

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

The screenshot shows a SQL IDE window titled "Instagram User Analytics". The query editor contains the following SQL code:

```
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
91 * LIMIT 5;
```

The results grid shows the following data:

id	username	created_at
1	Keriton_Kirin	2017-02-16 18:22:11
2	Andre_Purdy85	2017-04-02 17:11:21
3	Harley_Land18	2017-02-21 11:12:33
4	Aniya_Bogard63	2016-08-13 01:28:43
5	Aniya_Hackett	2016-12-07 01:04:39
6	Travon_Waters	2017-04-20 13:26:14
7	Kassandra_Homewick	2016-12-12 06:30:09
8	Tabitha_Schamberger11	2016-08-20 02:19:46
9	Guerr3	2016-06-24 19:36:31
10	Presley_McClure	2016-08-07 16:25:49
11	Justina_Geivler27	2017-09-04 16:32:16
12	Dereck65	2017-01-19 01:34:14
13	Alexandro35	2017-03-29 17:09:02
14	Nekron1	2017-03-08 23:29:16

The output pane shows the following actions:

#	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

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.

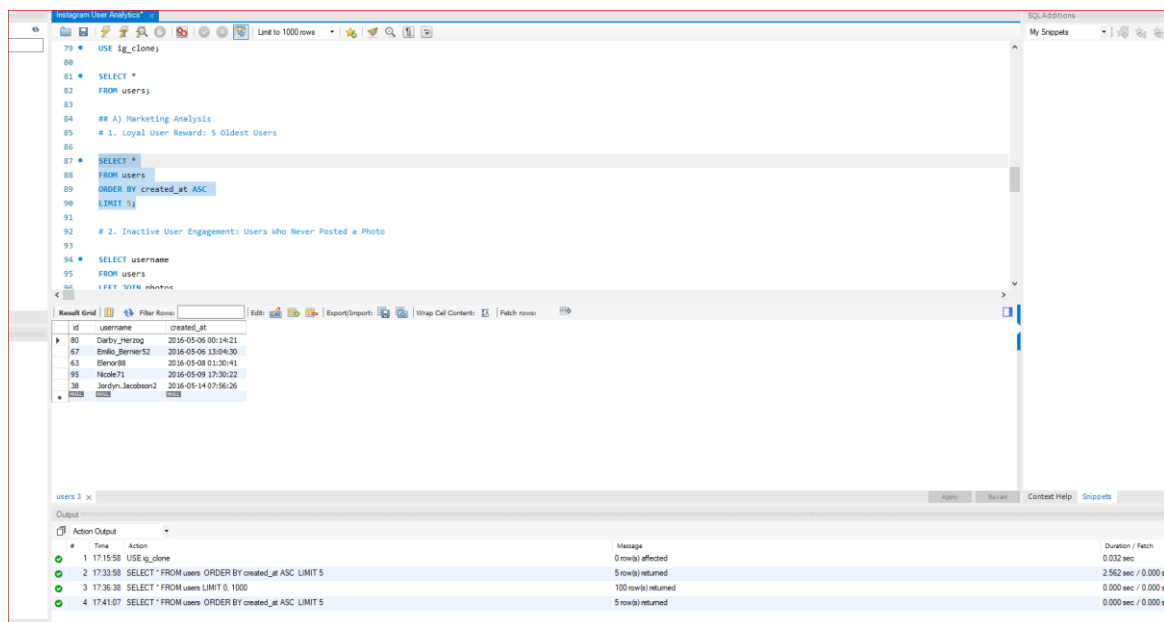
SQL Code:

```
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 a SQL IDE window titled "Instagram User Analytics". The query editor contains the following SQL code:

