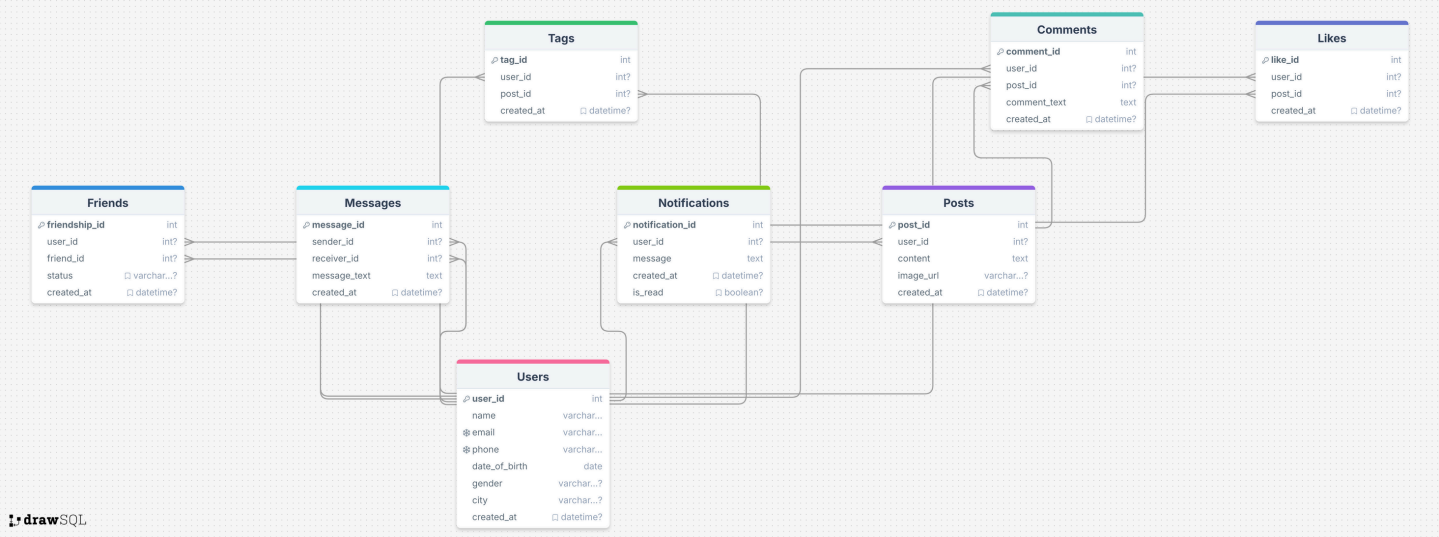
FACEBOOK DATABASE

GAURANG VIVEK PAWAR

ITVEDANT

TABLE OF CONTENTS

- ER DIAGRAM:



- CREATING TABLES:

CREATE DATABASE FacebookDB;

USE FacebookDB;

CREATE TABLE Users (

```
    user_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15) UNIQUE NOT NULL,
    date_of_birth DATE NOT NULL,
    gender VARCHAR(10),
    city VARCHAR(50),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Posts (
    post_id INT PRIMARY KEY,
    user_id INT,
    content TEXT NOT NULL,
    image_url VARCHAR(255),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Likes (
    like_id INT PRIMARY KEY,
    user_id INT,
    post_id INT,
```

```
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (post_id) REFERENCES Posts(post_id)
);

CREATE TABLE Friends (
    friendship_id INT PRIMARY KEY,
    user_id INT,
    friend_id INT,
    status VARCHAR(20) DEFAULT 'Pending',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (friend_id) REFERENCES Users(user_id)
);

CREATE TABLE Comments (
    comment_id INT PRIMARY KEY,
    user_id INT,
    post_id INT,
    comment_text TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (post_id) REFERENCES Posts(post_id)
);
```

```
CREATE TABLE Messages (
    message_id INT PRIMARY KEY,
    sender_id INT,
    receiver_id INT,
    message_text TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sender_id) REFERENCES Users(user_id),
    FOREIGN KEY (receiver_id) REFERENCES Users(user_id)
);

CREATE TABLE Notifications (
    notification_id INT PRIMARY KEY,
    user_id INT,
    message TEXT NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    is_read BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);


CREATE TABLE Tags (
    tag_id INT PRIMARY KEY,
    user_id INT,
    post_id INT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
```

```
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (post_id) REFERENCES Posts(post_id)
);
```

