

INSTAGRAM USER ANALYTICS

Project Description:

The Instagram Analytics project seeks to better understand how Instagram users engage with the platform. The primary purpose is to obtain insights into user behavior, identify areas for improvement, and design effective marketing campaigns.

In basic terms, we want to understand how users interact with the platform, identify what could improve their experience, and develop innovative strategies to promote the platform in order to attract new users or encourage existing ones to be more active. It's about knowing what people are executing, finding methods to improve things for them, and devising smart ways to spread the word about the platform.

Approach:

i. Loyal User Reward:

Identified the five oldest users on Instagram based on their registration date.

SQL Query:

```
1 • SELECT *
2   FROM users
3   ORDER BY created_at ASC;
4 • SELECT DISTINCT username, created_at FROM users ORDER BY created_at ASC limit 5;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:	Fetch rows:
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				

Explanation:

Query 1:

This query retrieves all columns (*) from the 'users' table. The 'ORDER BY created_at ASC' clause sorts the result set in ascending order based on the 'created_at' column.

This query is intended to retrieve all user entries from the 'users' table and display them chronologically depending on their creation date.

Query 2:

This query selects distinct combinations of username and 'created_at' from the 'users' table. The 'ORDER BY created_at ASC' clause sorts the result set in ascending order based on the 'created_at' column. The 'LIMIT 5' clause restricts the output to only the top 5 rows.

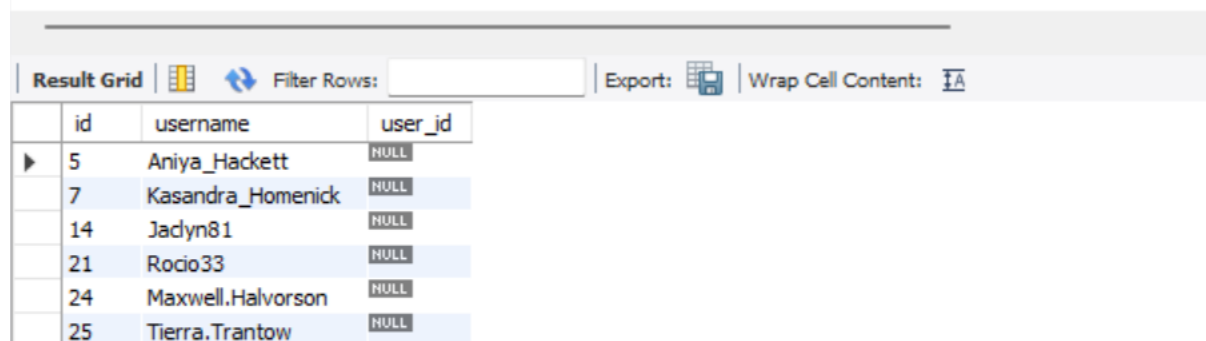
This query seeks to extract a list of unique usernames from the 'users' table, along with their creation dates. The findings are limited to the first five users who created their accounts.

ii. Inactive User Engagement:

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

SQL Query:

```
1 • select distinct users.id, users.username, photos.user_id from users
2   left join photos
3   on users.id = photos.user_id
4   where photos.user_id is null;
5
```



The screenshot shows a database interface with a query result grid. The grid has columns for 'id', 'username', and 'user_id'. The 'user_id' column contains 'NULL' for all rows, indicating that these users have not posted any photos. The interface includes a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'.

	id	username	user_id
▶	5	Aniya_Hackett	NULL
	7	Kasandra_Homenick	NULL
	14	Jadyn81	NULL
	21	Rocio33	NULL
	24	Maxwell.Halvorson	NULL
	25	Tierra.Trantow	NULL

Explanation:

The query is using a 'LEFT JOIN' between the 'users' and 'photos' tables based on the condition that the 'id' of the 'users' table matches the 'user_id' of the 'photos' table.

The 'SELECT' statement retrieves distinct combinations of columns: 'users.id', 'users.username', and 'photos.user_id'.

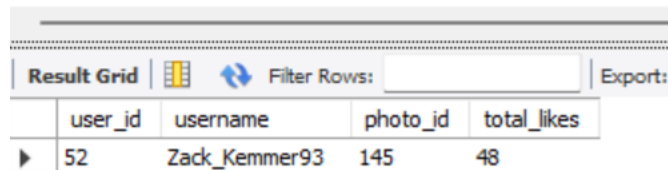
The 'WHERE photos.user_id IS NULL' filters the results to include only those records where there is no match in the 'photos' table, meaning the user has not posted any photos.

iii. Contest Winner Declaration:

Identified the user who has the highest number of likes on a single photo, declared them as the contest winner.

SQL Query:

```
1 • SELECT
2     u.id AS user_id,
3     u.username,
4     p.id AS photo_id,
5     COUNT(l.user_id) AS total_likes
6 FROM
7     users u,
8     photos p
9 LEFT JOIN
10    likes l ON p.id = l.photo_id
11 WHERE
12     u.id = p.user_id
13 GROUP BY
14     u.id, u.username, p.id
15 ORDER BY
16     total_likes DESC
```



The screenshot shows a SQL query editor with a query window and a results window. The query window contains the SQL query. The results window shows a table with 5 columns: user_id, username, photo_id, and total_likes. The first row of data shows user_id 52, username Zack_Kemmer93, photo_id 145, and total_likes 48.

	user_id	username	photo_id	total_likes
▶	52	Zack_Kemmer93	145	48

Explanation:

- The query retrieves information about users and their photos, focusing on the user with the most likes on a single photo.
- The SELECT statement extracts the user ID (u.id), username (u.username), photo ID (p.id), and the count of likes for that photo (COUNT(l.user_id)).
- The FROM clause specifies two tables: users (aliased as u) and photos (aliased as p).
- The LEFT JOIN connects the photos table with the likes table based on the condition that the id of the photo (p.id) matches the photo_id in the likes table (l alias).
- The WHERE clause filters the results to include only records where the user ID in the users table (u.id) matches the user ID in the photos table (p.user_id).

- The GROUP BY clause groups the result set by user ID, username, and photo ID, and the COUNT(l.user_id) provides the total number of likes for each user's photo.
- The ORDER BY total_likes DESC orders the result set in descending order based on the total number of likes.
- The LIMIT 1 restricts the result set to only one row, representing the user with the most likes on a single photo.
- This query aims to find the user with the highest number of likes on a single photo, providing details about that user, the photo, and the total number of likes.

iv. Hashtag Research:

Identified the top 5 most popular tags on the platform based on the number of times each tag has been associated with photos.

SQL Query:

```

1 • SELECT t.id, t.tag_name, COUNT(pt.tag_id) AS tag_count
2 FROM tags t, photo_tags pt
3 WHERE t.id = pt.tag_id
4 GROUP BY t.id, t.tag_name
5 ORDER BY tag_count DESC
6 LIMIT 5;
7

```

Result Grid			
Filter Rows: <input type="text"/>			
Export: Wrap Cell Content:			
	id	tag_name	tag_count
▶	21	smile	59
	20	beach	42
	17	party	39
	13	fun	38
	18	concert	24

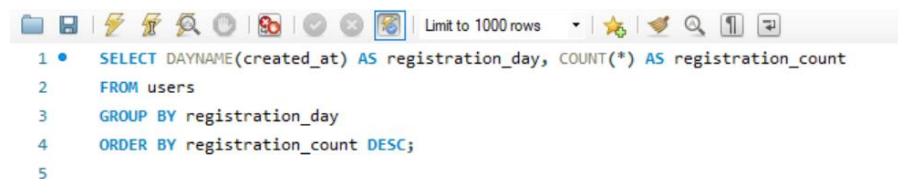
Explanation:

- The SELECT statement includes:
t.id: Tag ID.
t.tag_name: Tag name.
COUNT(pt.tag_id) AS tag_count: Total count of photos associated with each tag.
- The FROM clause specifies two tables: tags (aliased as t) and photo_tags (aliased as pt).
- The WHERE clause defines the condition for joining the tables: t.id = pt.tag_id, ensuring that only matching tag IDs are considered.
- The GROUP BY clause groups the result set by tag ID and tag name, allowing the COUNT function to aggregate the number of times each tag appears.
- The ORDER BY tag_count DESC orders the result set in descending order based on the tag count, ensuring that the most popular tags come first.
- The LIMIT 5 clause restricts the output to the top 5 tags with the highest tag counts.
- The purpose of this query is to identify and display the top 5 most popular tags on the platform based on the number of times they have been associated with photos.

v. Ad Campaign Launch:

Identified the total number of registration on each day of the week and found the best day for Ad Campaigns.

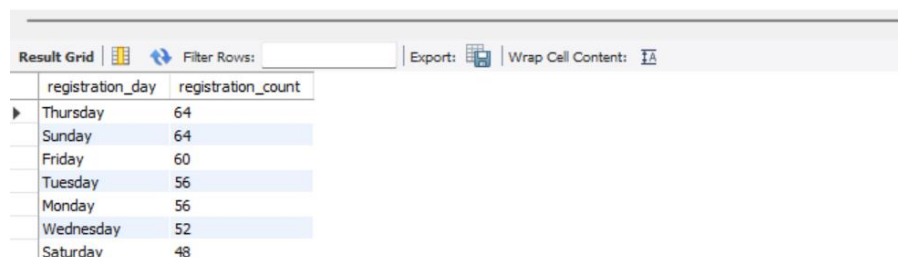
SQL Query:



