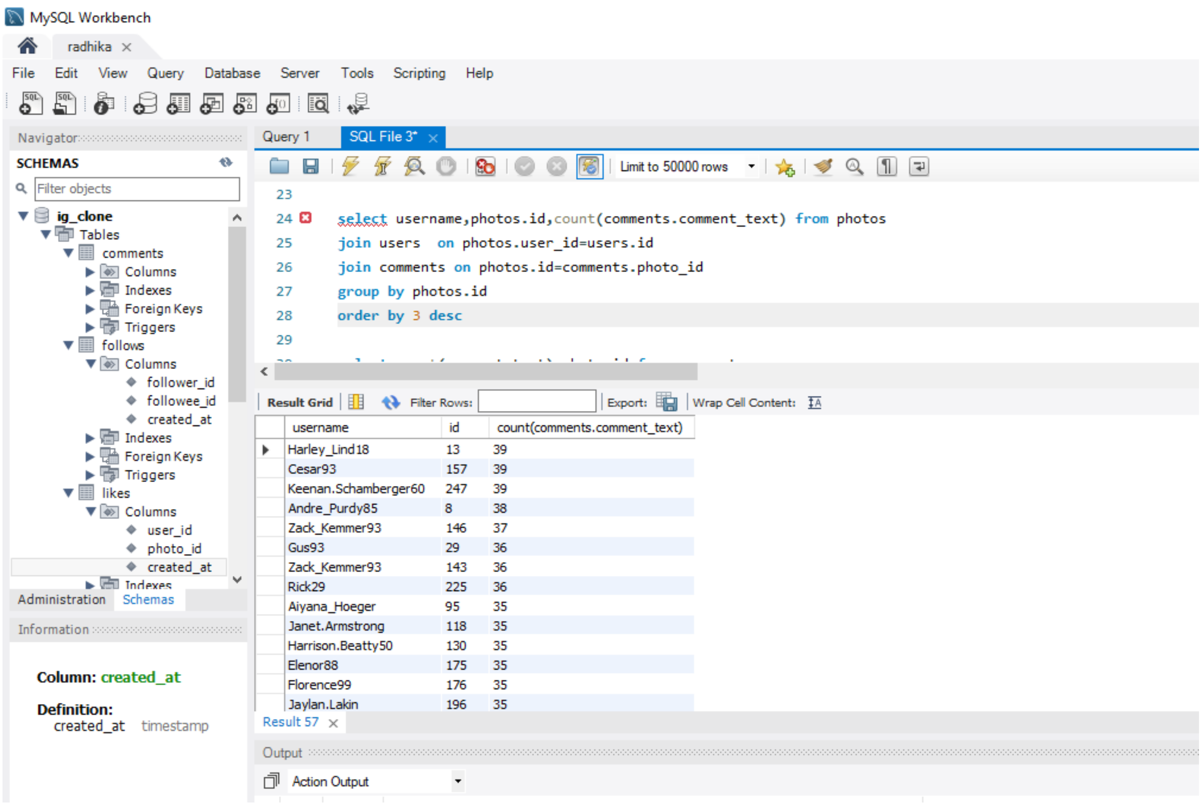
3] user with a single photo with most comments

SQL Code:

select username, photos.id, count(comments.comment_text) from photos
join users on photos.user_id=users.id
join comments on photos.id=comments.photo_id
group by photos.id
order by 3 desc

query provides insight into the popularity of photos based on the number of comments they receive, with the results showing the usernames of users who uploaded the photos aong with the photo IDs and the count of comments for each photo.

