THE CATHOLIC UNIVERSITY OF EASTERN AFRICA

CONSECRATE THEM IN THE TRUTH

SOCIALLYTIC - SOCIAL MEDIA ANALYTICS PLATFORM

<u>MEMBERS</u>

1. Beatrice Wamboi – 1052049
2. Vivian Ndung’u – 1049495
3. Zuena Kiplagat – 1049086
4. Abigail Kagoiza – 1052096
5. Abigail Kirimi – 1049507

ASSIGNMENT SUBMITTED ON 24TH NOVEMBER, 2024

INTRODUCTION

Overview

The Social Media Analysis Platform is designed to aggregate, analyze, and visualize data from various social media platforms. By leveraging advanced data analytics, the platform aims to provide insights into user behavior, trending topics, and sentiment analysis.

Rationale

With the exponential growth of social media, organizations need tools to understand and leverage user-generated content effectively. This platform addresses the demand for actionable insights that can drive marketing strategies, product development, and customer engagement.

Objectives

To develop efficient methods to extract data from various social media platforms (e.g., Twitter, Instagram, Facebook).

To design and implement a robust database schema to store the cleaned and pre-processed data.

To perform data analysis and generate reports on user engagement, sentiment, and trends.

To provide a user-friendly interface for querying and visualizing data.

To create interactive visualizations to communicate insights effectively.

ENTITY-RELATIONSHIP DIAGRAM

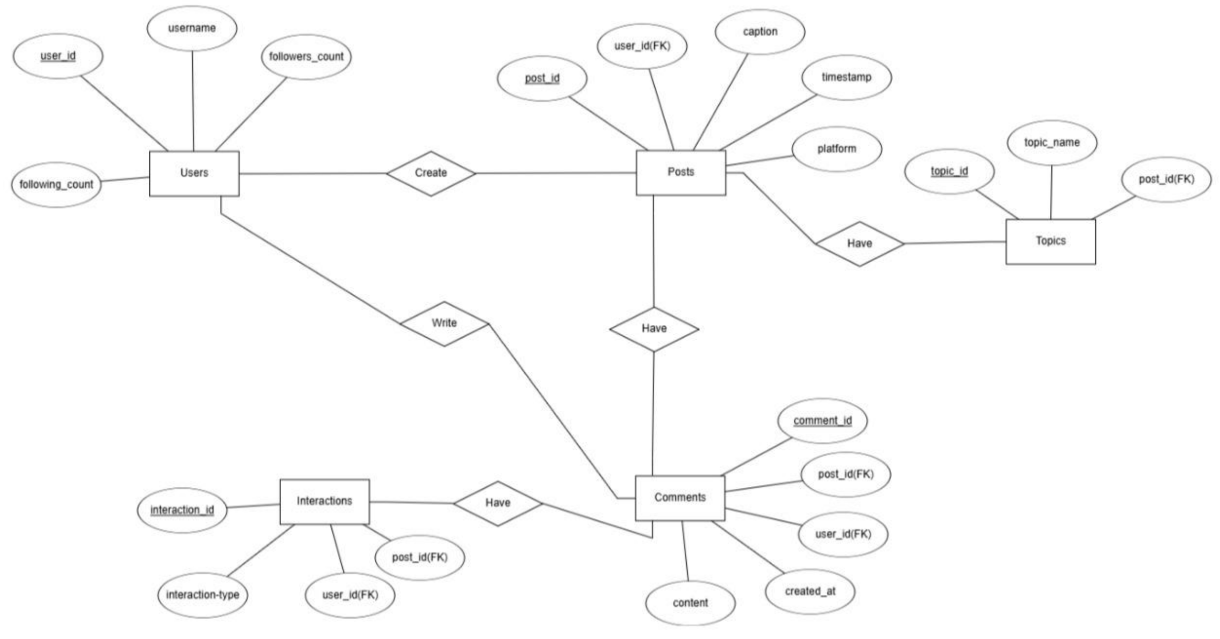


TABLE STRUCTURE

Posts

- post_id (Primary Key)
- user_id (Foreign Key)
- text
- timestamp
- platform

Users

- user_id (Primary Key)
- username
- location
- followers_count
- following_count

Topics

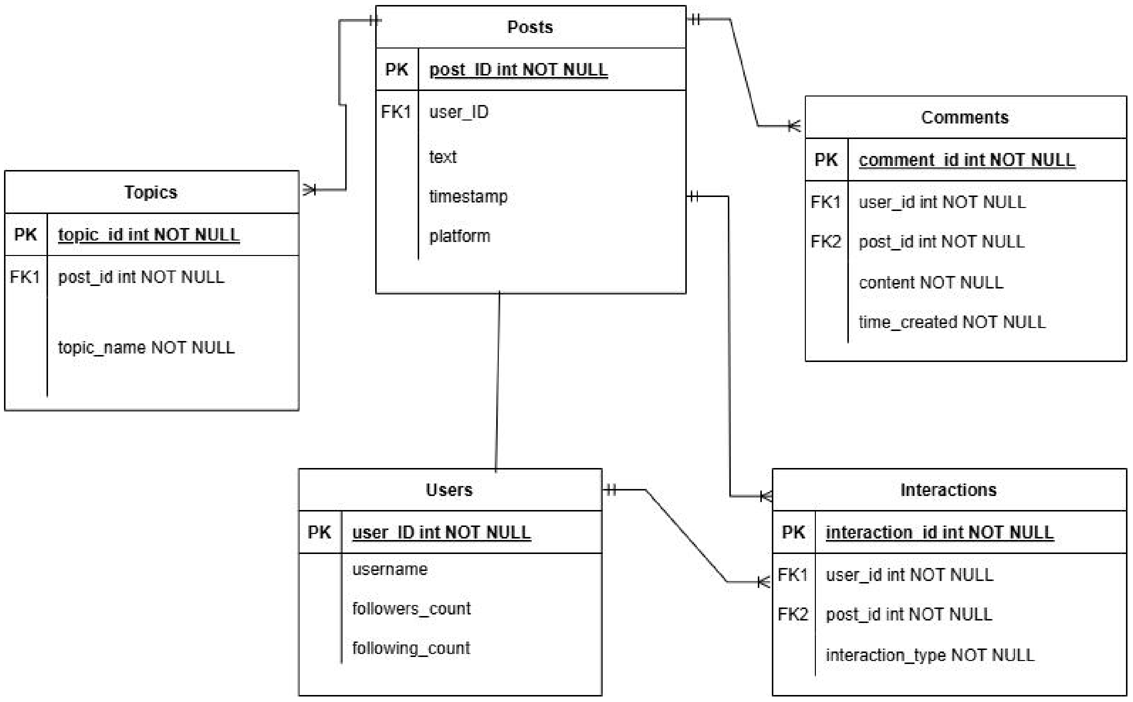
- topic_id (Primary Key)
- topic_namepost_id (Foreign Key)

Interactions

- interaction_id (Primary Key)
- post_id (Foreign Key)
- user_id (Foreign Key)
- interaction_type (like, comment, share)