```

1 • SELECT DAYNAME(created_at) AS registration_day, COUNT(*) AS registration_count
2 FROM users
3 GROUP BY registration_day
4 ORDER BY registration_count DESC;
5

```



registration_day	registration_count
Thursday	64
Sunday	64
Friday	60
Tuesday	56
Monday	56
Wednesday	52
Saturday	48

Explanation:

- The SELECT statement includes:
DAYNAME('created_at') AS 'registration_day': Extracts the day of the week from the 'created_at' column and aliases it as 'registration_day'.
COUNT(*) AS 'registration_count': Counts the number of user registrations for each day and aliases it as 'registration_count'.
- The FROM clause specifies the users table.
- The GROUP BY clause groups the result set by the day of the week ('registration_day'), allowing the COUNT function to aggregate the number of registrations for each day.
- The ORDER BY 'registration_count' DESC orders the result set in descending order based on the registration count, ensuring that the day with the highest registration count appears first.
- The purpose of this query is to understand the distribution of user registrations throughout the week. It helps identify which days have the highest and lowest registration activity.

vi. User Engagement:

Identified the total number of photos, total number of users, and the average number of photos per user on the platform.

SQL Query:

```
1 • SELECT AVG(posts_per_user) AS average_posts_per_user
2 FROM (
3     SELECT user_id, COUNT(*) AS posts_per_user
4     FROM photos
5     GROUP BY user_id
6 ) AS user_post_counts;
7 • SELECT COUNT(*) AS total_photos,
8     COUNT(DISTINCT user_id) AS total_users,
9     COUNT(*) / COUNT(DISTINCT user_id) AS photos_per_user
10 FROM photos;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

average_posts_per_user
6.9459

Result Grid			
Filter Rows: <input type="text"/>			
	total_photos	total_users	photos_per_user
▶	514	74	6.9459

Explanation:

Query 1:

- This query calculates the average number of posts per user on the platform.
- The inner subquery counts the number of posts (COUNT(*) AS posts_per_user) for each user (user_id) in the photos table, grouping the results by user.
- The outer query then calculates the average of the posts per user using the AVG function

Query 2:

- This query provides aggregated statistics about the total number of photos, total number of users, and the average number of photos per user on the platform.
- COUNT(*) AS total_photos calculates the total number of photos.
- COUNT(DISTINCT user_id) AS total_users calculates the total number of unique users.
- COUNT(*) / COUNT(DISTINCT user_id) AS photos_per_user calculates the average number of photos per user.

The first query specifically calculates the average number of posts per user, while the second query presents aggregated statistics about the overall photo activity on the platform.

vii. Bots & Fake Accounts:

Identified users (potential bots) who have liked 90% of photos on the site, as this is not typically possible for a normal user.

SQL Query:

```
1 • SELECT user_id
2 FROM (
3     SELECT user_id, COUNT(DISTINCT photo_id) AS liked_photos
4     FROM likes
5     GROUP BY user_id
6 ) AS user_like_counts
7 WHERE liked_photos > 0.9 * (SELECT COUNT(*) FROM photos);
8
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user_id

Explanation:

- The inner subquery:
Counts the number of distinct photos each user has liked (COUNT(DISTINCT photo_id) AS liked_photos).
Groups the results by user_id.
- The outer query:
Retrieves user_id from the inner subquery.
Filters users based on the condition liked_photos > 0.9 * (SELECT COUNT(*) FROM photos).
- Execution Steps:
Count the number of distinct photos liked by each user.
Calculate the total count of photos on the platform using (SELECT COUNT(*) FROM photos).
Identify users whose liked photos exceed 90% of the total photo count.

Tech-Stack Used:

- Database Management System: MySQL
- SQL Query Tool: MySQL Workbench
- MySQL was chosen for its reliability, performance, and simplicity of use.
MySQL Workbench provides an easy-to-use interface for querying and viewing SQL queries.

Insights:

- i. Loyal User Reward:**
Recognize and reward the platform's most persistent users based on their involvement.
Gain insights into user behavior over time by differentiating between consistent and infrequent users.
- ii. Inactive User Engagement:**
Identify users who have never posted, allowing for targeted re-engagement.
Recognize the reasons that contribute to user inactivity and create solutions to solve them.
- iii. Contest Winner Declaration:**
Contests' success can be measured by identifying winners based on likes.
Analyze content performance to uncover popular and engaging content.
- iv. Hashtag Research:**
Determine popular hashtags to gain a better understanding of current user interests.
Improve content categorization by using popular tags and subjects.
- v. Ad Campaign Launch:**
Determine the most effective day to run an ad campaign based on user registration habits.
Ad timing should be optimized for optimum exposure and user interaction.
- vi. User Engagement:**
Calculate the average number of postings per user, which provides information on user activity.
Determine differences in posting behavior across different user categories.
- vii. Bots and Fake accounts:**
SQL queries can be used to identify possible bots or bogus accounts.
Improve platform integrity by identifying and eliminating bogus accounts.

Results:

SQL Insights:

- Learned how to use SELECT command to retrieve specific data from tables.
- WHERE clause is used to filter data based on specified conditions.
- I learned how to group data using functions like COUNT, AVG, and GROUP BY.
- Understood how to join multiple tables to combine relevant data for analysis.
- CTEs were used to provide named result sets for complex queries, improving readability.

- Developed the ability to interpret business needs into SQL queries in order to make informed decisions.