Incorrect Queries by an LLM

Group Members: Keerat Singh (2023282), Harsh Sharma (2023282), Sanchay Singh (2023478), Ryan Bhan (2023452)

LLM Used: Deepseek - R1: 1.5b

Interaction:

Input:

– Table: Users

… CREATE TABLE Users (

… UID SERIAL PRIMARY KEY,

… Username VARCHAR(50) NOT NULL UNIQUE,

… Email VARCHAR(100) NOT NULL UNIQUE,

… Password VARCHAR(255) NOT NULL,

… DOB DATE NOT NULL,

… Bio TEXT

…);

…

… – Table: Posts

… CREATE TABLE Posts (

… PID SERIAL PRIMARY KEY,

… Content VARCHAR(250) NOT NULL,

… Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

… Views INT DEFAULT 0,

… Likes INT DEFAULT 0,

```
... Shares INT DEFAULT 0,
... UID INT NOT NULL,
... FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE CASCADE
... );
...
... – Table: Admins
... CREATE TABLE Admins (
... AID SERIAL PRIMARY KEY,
... Password VARCHAR(255) NOT NULL
... );
...
... – Table: Currently_Banned
... CREATE TABLE Banned (
... Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
... Reason TEXT NOT NULL,
... UID INT NOT NULL,
... AID INT NOT NULL,
... PRIMARY KEY (UID),
... FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE CASCADE,
... FOREIGN KEY (AID) REFERENCES Admins(AID) ON DELETE CASCADE
... );
...
... – Table: Removed_Posts
... CREATE TABLE Removed_Posts (
... PID INT PRIMARY KEY,
... AID INT NOT NULL,
... Reason TEXT NOT NULL,
... Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
... FOREIGN KEY (AID) REFERENCES Admins(AID) ON DELETE CASCADE,
... FOREIGN KEY (PID) REFERENCES Posts(PID) ON DELETE CASCADE
... );
...
... – Table: Comments
... CREATE TABLE Comments (
... Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
... Content TEXT NOT NULL,
... UID INT NOT NULL,
... PID INT NOT NULL,
... PRIMARY KEY (Date, UID, PID),
... FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE CASCADE,
... FOREIGN KEY (PID) REFERENCES Posts(PID) ON DELETE CASCADE
... );
...
... – Table: Follows
... CREATE TABLE Follows (
... UID1 INT NOT NULL,
... UID2 INT NOT NULL,

... PRIMARY KEY (UID1, UID2),

... FOREIGN KEY (UID1) REFERENCES Users(UID) ON DELETE CASCADE,

... FOREIGN KEY (UID2) REFERENCES Users(UID) ON DELETE CASCADE

... );

...

... – Table: Friends

... CREATE TABLE Friends (

... UID1 INT NOT NULL,

... UID2 INT NOT NULL,

... PRIMARY KEY (UID1, UID2),

... FOREIGN KEY (UID1) REFERENCES Users(UID) ON DELETE CASCADE,

... FOREIGN KEY (UID2) REFERENCES Users(UID) ON DELETE CASCADE

... );

...

... – Table: Messages

... CREATE TABLE Messages (

... UID1 INT NOT NULL,

... UID2 INT NOT NULL,

... Content TEXT NOT NULL,

... Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

... PRIMARY KEY (UID1, UID2, Date),

... FOREIGN KEY (UID1) REFERENCES Users(UID) ON DELETE CASCADE,
... FOREIGN KEY (UID2) REFERENCES Users(UID) ON DELETE CASCADE
... );
...
... – Table: Notification
... CREATE TABLE Notification (
... NID SERIAL PRIMARY KEY,
... UID INT NOT NULL,
... SID INT NOT NULL,
... Type VARCHAR(50) NOT NULL,
... PID INT,
... Content TEXT NOT NULL,
... isRead BOOLEAN DEFAULT FALSE,
... Created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
... FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE CASCADE,
... FOREIGN KEY (SID) REFERENCES Users(UID) ON DELETE CASCADE,
... FOREIGN KEY (PID) REFERENCES Posts(PID) ON DELETE CASCADE
... );
...
... – Table: Likes
... CREATE TABLE Likes (
... UID INT NOT NULL,
... PID INT NOT NULL,
... PRIMARY KEY (UID, PID),
```

… FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE CASCADE,

… FOREIGN KEY (PID) REFERENCES Posts(PID) ON DELETE CASCADE

…);

…

… – Table: Reports

… CREATE TABLE Reports (

… UID INT NOT NULL,

… PID INT NOT NULL,

… Reason TEXT NOT NULL,

… Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

… PRIMARY KEY (UID, PID),

… FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE CASCADE,

… FOREIGN KEY (PID) REFERENCES Posts(PID) ON DELETE CASCADE

…);

… Given the above relational schema, give a few complex queries (queries including joins, subqueries etc) which would be run on it to run the social media site

Output:

Alright, let me walk through how I approached creating those queries. First, I needed to understand the structure

of the database and what each table represents. Then, I identified the main tables involved: Users, Posts, Admins,

Removed_Posts, Comments, Follows, Friends, Messages, Notification, and Likes.