Comments

- **comment_id** (Primary Key)
- **post_id** (Foreign Key)
- **user_id** (Foreign Key)
- **content**
- **created_at**



SQL SCHEMA

```
CREATE TABLE Users (
   user_id INT PRIMARY KEY AUTO_INCREMENT,
   username VARCHAR(50) NOT NULL,
   location VARCHAR(100),
   followers_count INT,
   following_count INT
);

CREATE TABLE Posts (
   post_id INT PRIMARY KEY AUTO_INCREMENT,
   user_id INT,
   text TEXT,
   timestamp TIMESTAMP,
   platform VARCHAR(20),
   FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Sentiments (
   sentiment_id INT PRIMARY KEY AUTO_INCREMENT,
   post_id INT,
   sentiment_score FLOAT,
   FOREIGN KEY (post_id) REFERENCES Posts(post_id)
);

CREATE TABLE Topics (
    topic_id INT PRIMARY KEY AUTO_INCREMENT,
    topic_name VARCHAR(100),
    post_id INT,
    FOREIGN KEY (post_id) REFERENCES Posts(post_id)
);

CREATE TABLE Interactions (
    interaction_id INT PRIMARY KEY AUTO_INCREMENT,
    post_id INT,
    user_id INT,
    interaction_type ENUM('like', 'comment', 'share'),
    FOREIGN KEY (post_id) REFERENCES Posts(post_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Comments (
    comment_id INT PRIMARY KEY AUTO_INCREMENT,
    post_id INT,
    user_id INT,
    content TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES Posts(post_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

CRUD Operations, Stored Procedures and Triggers.

```
/*Create tables*/
CREATE TABLE Users (
user_id INT PRIMARY KEY,
username VARCHAR(50),
email VARCHAR(100)
);
CREATE TABLE Posts(
post_id INT PRIMARY KEY,
user_id INT,
content TEXT,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
CREATE TABLE Analytics (
analytics_id INT PRIMARY KEY,
post_id INT,
views INT,
likes INT,
shares INT,
FOREIGN KEY(post_id)
REFERENCES Posts(post_id)
);

/*Insert initial data*/
INSERT INTO Users (user_id, username, email) VALUES (102, 'James Mike',
'james@gmail.com');
```

INSERT INTO Users (user_id, username, email) VALUES (104, 'Mary Mueni', 'mueni@gmail.com');

INSERT INTO Users (user_id, username, email) VALUES (106, 'Miller Hugo', 'miller@gmail.com');

INSERT INTO Posts (post_id, user_id, content) VALUES (3, 102, 'One in a Million');

INSERT INTO Posts (post_id, user_id, content) VALUES (5, 104, 'Health and Fitness');

INSERT INTO Posts (post_id, user_id, content) VALUES (7, 106, 'Controversial Talks');

INSERT INTO Analytics (analytics_id, post_id, views, likes, shares) VALUES (3, 3, 150, 40, 38);

INSERT INTO Analytics (analytics_id, post_id, views, likes, shares) VALUES (5, 5, 385, 78, 50);

INSERT INTO Analytics (analytics_id, post_id, views, likes, shares) VALUES (7, 7, 890, 697, 89);

/*Read data*/

SELECT * FROM Posts;

/*Update data*/

UPDATE Posts SET Content = 'Music and luxury' WHERE post_id = 1;

/*Delete data*/

DELETE FROM Posts WHERE post_id = 3;

/*CREATE - To add new data to your database, you use the INSERT statement.*/

/*READ - To retrieve data, you use the SELECT statement.*/

/*UPDATE - To modify existing data, you use the UPDATE statement.*/

/* DELETE - To modify existing data, you use the DELETE statement.*/

Stored Procedures.

/*Stored Procedures*/

```
DELIMITER //
CREATE PROCEDURE AddUser (
IN p_user_id INT,
IN p_username VARCHAR(50),
IN p_email VARCHAR(100)
)
BEGIN
        INSERT INTO Users (user_id, username, email) VALUES (p_user_id, p_username,
p_email);
END;
CREATE PROCEDURE AddPost (
IN p_post_id INT,
IN p_user_id INT,
IN p_content TEXT
)
BEGIN
        INSERT INTO Posts (post_id, user_id, content) VALUES (p_post_id, user_id, content);
END;
CREATE PROCEDURE UpdatePostContent (
IN p_post_id INT,
IN p_new_content TEXT
)
BEGIN
        UPDATE Posts SET content = p_new_content WHERE post_id = p_post_id;
END;
CREATE PROCEDURE DeletePost (
IN p_post_id INT
)
BEGIN
```

```
        DELETE FROM Posts WHERE post_id = p_post_id;
END;
DELIMITER //
```

Triggers.

```
/*Triggers*/
DELIMITER //
CREATE TRIGGER AfterPostInsert
AFTER INSERT ON Posts
FOR EACH ROW
BEGIN
        INSERT INTO Analytics (post-id, views, likes, shares) VALES (NEW.post_id, 0, 0, 0);
END;
CREATE TRIGGER AfterPostDelete
AFTER DELETE ON Posts
FOR EACH ROW
BEGIN
        INSERT INTO PostDEletionLog (post_id) VALUES (OLD.post_id)
END;
DELIMITER //
/*CREATE - To add new data to your database, you use the INSERT statement.*/
/*READ - To retrieve data, you use the SELECT statement.*/
/*UPDATE - To modify existing data, you use the UPDATE statement.*/
/* DELETE - To modify existing data, you use the DELETE statement.*/
```

ADVANCED SQL QUERIES

```
CREATE TABLE Users (
   UserID INT AUTO_INCREMENT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL UNIQUE,
    FullName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    JoinDate DATETIME DEFAULT CURRENT_TIMESTAMP
);


INSERT INTO Users (Username, FullName, Email)
VALUES
    ('zuena', 'Zuena Kiplagat', 'zuena.kiplagat@example.com'),
    ('wamboi', 'Beatrice Wamboi', 'beatrice.wamboi@example.com'),
    ('vivian', 'Vivian Ndung\'u', 'vivian.ndungu@example.com');



CREATE TABLE Posts (
    PostID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT NOT NULL,
    Content TEXT NOT NULL,
    PostDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);


INSERT INTO Posts (UserID, Content)
VALUES
    (1, 'Hello world! This is my first post.'),
    (2, 'Loving the weather today!'),
    (3, 'Just completed a 10K run. Feeling great!');



CREATE TABLE Comments (
    CommentID INT AUTO_INCREMENT PRIMARY KEY,
    PostID INT NOT NULL,
    UserID INT NOT NULL,
    CommentText TEXT NOT NULL,
    CommentDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (PostID) REFERENCES Posts(PostID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);


INSERT INTO Comments (PostID, UserID, CommentText)
VALUES
    (1, 2, 'Welcome to the platform, Vivian'),
    (1, 3, 'Nice to see you here!'),
    (2, 1, 'Yes, the weather is amazing!');