- INSERTING VALUES IN THE TABLES:

```
INSERT INTO Users (user_id, name, email, phone, date_of_birth, gender, city) VALUES
(101, 'Alice Smith', 'alice@gmail.com', '1234567890', '1995-05-15', 'Female', 'New York'),
(102, 'Bob Johnson', 'bob@gmail.com', '2345678901', '1988-08-25', 'Male', 'Los Angeles'),
(103, 'Charlie Brown', 'charlie@gmail.com', '3456789012', '2002-01-30', 'Male', 'Chicago'),
(104, 'Diana Prince', 'diana@hotmail.com', '4567890123', '1975-04-10', 'Female', 'San Francisco'),
(105, 'Ethan Hunt', 'ethan@gmail.com', '5678901234', '1992-12-01', 'Male', 'New York');
```

```
INSERT INTO Posts (post_id, user_id, content, image_url) VALUES
(1, 101, 'Hello World!', NULL),
(2, 101, 'Just got a new job!', NULL),
(3, 102, 'Enjoying the sunny beach!', 'beach.jpg'),
(4, 103, 'Loving the new video game!', NULL),
(5, 104, 'Excited for the weekend!', 'weekend.jpg');
```

```
INSERT INTO Likes (like_id, user_id, post_id) VALUES
(1, 102, 1),
(2, 103, 1),
(3, 101, 3),
```

(4, 104, 2),

(5, 105, 4);

INSERT INTO Friends (friendship_id, user_id, friend_id, status) VALUES

(1, 101, 102, 'Accepted'),

(2, 101, 103, 'Pending'),

(3, 102, 104, 'Accepted'),

(4, 103, 105, 'Declined'),

(5, 104, 101, 'Accepted');

INSERT INTO Comments (comment_id, user_id, post_id, comment_text) VALUES

(1, 102, 1, 'Congratulations!'),

(2, 101, 3, 'Looks great!'),

(3, 103, 2, 'Amazing!'),

(4, 104, 1, 'So happy for you!'),

(5, 105, 4, 'I can relate!');

INSERT INTO Messages (message_id, sender_id, receiver_id, message_text) VALUES

(1, 101, 102, 'Hey, how are you?'),

(2, 102, 101, 'I am good, thanks!'),

(3, 103, 104, 'Wanna hang out this weekend?'),

(4, 104, 101, 'Sure, sounds good!'),

(5, 105, 101, 'Long time no see!');

INSERT INTO Notifications (notification_id, user_id, message, is_read) VALUES

(1, 101, 'Bob liked your post.', FALSE),

(2, 102, 'Alice commented on your post.', FALSE),

(3, 103, 'You have a new friend request from Diana.', TRUE),

(4, 104, 'Ethan mentioned you in a comment.', TRUE),

(5, 105, 'Charlie liked your photo.', FALSE);

INSERT INTO Tags (tag_id, user_id, post_id) VALUES

(1, 101, 2),

(2, 102, 3),

(3, 103, 4),

(4, 104, 1),

(5, 105, 5);

- QUESTION AND ANSWERS:

Questions:

1. Get all users' details from the Users table.

ANS.SELECT * FROM Users;

2. Get all posts created by a user with the user_id = 101.

ANS.SELECT * FROM Posts WHERE user_id = 101;

3. Find all users from the city of "New York".

ANS.SELECT * FROM Users WHERE city = 'New York';

4. Get all friends of user with user_id = 101 whose friendship status is 'Accepted'.

ANS.SELECT u.* FROM Friends f

JOIN Users u ON f.friend_id = u.user_id

WHERE f.user_id = 101 AND f.status = 'Accepted';

5. Find all posts created between 2024-01-01 and 2024-09-01.

ANS.SELECT * FROM Posts WHERE created_at BETWEEN '2024-01-01' AND '2024-09-01';

6. Get details of users from the city of "Chicago", "Los Angeles", or "San Francisco".

ANS.SELECT * FROM Users WHERE city IN ('Chicago', 'Los Angeles', 'San Francisco');

7. Find all users whose email address ends with "@gmail.com".

ANS.SELECT * FROM Users WHERE email LIKE '%@gmail.com';

