Quackstagram Database Report

i6367247: Julian Nijhuis (A, D, E)

i6293227: Ethan Goetsch (B, C, E)

Table of Contents

Database

The database includes data for users, posts, comments, notifications, pictures as well as user relationship, post relationship records and their histories.

Most tables contain an id as a primary key, except for the relationship tables as the entities for the relationship act as the composite primary key. This was primarily (get it ;) done so that the validation of the relationship between two entities is enforced by the table in order to minimize manual or human error.

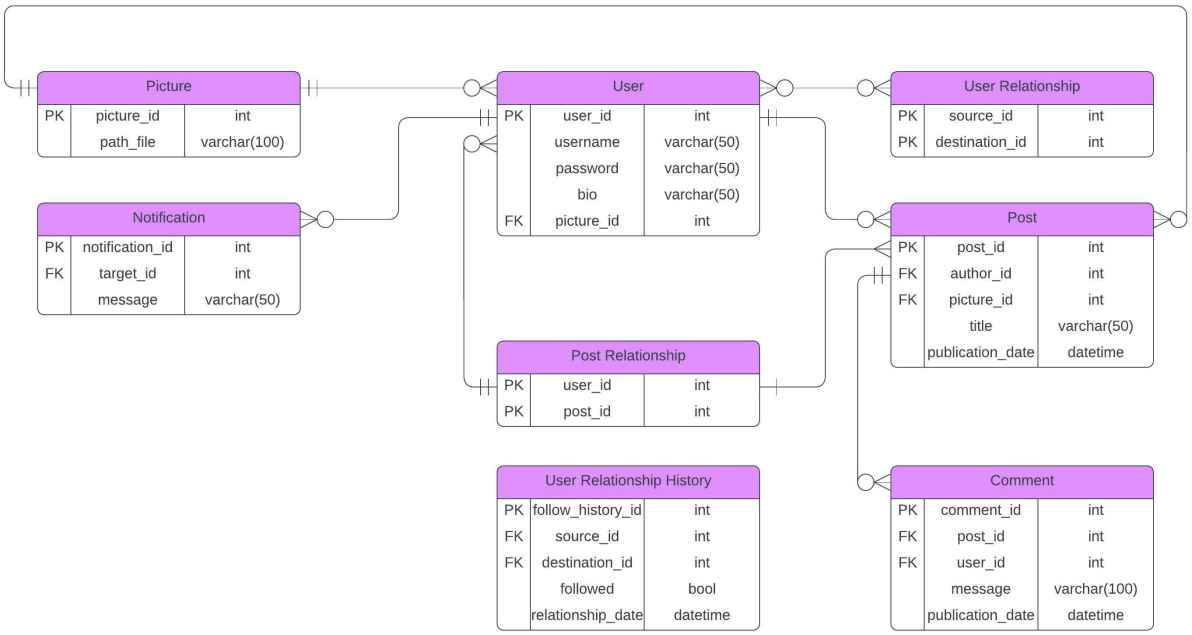
The database is used in Quackstagram through the use of domain objects, data transfer objects and query objects.

Domain objects are the primary entities in the database, such as user, post... etc. and the data transfer objects are the serializable versions of the domain objects that represent the data to be stored in the database.

On initialization, the application retrieves all entities in the database from as the data transfer objects and deserializes them into the domain object.

The communication between the application and database is done through query objects that encapsulate creating the SQL prepared query, applying parameters to the query and reading the returned result.

Entity-Relation Diagram



Tables

User Table

user_id	username	password	bio	picture_id
1	elvenarcher	bowandarrow123	Protector of the enchanted woods	1
2	wizardoflore	magicwand456	Keeper of ancient spells	2
3	dwarvenfighter	axeandshield789	Guardian of the mountain mines	3
4	fairydust	pixie123	Spreading magic wherever I fly	4
5	dragonrider	firebreath456	Soaring the skies on my dragon	5

User Relationship Table

source_id	destination_id

1	2
1	3
1	18
2	4
2	5
3	6
4	7
5	8

Post Table

post_id	author_id	picture_id	title	publication_date
1	1	1	The Hidden Glade	2024-05-02 00:00:00
2	2	2	Secrets of the Arcane	2024-05-11 00:00:00
3	3	3	An Ode to the Forge	2024-05-01 00:00:00
4	4	4	Magic in the Moonlight	2024-05-06 00:00:00
5	5	5	Flight of the Dragons	2024-05-03 00:00:00

Comment Table

comment_id	post_id	user_id	message	publication_date
1	1	1	Behold the arcane secrets unveiled!	2024-05-01 00:00:00
2	2	2	A true masterpiece of eldritch lore.	2024-05-02 00:00:00
3	3	3	Such a mystical journey through	2024-05-03 00:00:00

			the enchanted woods.	
4	4	4	This potion recipe has transformed my brew!	2024-05-04 00:00:00
5	5	5	The dragon’s flight was depicted with fiery passion!	2024-05-05 00:00:00

Functional Dependencies

Special Note

User table uses a user_id as its primary key instead of a username since users can change their username.