CREATE TABLE Likes (
    LikeID INT AUTO_INCREMENT PRIMARY KEY,
    PostID INT NOT NULL,
    UserID INT NOT NULL,
    LikeDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (PostID) REFERENCES Posts(PostID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);


INSERT INTO Likes (PostID, UserID)
VALUES
    (1, 2),
    (2, 3),
    (3, 1),
    (1, 3);



CREATE TABLE Shares (
    ShareID INT AUTO_INCREMENT PRIMARY KEY,
    PostID INT NOT NULL,
    UserID INT NOT NULL,
    ShareDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (PostID) REFERENCES Posts(PostID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);



INSERT INTO Shares (PostID, UserID)
VALUES
    (1, 2),
    (2, 1),
    (3, 3);



/*QUERY*/
/*Get All Users*/

SELECT * FROM Users;



/*Get All Posts with Their Authors*/

SELECT
    Posts.PostID,
    Users.Username,
    Users.FullName,
    Posts.Content,
    Posts.PostDate
FROM Posts
JOIN Users ON Posts.UserID = Users.UserID;



/*Get All Comments on a Specific Post*/

SELECT
    Comments.CommentID,
    Comments.CommentText,
    Users.Username,
    Comments.CommentDate
FROM Comments
JOIN Users ON Comments.UserID = Users.UserID
WHERE Comments.PostID = 1;



/*Count Likes on Each Post*/

SELECT
    Posts.PostID,
    Posts.Content,
    COUNT(Likes.LikeID) AS LikeCount
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
GROUP BY Posts.PostID;



/*Find Posts Shared by a Specific User*/

SELECT
    Posts.PostID,
    Posts.Content,
    Shares.ShareDate
FROM Shares
JOIN Posts ON Shares.PostID = Posts.PostID
WHERE Shares.UserID = 2;
```

```
/*Engagement Summary for All Users (Posts, Comments, Likes, Shares)*/

SELECT
    Users.UserID,
    Users.Username,
    (SELECT COUNT(*) FROM Posts WHERE Posts.UserID = Users.UserID) AS TotalPosts,
    (SELECT COUNT(*) FROM Comments WHERE Comments.UserID = Users.UserID) AS
TotalComments,
    (SELECT COUNT(*) FROM Likes WHERE Likes.UserID = Users.UserID) AS TotalLikes,
    (SELECT COUNT(*) FROM Shares WHERE Shares.UserID = Users.UserID) AS
TotalShares
FROM Users;



/*Most Liked Post*/

SELECT
    Posts.PostID,
    Posts.Content,
    COUNT(Likes.LikeID) AS LikeCount
FROM Posts
JOIN Likes ON Posts.PostID = Likes.PostID
GROUP BY Posts.PostID
ORDER BY LikeCount DESC
LIMIT 1;



/*Inactive Users (No Posts or Engagements)*/
```

```
SELECT Users.UserID, Users.Username
FROM Users
WHERE Users.UserID NOT IN (
    SELECT DISTINCT UserID FROM Posts
    UNION
    SELECT DISTINCT UserID FROM Comments
    UNION
    SELECT DISTINCT UserID FROM Likes
    UNION
    SELECT DISTINCT UserID FROM Shares
);


/*REPORTS*/
/*Comments and Replies by Post*/

SELECT
    Posts.PostID,
    Posts.Content,
    COUNT(Comments.CommentID) AS CommentCount
FROM Posts
LEFT JOIN Comments ON Posts.PostID = Comments.PostID
GROUP BY Posts.PostID
ORDER BY CommentCount DESC;


/*Post Engagement Score (Likes + Comments + Shares)*/
```

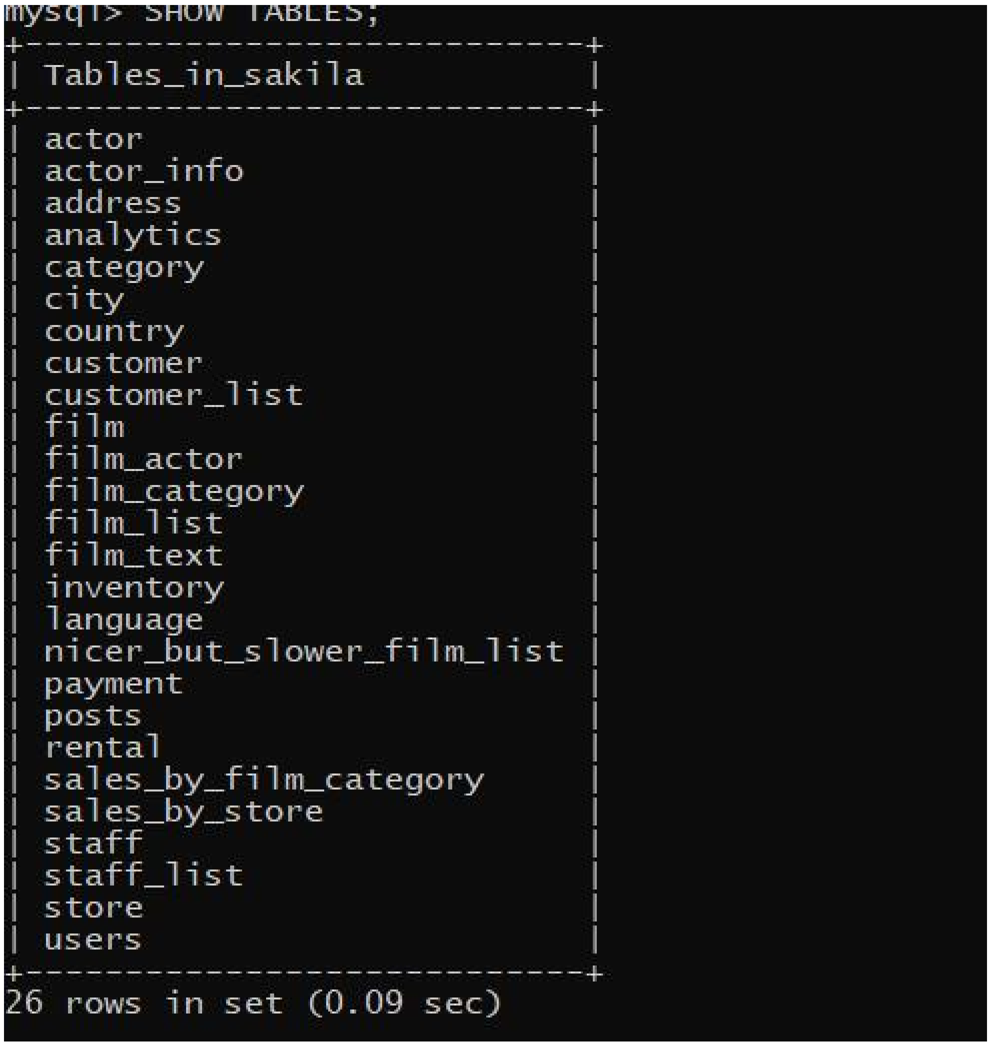
```
SELECT
    Posts.PostID,
    Posts.Content,
    (COUNT(Likes.LikeID) + COUNT(Comments.CommentID) + COUNT(Shares.ShareID)) AS
EngagementScore
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
LEFT JOIN Comments ON Posts.PostID = Comments.PostID
LEFT JOIN Shares ON Posts.PostID = Shares.PostID
GROUP BY Posts.PostID
ORDER BY EngagementScore DESC;