8. Get the number of posts created by each user.

ANS.SELECT user_id, COUNT(*) AS post_count FROM Posts GROUP BY user_id;

9. Get the total number of likes on each post.

ANS.SELECT post_id, COUNT(*) AS like_count FROM Likes GROUP BY post_id;

10. Get all comments on a specific post along with the user name of the commenter.

ANS.SELECT c.comment_text, u.name

FROM Comments c

JOIN Users u ON c.user_id = u.user_id

WHERE c.post_id = 1;

11. Get all posts liked by a specific user along with the post content and the user who posted it.

ANS.SELECT p.content, u.name

FROM Likes l

JOIN Posts p ON l.post_id = p.post_id

JOIN Users u ON p.user_id = u.user_id

WHERE l.user_id = 1;

12. Get the most liked post.

ANS.SELECT post_id, COUNT(*) AS like_count FROM Likes GROUP BY post_id

ORDER BY like_count DESC LIMIT 1;

13. Find the users who haven't posted anything yet.

ANS.SELECT * FROM Users u

WHERE NOT EXISTS (SELECT 1 FROM Posts p WHERE p.user_id = u.user_id);

14. Get the first 10 characters of each post's content.

ANS.SELECT post_id, SUBSTRING(content, 1, 10) AS short_content FROM Posts;

15. Convert all user names to uppercase.

ANS.SELECT UPPER(name) AS upper_name FROM Users;

16. Get the top 5 users who have received the most likes on their posts.

ANS.SELECT u.user_id, COUNT(*) AS total_likes FROM Likes l

JOIN Posts p ON l.post_id = p.post_id

JOIN Users u ON p.user_id = u.user_id

GROUP BY u.user_id

ORDER BY total_likes DESC LIMIT 5;

17. Get the users who have more than 100 friends.

ANS.SELECT u.user_id, COUNT(*) AS friend_count FROM Friends f

JOIN Users u ON f.user_id = u.user_id

GROUP BY u.user_id

HAVING friend_count > 100;

18. Show the friendship status as 'Friends' if the status is 'Accepted', 'Pending' if it's pending, and 'Not Friends' for any other status.

ANS.SELECT user_id, friend_id,

CASE

  WHEN status = 'Accepted' THEN 'Friends'

  WHEN status = 'Pending' THEN 'Pending'

  ELSE 'Not Friends'

END AS friendship_status

FROM Friends;

19. Get all private messages between two users (user_id = 101 and user_id = 102).

ANS.SELECT * FROM Messages

WHERE (sender_id = 101 AND receiver_id = 102) OR (sender_id = 102 AND receiver_id = 101);

20. Categorize the post length as 'Short', 'Medium', or 'Long' based on character count (e.g., < 50 characters = 'Short', 50-100 = 'Medium', > 100 = 'Long').

ANS.SELECT post_id,

CASE

WHEN CHAR_LENGTH(content) < 50 THEN 'Short'

WHEN CHAR_LENGTH(content) BETWEEN 50 AND 100 THEN 'Medium'

ELSE 'Long'

END AS length_category

FROM Posts;

21. Display the user's age category as 'Minor' if their age is less than 18, 'Adult' if their age is between 18 and 60, and 'Senior' if their age is greater than 60.

ANS.SELECT user_id, name,

CASE

WHEN DATEDIFF(CURRENT_DATE, date_of_birth) / 365 < 18 THEN 'Minor'

WHEN DATEDIFF(CURRENT_DATE, date_of_birth) / 365 BETWEEN 18 AND 60 THEN 'Adult'

ELSE 'Senior'

END AS age_category

FROM Users;

THE QUESTIONS WHICH I HAVE ADDED:

22.From the Users table, get the names and email addresses of all users.

ANS.SELECT name, email FROM Users;

23.From the Posts table, get the content of all posts that do not have an image.

SELECT content FROM Posts WHERE image_url IS NULL;

24.From the Likes table, get the user_id and post_id of all likes.

ANS.SELECT user_id, post_id FROM Likes;

25.From the Comments table, get all comments made by a user with user_id = 102.

ANS.SELECT comment_text FROM Comments WHERE user_id = 102;

26.From the Messages table, get all messages sent by the user with user_id = 101.

ANS.SELECT message_text FROM Messages WHERE sender_id = 101;