Similarluy, pictures use picture_id since for testing on a local device multiple posts/profile pictures could use the same path file for a picture

User

user_id → (user_id, username, password, bio, picture_id)

username → (username, user_id, password, bio, picture_id)

Picture

picture_id → (picture_id, path_file)

Post

post_id → (post_id, author_id, picture_id, title)

Notification

notification_id → (notification_id, target_id, message)

User Relationship

(source_id, destination_id) → (source_id, destination_id)

User Relationship History

(source_id, destination_id) → (source_id, destination_id)

Post Relationship

(user_id, post_id) → (user_id, post_id)

Normalization

A table is in 3NF when every functional dependency either has a superkey on the left hand side or a prime attribute on the right hand side. Here is proof that our tables are in 3NF

Proof for User table

The relations in the user table can be written down las follows:

user_id → username

user_id → password

user_id → bio

user_id → picture_id

This table is in 3NF because every dependency has a superkey on the left hand side.

Proof for Picture table

We can write down the relation in the Picture table as follows:

picture_id → path_file

This relation is in 3NF because there is a superkey on the left hand side of the dependency.

Proof for Post table

We can write down the relation in the Post table as follows:

post_id → author_id

post_id → picture_id

post_id → title.

These relations are all in 3NF since they all have a superkey on the left hand side.

Proof for Notification table

We can write down the relation in the Notification table as follows:

notification_id → target_id

notification_id → message.

All these relation are in 3NF because in both relations there is a superkey on the left hand side.

Proof for User Relationship table

We can write down the User Relationship table as follows:

(source_id, destination_id) → (source_id, destination_id).

This is in 3nf since the superkey is on the left hand side.

Proof for User Relationship History table

We can write down the User Relationship history table as follows:

follow_history_id → source_id

follow_history_id → destination_id

follow_history_id → followed

follow_history_id → relationship_date

This is in 3nf since the superkey is on the left hand side.

Proof for Post Relationship table

We can write down the Post Relationship table as follows:

(user_id, post_id) → (user_id, post_id)

This is in 3nf since the superkey on the left hand side.

Views

User Behaviour

The user_behaviour view returns the amount of followers and amount of people each user follows.

This is extremely valuable data and is important for our users.

Content Popularity

The content_popularity view returns the amount of likes and comments for each post.

This is also extremely valuable data as being able to see how popular a post is is important to our users and (money)

System Analytics

The system_analytics view returns the amount of posts made each day.

This is important for the business as being able to determine user engagement on a daily basis can be used in data analysis and for metrics.

Indexes

The proposed indexes are the primary keys for all tables.

Due to primary keys automatically being indexed I cannot analyze speed before and after indexing since the table has been indexed from the beginning

I cannot find any other attributes to index a table under as most tables are joined on their primary keys which are already indexed

Procedures, Functions and Triggers

Triggers

The triggers implemented are used to trigger a procedure thats updates the table user_relationship_history after data has been inserted or deleted to/from user_relationship table.

The purpose of this is to have the history of user's relationship automatically update whenever new data is needed without having to manually add in the data entries.

Another trigger implemented validates and deletes invalid entries in the user_relationship_history table after data has been inserted or deleted.

This is used to ensure that all automatically added entries in the table are valid.

So that our lovely employees do not have to do unnecessary work :)

Procedure

A procedure implemented is used to update the user_relationship table, populating it with new data.

This is because this procedure is called from multiple triggers and having a single source allows it to be maintainable and extendable.

Another procedure validates all the entries in user_relationship_history table such that a user can only unfollow another user if they have previously followed that user.

Another procedure then deletes invalid entries in the user_relationship_history table.

These procedures were implemented since the data entries are automatically added to the table and we would like to verify those entries are correct and if incorrect then can correct them easily.

This could be used to analyze user's history and interaction with other users for data analysis and marketing.

Functions

A function implemented returns the count users.

This is so that we have a metric for the popularity and user count for our application.