/*Timeline of Posts and Engagements*/


SELECT
    Posts.PostID,
    Posts.Content,
    Posts.PostDate,
    COUNT(Likes.LikeID) AS LikeCount,
    COUNT(Comments.CommentID) AS CommentCount,
    COUNT(Shares.ShareID) AS ShareCount
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
LEFT JOIN Comments ON Posts.PostID = Comments.PostID
LEFT JOIN Shares ON Posts.PostID = Shares.PostID
GROUP BY Posts.PostID
ORDER BY Posts.PostDate DESC;
```

```
/*Activity Report for a Specific User*/

SELECT
    Users.Username,
    (SELECT COUNT(*) FROM Posts WHERE Posts.UserID = Users.UserID) AS TotalPosts,
    (SELECT COUNT(*) FROM Comments WHERE Comments.UserID = Users.UserID) AS
TotalComments,
    (SELECT COUNT(*) FROM Likes WHERE Likes.UserID = Users.UserID) AS TotalLikes,
    (SELECT COUNT(*) FROM Shares WHERE Shares.UserID = Users.UserID) AS
TotalShares
FROM Users
WHERE Users.UserID = 2;


/*Views*/
/* View to Retrieve All Users*/

CREATE VIEW View_AllUsers AS
SELECT * FROM Users;


/*View to Get All Posts with Their Authors*/

CREATE VIEW View_PostsWithAuthors AS
SELECT
    Posts.PostID,
    Users.Username,
    Users.FullName,
    Posts.Content,
    Posts.PostDate
FROM Posts
JOIN Users ON Posts.UserID = Users.UserID;


/*View to Get All Comments on a Specific Post*/

CREATE VIEW View_CommentsByPost AS
SELECT
    Comments.CommentID,
    Comments.CommentText,
    Users.Username,
    Comments.CommentDate,
    Comments.PostID
FROM Comments
JOIN Users ON Comments.UserID = Users.UserID;


/*View to Count Likes on Each Post*/

CREATE VIEW View_LikesCountByPost AS
SELECT
    Posts.PostID,
    Posts.Content,
    COUNT(Likes.LikeID) AS LikeCount
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
```

```
GROUP BY Posts.PostID;