```
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;  
91  
92 # 2. Inactive User Engagement: Users who Never Posted a Photo  
93  
94 SELECT username  
95 FROM users  
96 WHERE NOT EXISTS (SELECT 1 FROM posts WHERE user_id = users.id);
```

The results grid shows the output of the query:

#	id	username	created_at
1	80	Darby_Zerog	2016-05-08 00:14:21
2	47	Emile_Bernier52	2016-05-08 13:04:30
3	43	Elmor88	2016-05-08 01:30:41
4	95	Nicole71	2016-05-09 17:30:22
5	38	Jordyn_Jacobson2	2016-05-14 07:36:26

The bottom panel shows the execution log:

#	Time	Action	Message	Duration / Patch
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 a SQL IDE with three queries and their results. The first query, 'Loyal User Reward: 5 Oldest Users', returns 5 usernames. The second query, 'Inactive User Engagement: Users who Never Posted a Photo', returns 25 usernames. The third query, 'Contest Winner: Most Likes On a Single Photo', returns 1 photo ID and its total likes.

```
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
103 FROM photos
104 INNER JOIN likes
105 ON likes.photo_id = photos.id
106 ORDER BY total DESC
107 LIMIT 1;
```

Result Grid

username
Anya_Hackett
Kassandra_Jomnick
Jaclyn81
Roddy33
Maxwell_Mahorson
Tierra_Tranbow
Pearl7
Oliver_Ledner37
McKenzie17
David_Daniel47
Morgan_Kassulke
Linnea59
Duane60
Liam_Kohene6

Output

#	Time	Action	Message	Duration / Feedback
1	17:15:50	USE iq_clone	0 row(s) affected	0.032 sec
2	17:33:50	SELECT * FROM users ORDER BY created_at ASC LIMIT 5	5 row(s) returned	2.562 sec / 0.0
3	17:36:30	SELECT * FROM users LIMIT 0, 1000	100 row(s) returned	0.000 sec / 0.0
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	25 row(s) returned	0.141 sec / 0.0

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:

```
197 ON users.id = photos.user_id
198 WHERE photos.id IS NULL;
199
200 # 3. Contest Winner: Post Likes On a Single Photo
201
202 SELECT username, photos.id, photos.image_url, COUNT(likes.user_id) AS total
203 FROM photos
204 inner join likes
205 on likes.photo_id = photos.id
206 inner join users
207 on photos.user_id = users.id
208 group by photos.id
209 order by total desc
210 limit 1 ;
211
212 # 4. Hashtag Research: Top 5 Post Common Hashtags
213
214 SELECT post_tag_name, COUNT(*) AS hashtag_count
```

username	id	image_url	total
Zack_Jammer93	145	https://i.pinimg.com/48	48

Time	Action	Message	Duration / Fetch
2 17:33:58	SELECT * FROM users ORDER BY created_at ASC LIMIT 5	5 row(s) returned	2.952 sec / 0.000 se
3 17:36:38	SELECT * FROM users LIMIT 0, 1000	100 row(s) returned	0.000 sec / 0.000 se
4 17:41:07	SELECT * FROM users ORDER BY created_at ASC LIMIT 5	5 row(s) returned	0.000 sec / 0.000 se
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 se
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 group by photos.id order by total desc limit 1 ;	1 row(s) returned	0.844 sec / 0.000 se

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.

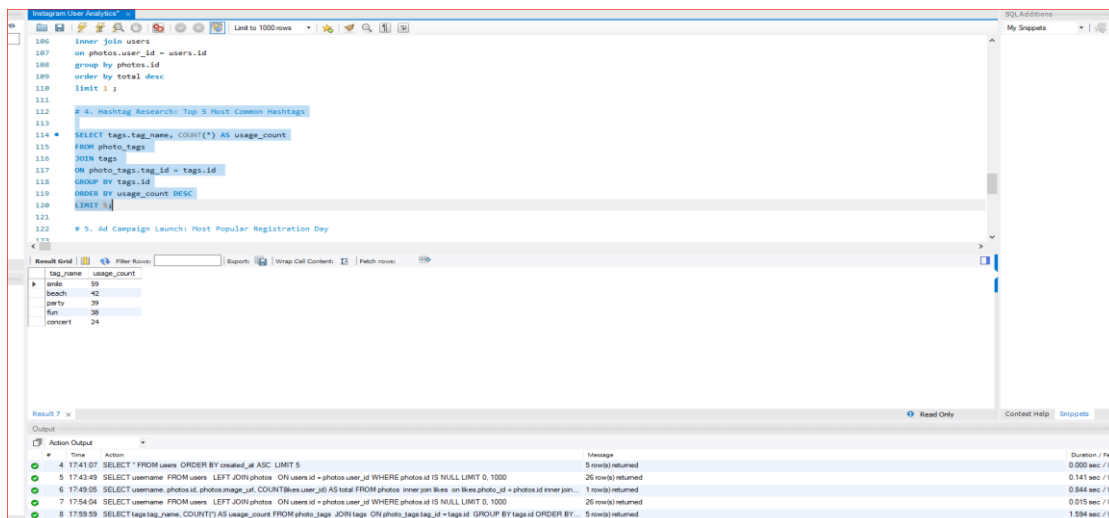
SQL Code:

```
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 a SQL IDE window titled "Instagram User Analytics". The query editor contains the following SQL code:

```
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 Most Common Hashtags
113
114 * SELECT tags.tag_name, COUNT(*) AS usage_count
115 FROM photo_tags
116 JOIN tags
117 ON photo_tags.tag_id = tags.id
118 GROUP BY tags.id
119 ORDER BY usage_count DESC
120 LIMIT 5;
121
122 # 5. Ad Campaign Launch: Most Popular Registration Day
123
```

The results table shows the top 5 most common hashtags:

tag_name	usage_count
#love	59
#beach	42
#party	39
#fun	38
#concert	24

The bottom section of the screenshot shows the "Output" pane with a table of actions and their results:

#	Action	Message	Duration / Rows
4	17:41:57 SELECT * FROM users ORDER BY created_at ASC LIMIT 5	5 rows(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	0.147 sec / 0.0
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...	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 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...	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 IDE interface with a query editor on the left and a results pane on the right. The query editor contains the following SQL code:

```
115 FROM photo_tags
116 JOIN tags
117 ON photo_tags.tag_id = tags.id
118 GROUP BY tags.id
119 ORDER BY usage_count DESC
120 LIMIT 5;
121
122 # 5. Ad Campaign Launch: Most Popular Registration Day
123
124 SELECT DAYNAME(created_at) AS day, COUNT(*) AS total
125 FROM users
126 GROUP BY day
127 ORDER BY total DESC
128 LIMIT 1;
129
130
131 # 8) Investor Metrics
```

The results pane shows the output of the query, which is a single row representing the most popular registration day:

day	total
Thursday	35

The bottom of the screenshot shows the 'Output' pane with a log of SQL actions and their execution details, including timestamps, messages, and durations.

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:

The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
122 # 5. Ad Campaign Launch: Most Popular Registration Day
123
124 SELECT DAYNAME(created_at) AS day, COUNT(*) AS total
125 FROM users
126 GROUP BY day
127 ORDER BY total DESC
128 LIMIT 1;
129
130 # 6) Investor Metrics
131
132 # 1. User Engagement: Average Posts Per User
133
134 SELECT
135 (SELECT COUNT(*) FROM photos) / (SELECT COUNT(*) FROM users) AS Average;
136
137 # 2. Bots & Fake Accounts: Users who Like Every Photo
138
```

The results pane shows the output of the query, which is a single row with the value 2.5700.

Result Grid
Average
2.5700

The bottom pane shows the execution log with the following entries:

Time	Action	Message	Duration / Fetch
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 WHERE photos.id IS NULL LIMIT 0, 1000	1 row(s) returned	0.044 sec / 0.000
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	25 row(s) returned	0.015 sec / 0.000
8:17:59.59	SELECT tag_name, COUNT(*) AS usage_count FROM photo_tags JOIN tags ON photo_tag_id = tags.id GROUP BY tag_name ORDER BY usage_count DESC LIMIT 1	5 row(s) returned	1.594 sec / 0.000
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	0.782 sec / 0.000
10:18:25.14	SELECT (SELECT COUNT(*) FROM photos) / (SELECT COUNT(*) FROM users) AS Average LIMIT 0, 1000	1 row(s) returned	0.125 sec / 0.000

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.

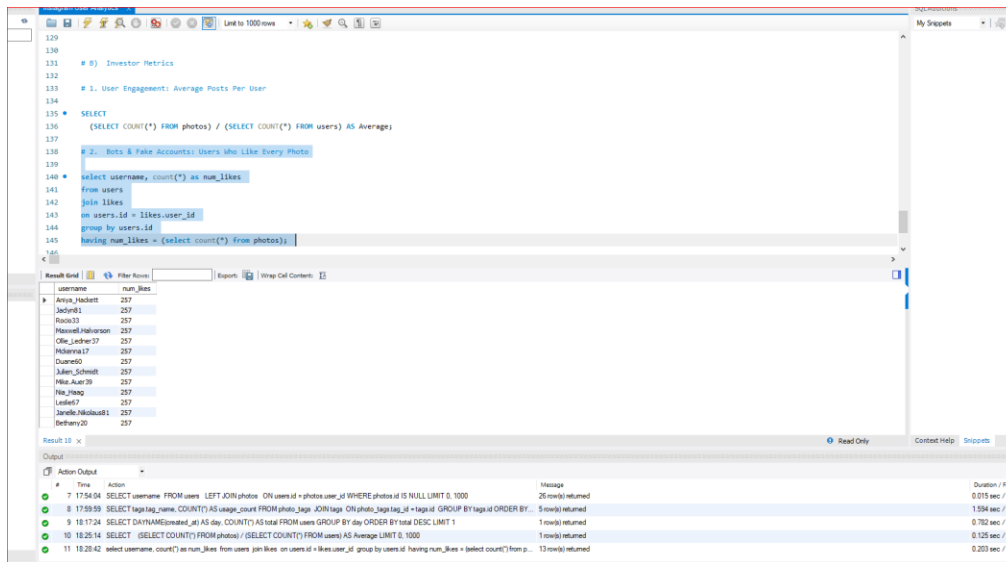
SQL Code:

```
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 SQL IDE with a query editor at the top and a results grid at the bottom. The query is a multi-part SQL statement. The results grid displays a table with two columns: 'username' and 'num_likes'. The table contains 10 rows of data, including usernames like 'Anya_Hackett', 'JackieB', 'Rosa33', 'Hassan_Habibson', 'Ola_Jedrej37', 'Mikenna17', 'Quarrel', 'Julian_Schmidt', 'Mike_Auer39', 'Hag_Thag', 'Lestel67', and 'Bethany20', all with a value of 257 in the 'num_likes' column. The bottom panel shows the execution log with timestamps and messages for each query step.

```
129
130
131 # B) Investor Metrics
132
133 # 1. User Engagement: Average Posts Per User
134
135 SELECT
136 (SELECT COUNT(*) FROM photos) / (SELECT COUNT(*) FROM users) AS Average;
137
138 # 2. Bots & Fake Accounts: Users who Like Every Photo
139
140 select username, count(*) as num_likes
141 from users
142 join likes
143 on users.id = likes.user_id
144 group by users.id
145 having num_likes = (select count(*) from photos);
```

username	num_likes
Anya_Hackett	257
JackieB	257
Rosa33	257
Hassan_Habibson	257
Ola_Jedrej37	257
Mikenna17	257
Quarrel	257
Julian_Schmidt	257
Mike_Auer39	257
Hag_Thag	257
Lestel67	257
Bethany20	257

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/1Fr42v1rvfD3k4FDfRjk4QzBrqAeRPLOL?usp=sharing>