SQL Queries

1. List all users who have more than X followers where X can be any integer value

Query

```
SELECT destination_id AS user_id, COUNT(source_id) AS followers
FROM user_relationship
```

```
GROUP BY destination_id
HAVING followers > X
```

Answer

Answer for above query for followers > 1

user_id	followers
18	2

2. Show the total number of posts made by each user. (You will have to decide how this is done, via a username or userid)

Query

```
SELECT user_id, COUNT(*)
FROM quackstagram.post_relationship
GROUP BY user_id
```

Answer

Note* this answer is shortend to 5 rows. Otherwise we would have to make a too large table.

user_id	COUNT(*)
1	4
2	1
3	1
4	1
5	1

3. Find all comments made on a particular user’s post

Query

```
SELECT user_id, message
FROM quackstagram.comment
WHERE user_id = X
```

Answer for user_id = 3

user_id	message
3	Such a mystical journey through the enchanted woods.

4. Display the top X most liked posts

Query

```
SELECT post_id, like_count
        FROM (SELECT
        p.post_id,
        coalesce(l.like_count, 0) AS like_count,
        coalesce (c.comment_count, 0) AS comment_count
    FROM
        post p
    LEFT JOIN
        (SELECT post_id, count(user_id) AS like_count
        FROM post_relationship
        GROUP BY post_id) l ON p.post_id = l.post_id
    LEFT JOIN
        (SELECT post_id, count(comment_id) AS comment_count
        FROM comment
        GROUP BY post_id) c ON p.post_id = c.post_id) AS most_li
    ORDER BY like_count DESC
    LIMIT X
```

Answer for X = 5

post_id	like_count
2	2
3	2
17	2
1	1
4	1

5. Count the number of posts each user has liked

Query

```
SELECT user_id, count(*)
FROM quackstagram.post_relationship
GROUP BY user_id
```

Answer

Note* this answer is shortend to 5 rows. Otherwise we would have to make a too large table.

user_id	count(*)
1	4
2	1
3	1
4	1
5	1

6. List all users who haven’t made a post yet

Query

```
SELECT user_id
FROM user
```

```
WHERE user_id NOT IN (SELECT author_id FROM post)
```

Answer

The query will return an empty column since every user has a post

7. List users who follow each other

Query

```
SELECT source_id, destination_id
FROM user_relationship
WHERE (destination_id, source_id) in (select source_id, destinat
```

Answer

User 1 follows User 18, and vice versa.

source_id	destination_id
1	18
18	1

8. Show the user with the highest number of posts

Query

```
SELECT author_id, COUNT(author_id) as posts
FROM post
GROUP BY author_id
ORDER BY posts
DESC
LIMIT 1
```

Answer

author_id	posts
1	2

9. List the top X users with the most followers

Query

```
SELECT destination_id, count(destination_id)
FROM user_relationship
GROUP BY destination_id
ORDER BY count(destination_id) DESC
LIMIT X
```

Answer for X = 5

destination_id	count(destination_id)
18	2
2	1
3	1
4	1
5	1

10. Find posts that have been liked by all users

Query

```
SELECT post_id, count(post_id)
FROM post_relationship
GROUP BY post_id
HAVING count(post_id) = (SELECT count(user_id) FROM user)
ORDER BY count(post_id) DESC
```

Answer

The query will return empty columns since no posts have been like by all users.

11. Display the most active user (based on posts, comments , and likes)

Query

```
WITH user_posts AS
(
    SELECT author_id, count(author_id) AS posts
    FROM post
    GROUP BY author_id
),
user_comments AS
(
    SELECT user_id, count(comment_id) AS comments
    FROM comment
    GROUP BY user_id
),
user_likes AS
(
    SELECT user_id, count(post_id) AS likes
    FROM post_relationship
    GROUP user_id
)
SELECT author_id AS user, max(posts + comments + likes) AS activ
FROM user_posts
INNER JOIN user_comments ON user_posts.author_id = user_comments
INNER JOIN user_likes ON user_posts.author_id = user_likes.user_
GROUP BY user
LIMIT 1
```

Answer

user	activity

1	7

12. Find the average number of likes per post for each user

Query

```
SELECT author_id , AVG(like_count)
FROM post
JOIN content_popularity ON post.post_id = content_popularity.pos
GROUP by author_id
```

Answer

Note* this answer is shortend to the first 5 users. Otherwise we would have to make a too large table.

author_id	AVG(like_count)
1	1.0000
2	2.0000
3	2.0000
4	1.0000
5	1.0000

13. Show posts that have more comments than likes

Query

```
SELECT post_id
FROM (content_popularity)
WHERE comment_count>like_count
```

Answer

This query will return empty columns because there are no posts which have more comments than likes.