/*View to Find Posts Shared by a Specific User*/


CREATE VIEW View_SharesByUser AS
SELECT
    Shares.UserID,
    Posts.PostID,
    Posts.Content,
    Shares.ShareDate
FROM Shares
JOIN Posts ON Shares.PostID = Posts.PostID;
```

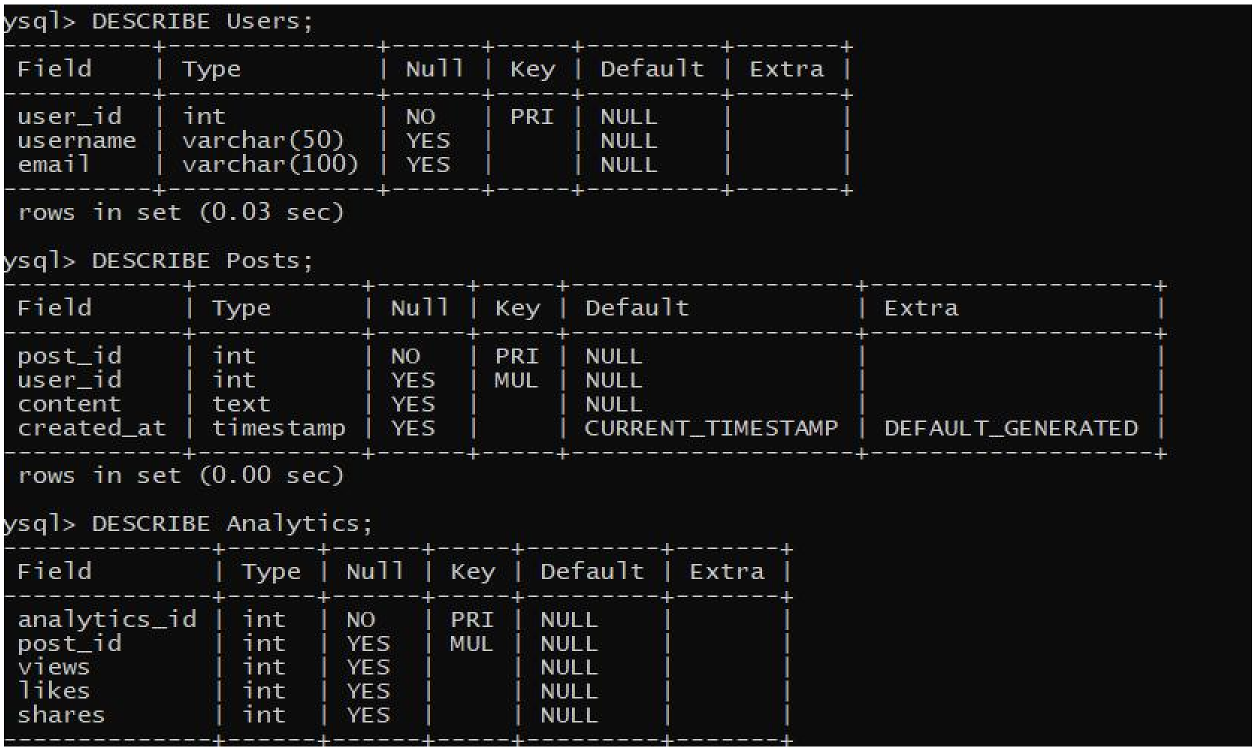
TESTING AND VALIDATION

1. This lists all the tables in the schema and shows that they have been created successfully.



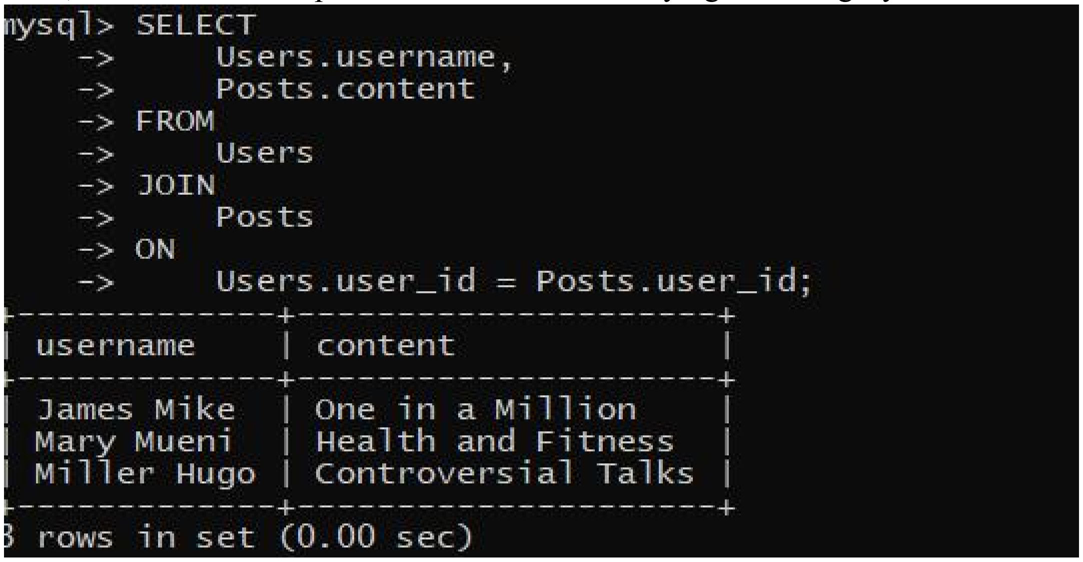
```
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_sakila           |
+----------------------------+
| actor                      |
| actor_info                 |
| address                    |
| analytics                  |
| category                   |
| city                       |
| country                    |
| customer                   |
| customer_list              |
| film                       |
| film_actor                 |
| film_category              |
| film_list                  |
| film_text                  |
| inventory                  |
| language                   |
| nicer_but_slower_film_list |
| payment                    |
| posts                      |
| rental                     |
| sales_by_film_category     |
| sales_by_store             |
| staff                      |
| staff_list                 |
| store                      |
| users                      |
+----------------------------+
26 rows in set (0.09 sec)
```

2. Here, we see the details of table columns, their data types and constraints.



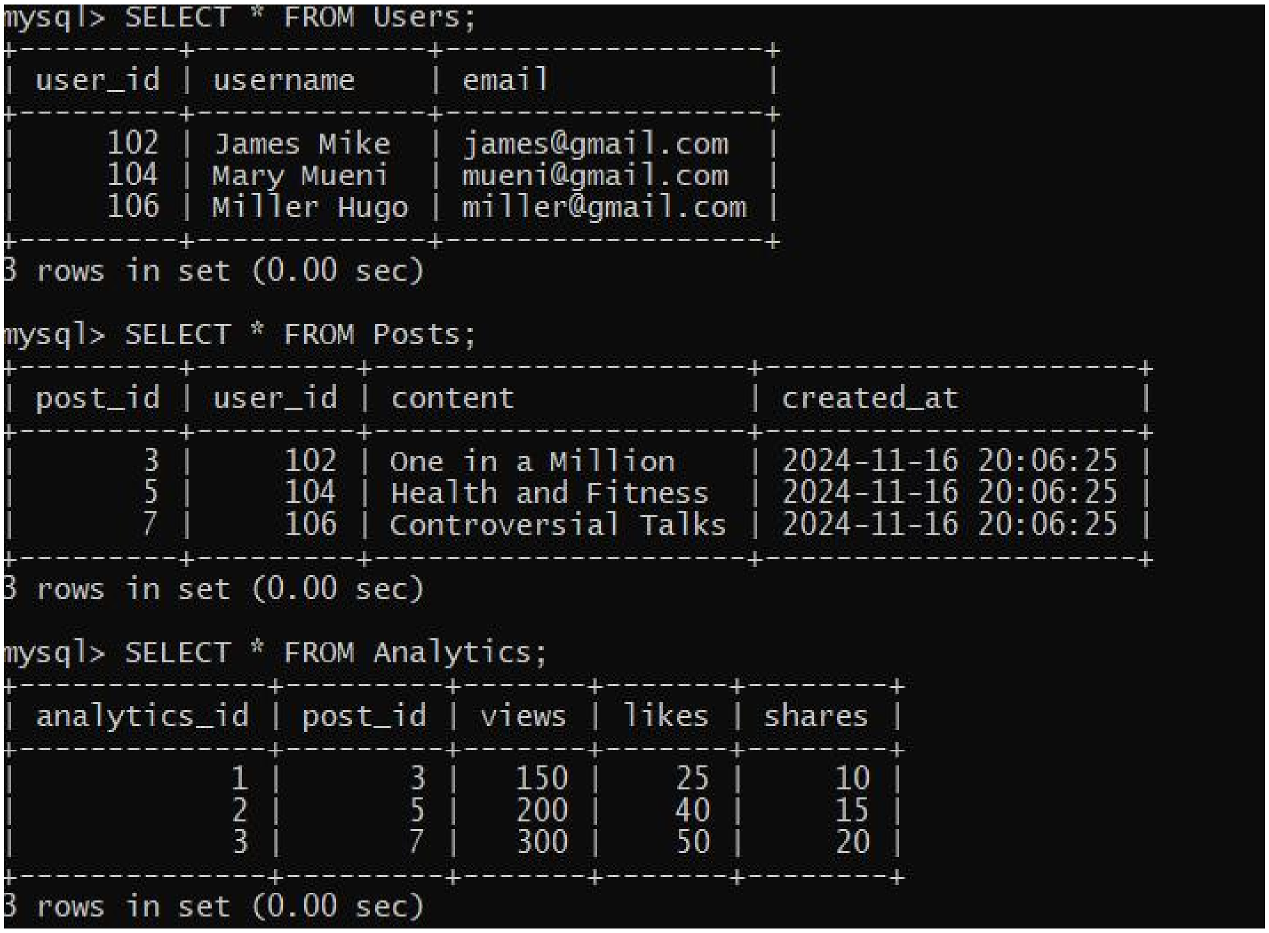
```
ysql> DESCRIBE Users;
----------+--------------+------+-----+---------+-------+
 Field    | Type         | Null | Key | Default | Extra |
----------+--------------+------+-----+---------+-------+
 user_id  | int          | NO   | PRI | NULL    |       |
 username | varchar(50)  | YES  |     | NULL    |       |
 email    | varchar(100) | YES  |     | NULL    |       |
----------+--------------+------+-----+---------+-------+
 rows in set (0.03 sec)

ysql> DESCRIBE Posts;
------------+-----------+------+-----+-------------------+-------------------+
 Field      | Type      | Null | Key | Default           | Extra             |
------------+-----------+------+-----+-------------------+-------------------+
 post_id    | int       | NO   | PRI | NULL              |                   |
 user_id    | int       | YES  | MUL | NULL              |                   |
 content    | text      | YES  |     | NULL              |                   |
 created_at | timestamp | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
------------+-----------+------+-----+-------------------+-------------------+
 rows in set (0.00 sec)