This could indicate the users are more engaging with the viewers.



```
select username,photos.id,count(comments.comment_text) from photos
join users  on photos.user_id=users.id
join comments on photos.id=comments.photo_id
group by photos.id
order by 3 desc
```

username	id	count(comments.comment_text)
Harley_Lind18	13	39
Cesar93	157	39
Keenan.Schamberger60	247	39
Andre_Purdy85	8	38
Zack_Kemmer93	146	37
Gus93	29	36
Zack_Kemmer93	143	36
Rick29	225	36
Aiyana_Hoeger	95	35
Janet.Armstrong	118	35
Harrison.Beatty50	130	35
Elenor88	175	35
Florence99	176	35
Jaylan.Lakin	196	35

4] users who has posted more no of photos

SQL Code:

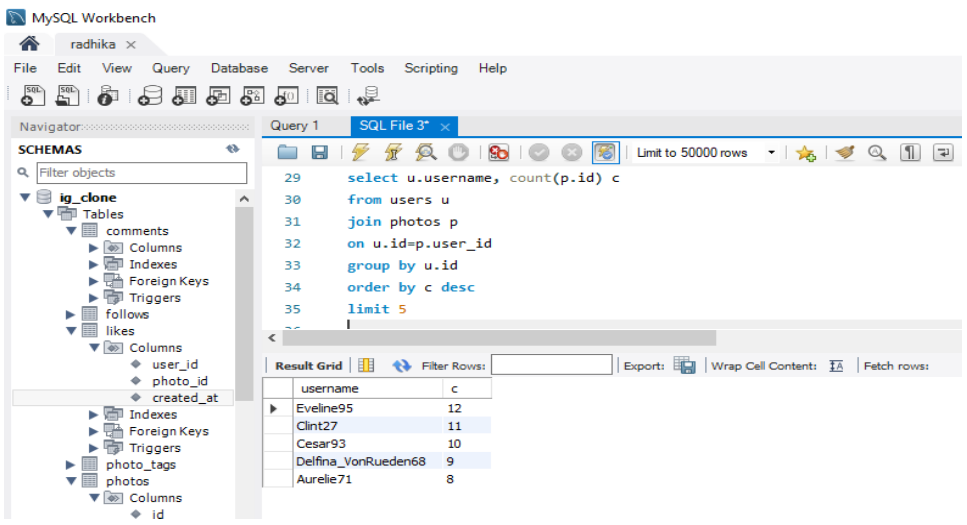
select u.username, count(p.id) c
from users u
join photos p
on u.id=p.user_id
group by u.id
order by c desc
limit 5

the query identifies the top 5 users with the highest number of uploaded photos, providing insights into user activity and contribution to the platform
I wanted to see if these users had highest follower count as well. To achieve this i joined the results from both queries "top five users who have posted most photos" and "top users with highest followers" and found out the users where among the top users with the highest follower count as well. Having highest no of followers and posting the most photos means the user is active and consistent on the platform.

SQL Code:

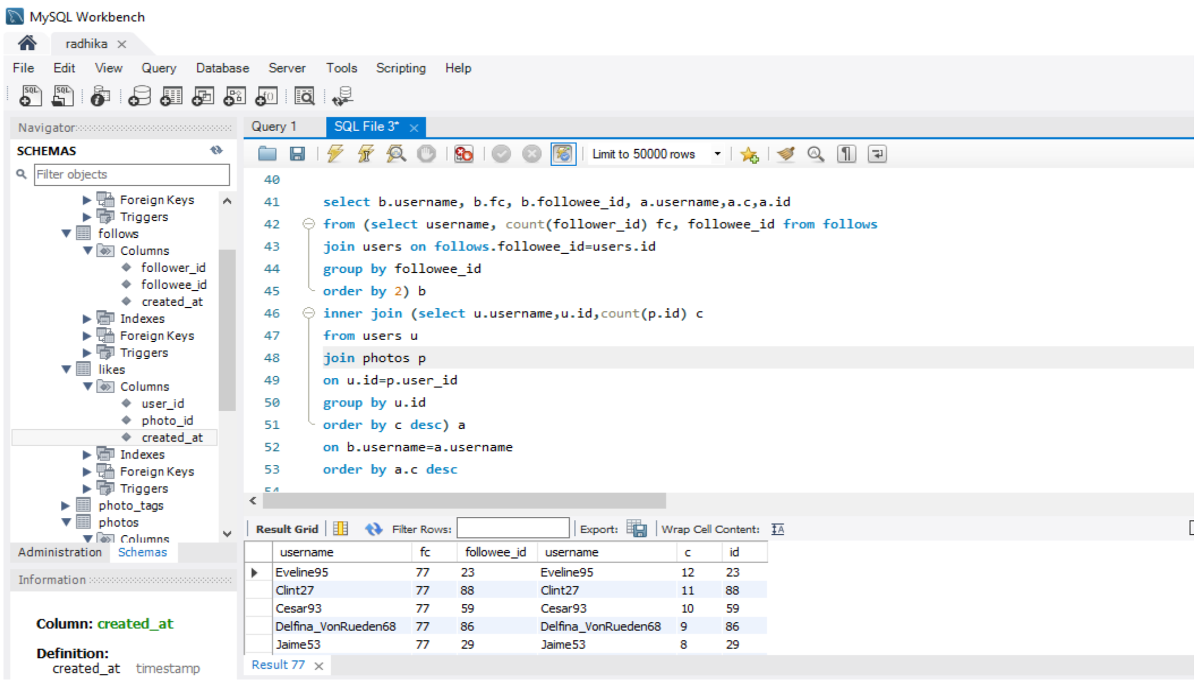
select b.username, b.fc, b.followee_id, a.username,a.c,a.id
from (select username, count(follower_id) fc, followee_id from follows
join users on follows.followee_id=users.id
group by followee_id

```
order by 2) b
inner join (select u.username,u.id,count(p.id) c
from users u
join photos p
on u.id=p.user_id
group by u.id
order by c desc) a
on b.username=a.username
order by a.c desc
```



```
select u.username, count(p.id) c
from users u
join photos p
on u.id=p.user_id
group by u.id
order by c desc
limit 5
```

username	c
Eveline95	12
Clint27	11
Cesar93	10
Delfina_VonRueden68	9
Aurelie71	8



```
select b.username, b.fc, b.followee_id, a.username,a.c,a.id
from (select username, count(follower_id) fc, followee_id from follows
join users on follows.followee_id=users.id
group by followee_id
order by 2) b
inner join (select u.username,u.id,count(p.id) c
from users u
join photos p
on u.id=p.user_id
group by u.id
order by c desc) a
on b.username=a.username
order by a.c desc
```

username	fc	followee_id	username	c	id
Eveline95	77	23	Eveline95	12	23
Clint27	77	88	Clint27	11	88
Cesar93	77	59	Cesar93	10	59
Delfina_VonRueden68	77	86	Delfina_VonRueden68	9	86
Jaime53	77	29	Jaime53	8	29

Column: created_at

Definition:
created_at timestamp

Result:

Working on this project involving Instagram data analysis has enhanced my SQL query writing skills particularly in aggregating data from multiple tables, usage of different kinds of joins, how to write subqueries, joining tables, understanding how tables are referenced to one another and so on.

It has helped me develop my analytical skills on how to look for patterns and trends in the data, how to look for possible outcomes and more by conducting different kind of analysis.

By analysing the number of photos uploaded by each user, we can identify top contributors and understand the level of user engagement with content creation on the platform.

Examining the count of likes, comments and followers for photos and users can provide insights into the popularity and engagement levels of content. Users with higher engagement metrics may have more influence and reach on the platform.

Analysing the frequency of hashtags, user interactions, and posting behaviours can help identify trends and patterns in user-generated content. Understanding popular themes and topics can inform content strategy and marketing efforts.

Tracking user registrations and sign-up trends over time can provide insights into user growth patterns and platform adoption. Monitoring user retention rates and engagement levels can help identify opportunities for improving user experience and retention

Overall, these insights can inform strategic decision-making for product development, content strategy, marketing campaigns and user engagement initiatives on the Instagram platform. Understanding user behaviour and engagement trends is essential for optimizing the platform's features and driving sustainable growth and user satisfaction.