14. List the users who have liked every post of a specific user

Query

```
WITH user_posts AS
(
    SELECT post_id
    FROM post
    WHERE author_id = X
),
user_likes as
(
    SELECT user_id, post_id
    FROM post_relationship
    WHERE post_id in (SELECT post_id FROM user_posts)
    GROUP BY user_id, post_id
),
user_likes_count AS
(
    SELECT user_id, count(post_id) ASliked_posts_count
    FROM user_likes
    GROUP BY user_id
)
SELECT USER.user_id
FROM user
JOIN user_likes_count ulc ON USER.user_id = ulc.user_id
WHERE ulc.liked_posts_count = (SELECT COUNT(post_id) FROM user_
```

Answer

The query will return an empty column since no user has liked every post of another user.

Returns an empty set for user_id = 1

15. Display the most popular post of each user (based on likes)

Query

```
SELECT author_id, MAX(like_count)
FROM post
JOIN content_popularity ON post.post_id = content_popularity.pos
GROUP BY author_id
```

Answer

Note* this answer is shortend to the first 5 users. Otherwise we would have to make a too large table.

author_id	MAX(like_count)
1	1
2	2
3	2
4	1
5	1

16. Find the user(s) with the highest ratio of followers to following

Query

```
SELECT user_id, followers_count / followings_count AS ratio
FROM user_behaviour
```

```
ORDER BY ratio DESC
LIMIT 1
```

Answer

user_id	ratio
18	2.0000

17. Show the month with the highest number of posts made

Query

```
SELECT post_date, posts_count
FROM system_analytics
ORDER BY posts_count DESC
LIMIT 1
```

Answer

post_date	post_count
2024-05-01 00:00:00	3

18. Identify users who have not interacted with a specific user's posts

Query

```
-- Naive approach with duplicate query
SELECT user_id
FROM user
WHERE user_id NOT IN
    (SELECT user_id
```

```
    FROM post_relationship
    WHERE post_id IN (SELECT post_id FROM post WHERE author_id =
AND user_id NOT IN
    (SELECT user_id
    FROM comment
    WHERE post_id in (SELECT post_id FROM post WHERE author_id =

    -- Better approach with no duplicate query
WITH user_posts AS
(
    SELECT post_id
    FROM post
    WHERE post_id = X
),
liked_users AS
(
    SELECT user_id
    FROM post_relationship
    WHERE post_id IN (SELECT post_id FROM user_posts)
),
commented_users AS
(
    SELECT user_id
    FROM comment
    WHERE post_id IN (SELECT post_id FROM user_posts)
)
SELECT user_id
FROM user
WHERE user_id NOT IN (SELECT user_id FROM liked_users)
AND user_id NOT IN (SELECT user_id FROM commented_users)
```

Answer

Answer for above query for user_id = 1. Shortened as only user 1 and 5 have interacted with user 1's posts

User
2
3
4
6
7

19. Display the user with the greatest increase in followers in the last X days

Query

```
ESCSELECT destination_id AS user, count(followed) AS followers_g
FROM user_relationship_history urh
WHERE followed = 1
AND relationship_date > date_sub(now() , INTERVAL X DAY)
GROUP BY destination_id
ORDER BY followers_gained DESC
LIMIT 1
```

Answer

Answer for above query for last 10 days

user	followers_gained
12	1

20. Find users who are followed by more than X% of the platform users

Query

```
SELECT user_id, followers_count
FROM user_behaviour
```

```
WHERE followers_count > (SELECT COUNT(user_id) FROM USER) / 100
```

Answer

Answer for above query for a user followed by more than 5% (X = 5) of the platform users

user_id	followers_count
18	2

Appendix

Creates

```
CREATE TABLE `Post` (
  `Post_ID` int,
  `Author_ID` int,
  `Picture_ID` int,
  `Title` varchar(50),
  PRIMARY KEY (`Post_ID`)
);

CREATE TABLE `Picture` (
  `Picture_ID` int,
  `Path_File` varchar(50),
  PRIMARY KEY (`Picture_ID`)
);

CREATE TABLE `Relationship` (
  `Follower_ID` int,
  `Followee_ID` int,
  PRIMARY KEY (`Follower_ID`, `Followee_ID`)
);

CREATE TABLE `Notification` (
```

```
  `Notification_ID` int,
  `Target_ID` int,
  `Message` varchar(50),
  PRIMARY KEY (`Notification_ID`)
);

CREATE TABLE `User` (
  `User_ID` int,
  `Username` varchar(50),
  `Password` varchar(50),
  `Bio` varchar(50),
  `Picture_ID` int,
  PRIMARY KEY (`User_ID`)
);

CREATE TABLE `post_relationship` (
  `user_id` int,
  `post_id` int,
  PRIMARY KEY (`user_id`, `post_id`)
);

CREATE TABLE `Comment` (
  `comment_id` int,
  `post_id` int,
  `user_id` int,
  `message` int,
  `publication_date` time,
  PRIMARY KEY (`comment_id`)
);

CREATE TABLE `user_relationship_history` (
  `follow_history_id` int,
  `source_id` int,
  `destination_id` int,
  `followed` bool,
  `relationship_date` datetime,
```

```
  PRIMARY KEY (`follow_history_id`)
);
```