ysql> DESCRIBE Analytics;
--------------+------+------+-----+---------+-------+
 Field        | Type | Null | Key | Default | Extra |
--------------+------+------+-----+---------+-------+
 analytics_id | int  | NO   | PRI | NULL    |       |
 post_id      | int  | YES  | MUL | NULL    |       |
 views        | int  | YES  |     | NULL    |       |
 likes        | int  | YES  |     | NULL    |       |
 shares       | int  | YES  |     | NULL    |       |
--------------+------+------+-----+---------+-------+
```

3. Here, we test relationships between tables and verifying data integrity.



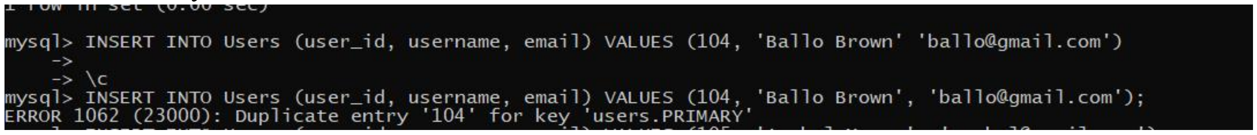
```
nysql> SELECT
    ->     Users.username,
    ->     Posts.content
    -> FROM
    ->     Users
    -> JOIN
    ->     Posts
    -> ON
    ->     Users.user_id = Posts.user_id;
+-------------+---------------------+
| username    | content             |
+-------------+---------------------+
| James Mike  | One in a Million    |
| Mary Mueni  | Health and Fitness  |
| Miller Hugo | Controversial Talks |
+-------------+---------------------+
3 rows in set (0.00 sec)
```

4. This shows all the rows from the tables**.**



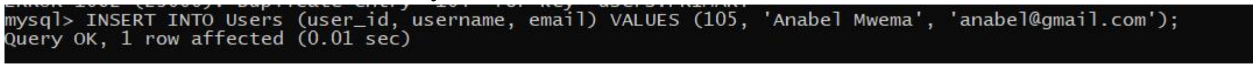
```
mysql> SELECT * FROM Users;
+---------+-------------+------------------+
| user_id | username    | email            |
+---------+-------------+------------------+
|     102 | James Mike  | james@gmail.com  |
|     104 | Mary Mueni  | mueni@gmail.com  |
|     106 | Miller Hugo | miller@gmail.com |
+---------+-------------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Posts;
+---------+---------+---------------------+---------------------+
| post_id | user_id | content             | created_at          |
+---------+---------+---------------------+---------------------+
|       3 |     102 | One in a Million    | 2024-11-16 20:06:25 |
|       5 |     104 | Health and Fitness  | 2024-11-16 20:06:25 |
|       7 |     106 | Controversial Talks | 2024-11-16 20:06:25 |
+---------+---------+---------------------+---------------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Analytics;
+--------------+---------+-------+-------+--------+
| analytics_id | post_id | views | likes | shares |
+--------------+---------+-------+-------+--------+
|            1 |       3 |   150 |    25 |     10 |
|            2 |       5 |   200 |    40 |     15 |
|            3 |       7 |   300 |    50 |     20 |
+--------------+---------+-------+-------+--------+
3 rows in set (0.00 sec)
```

1. In this screenshot, we try to add an existing user and an error is thrown back showing that the user already exists in the database.



```
mysql> INSERT INTO Users (user_id, username, email) VALUES (104, 'Ballo Brown' 'ballo@gmail.com')
    ->
    -> \c
mysql> INSERT INTO Users (user_id, username, email) VALUES (104, 'Ballo Brown', 'ballo@gmail.com');
ERROR 1062 (23000): Duplicate entry '104' for key 'users.PRIMARY'
```

2. In this screenshot, we successfully add a user to the database.



```
mysql> INSERT INTO Users (user_id, username, email) VALUES (105, 'Anabel Mwema', 'anabel@gmail.com');
Query OK, 1 row affected (0.01 sec)
```

CONCLUSION

The whole design and execution of the social analytics database has set up a strong base for handling how users interact, how posts are engaged with and all other analytical insights. This database captures the key parts of social media activity, that is, user data, content posts, and other analytics like views, likes, and shares. Having this database structure all sorted out means that we can retrieve and analyze data efficiently. It shows some seriously valuable insights into how users are engaging and what trends are coming up. With all these features up, the app can help both users and administrators make decisions based on solid data.

RECOMMENDATIONS

1. Regular database maintenance – Perform regular maintenance with stuff like indexing, optimization of queries and cleaning up old and unused data to improve overall performance.
2. Implement data security measures- Ensure the safety of sensitive data by putting in place measures like encryption, data masking and authorization.
3. Ensure data integrity- Use constraints, triggers and stored procedures to keep things organized and reliable.
4. Implement a reliable backup strategy- Perform regular backups to prevent accidental loss, corruption or hardware mishaps. This ensures there is no loss of important data.

FUTURE ENHANCEMENTS

1. Sentiment Analysis: Improve the quantity of the analytics by including the sentiment analysis for comments and posts of users. This would enable the platform to see the tenor of user engagement and shift to sentimental on matters within a particular time frame.
2. Real-Time Analytics: Ensure ERP integration of real-time data processing to monitor, as it happens, the engagement metrics (view, likes, shares etc.). This would enable the users to keep abreast with the real time feedbacks on post and or campaigns.
3. Advanced User Segmentation: Expand a user segmentation by using geographical, activity level, and interest criteria. This would help in gaining more specific information about different categories of users and, therefore, offering them easily relevant content.
4. Predictive Analytics: Use modules in machine learning to make probable the engagement trends of particular users, post the best time to post. One way to operate is that through the use of predictive analysis the amount of engagement from users could be better predicted.

5. Integration with Other Social Platforms: Expand the database to embrace data import from other social media sites such as Twitter and Instagram. This would empower users to combine engagement data across the various platform in one location.
6. Support for Multimedia Content Analytics: With changes in trends within social media, it will be useful to expand the list of supported content, such as multimedia content view, and audio interaction. This would enable the app to remain relevant should there be an increase in content.

REFERENCES

Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media.

Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling (3rd ed.).* Wiley.

MySQL. (2024). MySQL 8.0 reference manual. Oracle Corporation. Retrieved from https://dev.mysql.com/doc/

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database system concepts (7th ed.).* McGraw-Hill Education.

Zafarani, R., Abbasi, M. A., & Liu, H. (2014). *Social media mining: An introduction.* Cambridge University Press.

FINAL APPENDICES OF THE SOCIAL ANALYTICS PLATFORM.

Appendix A: Database Schema

Users Table

SQL

```
CREATE TABLE Users (
   user_id INT PRIMARY KEY,
   username VARCHAR(50),
   email VARCHAR(100)
);
```

user_id: Unique identifier for each user.

username: The username of the user.

email: The email address of the user.

Posts Table

SQL

```
CREATE TABLE Posts (
   post_id INT PRIMARY KEY,
   user_id INT,
   content TEXT,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

post_id: Unique identifier for each post.

user_id: Identifier linking the post to a user.

content: The content of the post.

created_at: Timestamp of when the post was created.

Analytics Table

SQL

```
CREATE TABLE Analytics (
    analytics_id INT PRIMARY KEY,
    post_id INT,
    views INT,
    likes INT,
    shares INT,
    FOREIGN KEY (post_id) REFERENCES Posts(post_id),
);
```

analytics_id: Unique identifier for each analytics record.

post_id: Identifier linking the analytics data to a post.

views: Number of views the post has received.

likes: Number of likes the post has received.

shares: Number of times the post has been shared.