Next, I thought about the common operations a user might want to perform. From there, I started sketching out different query structures that could achieve those goals using joins, subqueries, and other SQL features.

I wanted to cover various aspects of database management, like data fetching, reporting, notifications, and

relationships between tables. This approach ensures that users can find the queries they need for their specific

needs or projects.

Finally, I organized these queries into a coherent set, making sure each one is complex enough to demonstrate

advanced use cases while being practical for common operations.

Here are several complex queries you can run on your database table based on the schema provided. These queries

involve joins, subqueries, and other advanced SQL features:

---

1. Retrieve User Information with Comments

- This query retrieves user information along with their comments.

```
SELECT u.*, c.* FROM users u JOIN posts p ON u.id = p.uid JOIN comments c O
```

---

2. Show All Posts by Each User

- This query shows all posts associated with each user, including comments and likes.

```
SELECT u.*, p., a.* FROM users u JOIN posts p ON u.id = p.uid

JOIN admins a ON p.uid = a.uid
JOIN removed_posts r ON r.PID = p.PID

WHERE NOT EXISTS (r.AID);
```

---

3. Find Admins with Their Posts

- This query retrieves admin IDs and their associated posts.

```
SELECT a.*, p.* FROM admins a JOIN posts p ON a.id = p.uid;
```

---

4. List Users Who Have Shared a Post

- This query lists users who have shared a post with the specified comment ID.

```
SELECT u.*

FROM users u

JOIN posts p ON u.id = p.uid

JOIN comments c ON u.id = c.uid

WHERE r.* = (SELECT * FROM removed_posts WHERE r.PID = 29835);
```

---

5. Show Notifications Received by a User

- This query shows notifications received by each user.

```
SELECT n.* FROM notifications n JOIN reports r ON n.UID = r.UID
```

```
JOIN follows f ON n.PID = f.uid

JOIN friends f2 ON n.PID = f2.uid;
```

---

6. Get All Comments for a Post

- This query retrieves comments for each post.

```
SELECT c.* FROM comments c JOIN posts p ON c.pid = p.PID

JOIN likes l ON c.*, a.* WHERE l., a.* = (SELECT * FROM likes WHERE tid = 3
```

---

7. Find All Messages in the ‘Reports’ Table

- This query lists all messages from each user’s report.

```
SELECT m.* FROM users u JOIN reports r ON u.id = r.UID

JOIN messages m ON u.id = m.PID;
```

---

8. Show Notifications Received for Each User by Their Friends

- This query shows notifications received by each user from their friends.

```
SELECT f.* FROM friends f2 JOIN notifications n ON f2.UID = n.UID

JOIN follows f3 ON n.PID = f3.uid;
```

---

9. Retrieve All Users with Comments and Likes

- This query retrieves user information along with their comments and likes.

```
SELECT u.* FROM users u

LEFT JOIN posts p ON u.id = p.uid

LEFT JOIN comments c ON u.id = c.uid

LEFT JOIN likes l ON u.id = l.uid;
```

---

10. Find All Users Who Have Shared a Post

- This query lists users who have shared a post with the specified comment ID.

```
SELECT u.*

FROM users u

JOIN posts p ON u.id = p.uid

JOIN comments c ON u.id = c.uid

WHERE r.* = (SELECT * FROM removed_posts WHERE r.PID = 29835);
```

These queries demonstrate various aspects of database management, including joins, subqueries, and filtering

operations. Let me know if you'd like help with specific parts of these queries!

Mistakes:

Query 1:

- Unnecessary join with posts.

Query 2:

- p., a.* is incorrect SQL syntax.
- Admins table does not have UID, so it can’t be joined with posts directly.
- removed_posts should be joined properly.
- WHERE NOT EXISTS (r.AID); is incorrect; r.AID should be checked for NULL.

Query 3:

- Incorrect join on admin since admin does not have uid.

Query 4:

- WHERE r.* = (SELECT * FROM removed_posts WHERE r.PID = 29835); is invalid SQL.
- The table removed_posts is not relevant for shared posts.
- Shares information is stored in the Posts table.

Query 5:

- notifications table is named Notification (singular).
- reports table does not have a direct relationship with notifications.
- follows and friends are incorrectly joined.

Query 6:

- c., *a.* is invalid.
- WHERE l., a.* = … is completely incorrect.
- Comments should be retrieved using PID.

Query 7:

- Messages does not have PID, so JOIN messages m ON u.id = m.PID; is incorrect.

Query 8:

- Incorrect joins with Friends and Follows.

- Friends should be joined with Notification using UID.

Query 9:

- Incorrect LEFT JOIN with comments and likes; they should be joined using the correct columns.

Query 10:

- Same query as query 4 with the same mistakes.