<u>Appendix B: Sample Data</u>

Inserting Sample Data into Users Table

SQL

```
INSERT INTO Users (user_id, username, email) VALUES (102, 'James Mike',
'james@example.com');

INSERT INTO Users (user_id, username, email) VALUES (104, 'Mary Mueni',
'mueni@example.com');

INSERT INTO Users (user_id, username, email) VALUES (106, 'Miller Hugo',
'miller@example.com');
```

Inserting Sample Data into Posts Table

SQL

```
INSERT INTO Posts (post_id, user_id, content) VALUES (3, 102, 'One in a MIllion');

INSERT INTO Posts (post_id, user_id, content) VALUES (5, 104, 'Health and Fitness');
INSERT INTO Posts (post_id, user_id, content) VALUES (7, 106, 'Controversial Talks');
```

Inserting Sample Data into Analytics Table

SQL

```
INSERT INTO Analytics (analytics_id, post_id, views, likes, shares) VALUES (3, 3, 150, 40, 38);
INSERT INTO Analytics (analytics_id, post_id, views, likes, shares) VALUES (5, 5, 385, 70, 50);
INSERT INTO Analytics (analytics_id, post_id, views, likes, shares) VALUES (7, 7, 890, 697,
89);
```

Appendix C: CRUD Operations

Create

SQL

```
-- Insert a new user
INSERT INTO Users (user_id, username, email) VALUES (103, 'alice', 'alice@example.com');

-- Insert a new post
INSERT INTO Posts (post_id, user_id, content) VALUES (3, 103, 'Excited to join!');
```

Read

SQL

```
-- Select all posts
SELECT * FROM Posts;

-- Select all posts by a specific user
SELECT * FROM Posts WHERE user_id = 102;

-- Select analytics data for a specific post
SELECT * FROM Analytics WHERE post_id = 1;
```

Update

SQL

```
-- Update the content of a post
UPDATE Posts SET content = 'Music and Luxury' WHERE post_id = 3;
```

Delete

SQL

```
-- Delete a post
DELETE FROM Posts WHERE post_id = 3;
```

<u>Appendix D: Advanced Queries</u>

```
CREATE TABLE Users (
   UserID INT AUTO_INCREMENT PRIMARY KEY,
   Username VARCHAR(50) NOT NULL UNIQUE,
   FullName VARCHAR(100) NOT NULL,
   Email VARCHAR(100) NOT NULL UNIQUE,
   JoinDate DATETIME DEFAULT CURRENT_TIMESTAMP
);



INSERT INTO Users (Username, FullName, Email)
VALUES
   ('zuena', 'Zuena Kiplagat', 'zuena.kiplagat@example.com'),
   ('wamboi', 'Beatrice Wamboi', 'beatrice.wamboi@example.com'),
   ('vivian', 'Vivian Ndung\'u', 'vivian.ndungu@example.com');
```

```
CREATE TABLE Posts (
    PostID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT NOT NULL,
    Content TEXT NOT NULL,
    PostDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);


INSERT INTO Posts (UserID, Content)
VALUES
    (1, 'Hello world! This is my first post.'),
    (2, 'Loving the weather today!'),
    (3, 'Just completed a 10K run. Feeling great!');



CREATE TABLE Comments (
    CommentID INT AUTO_INCREMENT PRIMARY KEY,
    PostID INT NOT NULL,
    UserID INT NOT NULL,
    CommentText TEXT NOT NULL,
    CommentDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (PostID) REFERENCES Posts(PostID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);
```

```
INSERT INTO Comments (PostID, UserID, CommentText)
VALUES
    (1, 2, 'Welcome to the platform, Vivian'),
    (1, 3, 'Nice to see you here!'),
    (2, 1, 'Yes, the weather is amazing!');


CREATE TABLE Likes (
    LikeID INT AUTO_INCREMENT PRIMARY KEY,
    PostID INT NOT NULL,
    UserID INT NOT NULL,
    LikeDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (PostID) REFERENCES Posts(PostID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);


INSERT INTO Likes (PostID, UserID)
VALUES
    (1, 2),
    (2, 3),
    (3, 1),
    (1, 3);



CREATE TABLE Shares (
    ShareID INT AUTO_INCREMENT PRIMARY KEY,
    PostID INT NOT NULL,
    UserID INT NOT NULL,
    ShareDate DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (PostID) REFERENCES Posts(PostID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE CASCADE
);



INSERT INTO Shares (PostID, UserID)
VALUES
    (1, 2),
    (2, 1),
    (3, 3);



/*QUERY*/
/*Get All Users*/

SELECT * FROM Users;



/*Get All Posts with Their Authors*/

SELECT
    Posts.PostID,
    Users.Username,
    Users.FullName,
    Posts.Content,
    Posts.PostDate
FROM Posts
JOIN Users ON Posts.UserID = Users.UserID;



/*Get All Comments on a Specific Post*/

SELECT
    Comments.CommentID,
    Comments.CommentText,
    Users.Username,
    Comments.CommentDate
FROM Comments
JOIN Users ON Comments.UserID = Users.UserID
WHERE Comments.PostID = 1;




/*Count Likes on Each Post*/

SELECT
    Posts.PostID,
    Posts.Content,
    COUNT(Likes.LikeID) AS LikeCount
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
```

```
GROUP BY Posts.PostID;



/*Find Posts Shared by a Specific User*/

SELECT
    Posts.PostID,
    Posts.Content,
    Shares.ShareDate
FROM Shares
JOIN Posts ON Shares.PostID = Posts.PostID
WHERE Shares.UserID = 2;



/*Engagement Summary for All Users (Posts, Comments, Likes, Shares)*/

SELECT
    Users.UserID,
    Users.Username,
    (SELECT COUNT(*) FROM Posts WHERE Posts.UserID = Users.UserID) AS TotalPosts,
    (SELECT COUNT(*) FROM Comments WHERE Comments.UserID = Users.UserID) AS
TotalComments,
    (SELECT COUNT(*) FROM Likes WHERE Likes.UserID = Users.UserID) AS TotalLikes,
    (SELECT COUNT(*) FROM Shares WHERE Shares.UserID = Users.UserID) AS
TotalShares
FROM Users;



/*Most Liked Post*/
```

```
SELECT
   Posts.PostID,
   Posts.Content,
   COUNT(Likes.LikeID) AS LikeCount
FROM Posts
JOIN Likes ON Posts.PostID = Likes.PostID
GROUP BY Posts.PostID
ORDER BY LikeCount DESC
LIMIT 1;



/*Inactive Users (No Posts or Engagements)*/

SELECT Users.UserID, Users.Username
FROM Users
WHERE Users.UserID NOT IN (
   SELECT DISTINCT UserID FROM Posts
   UNION
   SELECT DISTINCT UserID FROM Comments
   UNION
   SELECT DISTINCT UserID FROM Likes
   UNION
   SELECT DISTINCT UserID FROM Shares
);



/*REPORTS*/
```

```
/*Comments and Replies by Post*/

SELECT
    Posts.PostID,
    Posts.Content,
    COUNT(Comments.CommentID) AS CommentCount
FROM Posts
LEFT JOIN Comments ON Posts.PostID = Comments.PostID
GROUP BY Posts.PostID
ORDER BY CommentCount DESC;


/*Post Engagement Score (Likes + Comments + Shares)*/
SELECT
    Posts.PostID,
    Posts.Content,
    (COUNT(Likes.LikeID) + COUNT(Comments.CommentID) + COUNT(Shares.ShareID)) AS
EngagementScore
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
LEFT JOIN Comments ON Posts.PostID = Comments.PostID
LEFT JOIN Shares ON Posts.PostID = Shares.PostID
GROUP BY Posts.PostID
ORDER BY EngagementScore DESC;


/*Timeline of Posts and Engagements*/

SELECT
```

```
    Posts.PostID,
    Posts.Content,
    Posts.PostDate,
    COUNT(Likes.LikeID) AS LikeCount,
    COUNT(Comments.CommentID) AS CommentCount,
    COUNT(Shares.ShareID) AS ShareCount
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
LEFT JOIN Comments ON Posts.PostID = Comments.PostID
LEFT JOIN Shares ON Posts.PostID = Shares.PostID
GROUP BY Posts.PostID
ORDER BY Posts.PostDate DESC;



/*Activity Report for a Specific User*/


SELECT
    Users.Username,
    (SELECT COUNT(*) FROM Posts WHERE Posts.UserID = Users.UserID) AS TotalPosts,
    (SELECT COUNT(*) FROM Comments WHERE Comments.UserID = Users.UserID) AS
TotalComments,
    (SELECT COUNT(*) FROM Likes WHERE Likes.UserID = Users.UserID) AS TotalLikes,
    (SELECT COUNT(*) FROM Shares WHERE Shares.UserID = Users.UserID) AS
TotalShares
FROM Users
WHERE Users.UserID = 2;



/*Views*/
```

```
/* View to Retrieve All Users*/

CREATE VIEW View_AllUsers AS
SELECT * FROM Users;


/*View to Get All Posts with Their Authors*/

CREATE VIEW View_PostsWithAuthors AS
SELECT
    Posts.PostID,
    Users.Username,
    Users.FullName,
    Posts.Content,
    Posts.PostDate
FROM Posts
JOIN Users ON Posts.UserID = Users.UserID;


/*View to Get All Comments on a Specific Post*/

CREATE VIEW View_CommentsByPost AS
SELECT
    Comments.CommentID,
    Comments.CommentText,
    Users.Username,
    Comments.CommentDate,
    Comments.PostID
```

```
FROM Comments
JOIN Users ON Comments.UserID = Users.UserID;



/*View to Count Likes on Each Post*/

CREATE VIEW View_LikesCountByPost AS
SELECT
   Posts.PostID,
   Posts.Content,
   COUNT(Likes.LikeID) AS LikeCount
FROM Posts
LEFT JOIN Likes ON Posts.PostID = Likes.PostID
GROUP BY Posts.PostID;



/*View to Find Posts Shared by a Specific User*/

CREATE VIEW View_SharesByUser AS
SELECT
   Shares.UserID,
   Posts.PostID,
   Posts.Content,
   Shares.ShareDate
FROM Shares
JOIN Posts ON Shares.PostID = Posts.PostID;
```

Appendix E: Additional Notes

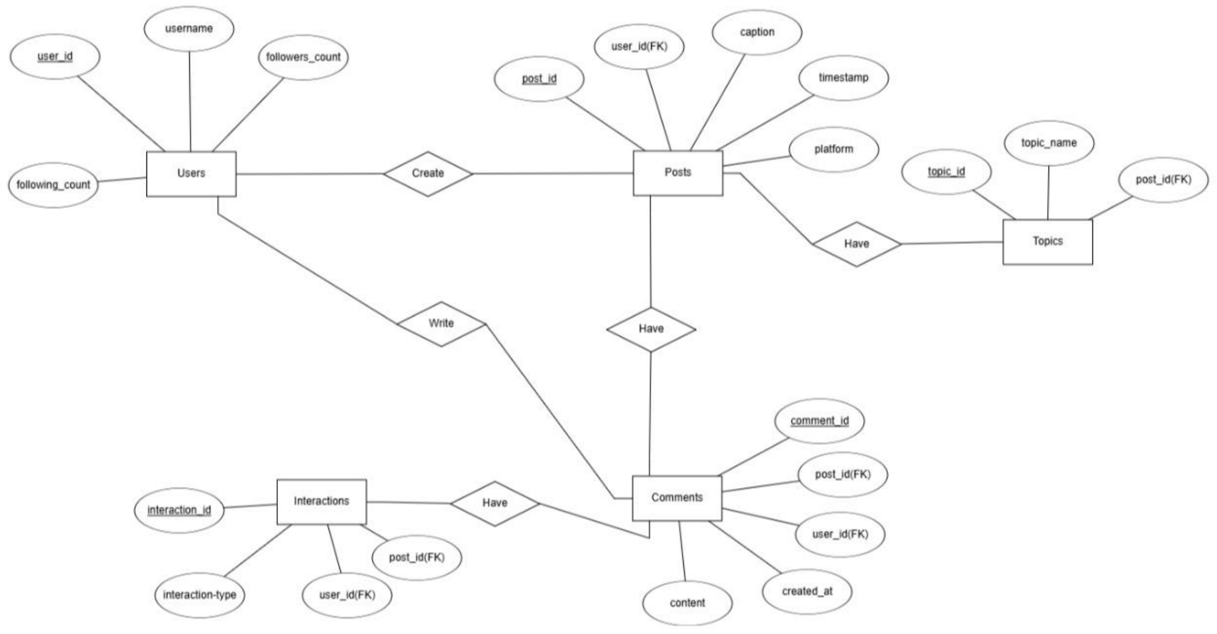
Foreign Key Constraints: Ensure that foreign key constraints are properly set up to maintain referential integrity. This prevents orphaned records in the Posts and Analytics tables.

Indexing: Consider adding indexes on frequently queried columns to improve query performance. For example, indexing the user_id column in the Posts table.

Transactions: Use transactions to group multiple SQL statements into a single unit of work. This ensures that either all operations succeed or none do, maintaining data consistency.

Error Handling: Implement error handling in your SQL scripts to manage exceptions and ensure smooth execution of operations.

Appendix F: ER Diagram


user_id
username
followers_count
following_count
Users
Create
post_id
user_id(FK)
caption
timestamp
platform
Posts
topic_id
topic_name
post_id(FK)
Have
Topics
Write
Have
comment_id
post_id(FK)
user_id(FK)
created_at
content
Comments
interaction_id
Interactions
Have
interaction-type
user_id(FK)
post_id(FK)