Inserts

```
INSERT INTO quackstagram.user_relationship_history (follow_histo
VALUES
(1, 1, 2, 1, '2024-05-01'),  -- 1 follows 2
(2, 2, 3, 1, '2024-05-01'),
(3, 3, 4, 1, '2024-05-02'),
(4, 4, 5, 1, '2024-05-02'),
(5, 1, 2, 0, '2024-05-03'),  -- 1 unfollows 2 after following
(6, 2, 1, 1, '2024-05-03'),  -- 2 follows 1
(7, 2, 1, 0, '2024-05-04'),  -- 2 unfollows 1 after following
(8, 5, 6, 1, '2024-05-04'),  -- 5 follows 6
(9, 6, 5, 0, '2024-05-05'),  -- 6 unfollows 5 after following
(10, 5, 6, 1, '2024-05-05'), -- 5 follows 6 again
(11, 7, 8, 1, '2024-05-06'),
(12, 8, 7, 1, '2024-05-06'),
(13, 8, 7, 0, '2024-05-07'), -- 8 unfollows 7 after following
(14, 9, 10, 1, '2024-05-07'),
(15, 10, 9, 1, '2024-05-08'),
(16, 10, 9, 0, '2024-05-08'), -- 10 unfollows 9 after following
(17, 11, 12, 1, '2024-05-09'),
(18, 12, 11, 1, '2024-05-09'),
(19, 13, 14, 1, '2024-05-10'),
(20, 14, 13, 0, '2024-05-10'); -- 14 unfollows 13 after followi

INSERT INTO user (user_id, username, password, bio, picture_id)
(1, 'elvenarcher', 'bowandarrow123', 'Protector of the enchante
(2, 'wizardoflore', 'magicwand456', 'Keeper of ancient spells',
(3, 'dwarvenfighter', 'axeandshield789', 'Guardian of the mounta
(4, 'fairydust', 'pixie123', 'Spreading magic wherever I fly', 4
(5, 'dragonrider', 'firebreath456', 'Soaring the skies on my dra
(6, 'orcchieftain', 'battleaxe789', 'Ruler of the orc tribes', 6
```

```
(7, 'centaurguard', 'hoovesandarrows123', 'Patroller of the myst
(8, 'sirenssong', 'oceanmelody456', 'Singer of the sea', 8),
(9, 'goblintrickster', 'mischief789', 'Master of pranks and trea
(10, 'nymphoftheforest', 'treespirit123', 'Protector of the anci
(11, 'vampirenight', 'moonlight456', 'Wanderer under the moon',
(12, 'witchybrew', 'cauldron789', 'Concocter of mystical potions
(13, 'trollbridge', 'billygoats123', 'Guardian of the forgotten
(14, 'phoenixreborn', 'flameoflife456', 'Eternal bird of fire an
(15, 'elfscribe', 'quillandink789', 'Keeper of the enchanted scr
(16, 'krakenbeast', 'depthsofsea123', 'Ruler of the ocean depths
(17, 'griffintamer', 'windsoar456', 'Flyer among the clouds', 17
(18, 'mermaidpearl', 'deepblue789', 'Dweller of the coral palace
(19, 'giantogre', 'clubandsmash123', 'Giant of the northern hill
(20, 'leprechaunluck', 'rainbowgold456', 'Bringer of fortune and

INSERT INTO picture (picture_id, path_file) VALUES
(1, 'src/main/resources/img/storage/elvenarcher.jpg'),
(2, 'src/main/resources/img/storage/wizardoflore.jpg'),
(3, 'src/main/resources/img/storage/dwarvenfighter.jpg'),
(4, 'src/main/resources/img/storage/fairydust.jpg'),
(5, 'src/main/resources/img/storage/dragonrider.jpg'),
(6, 'src/main/resources/img/storage/orcchieftain.jpg'),
(7, 'src/main/resources/img/storage/centaurguard.jpg'),
(8, 'src/main/resources/img/storage/sirenssong.jpg'),
(9, 'src/main/resources/img/storage/goblintrickster.jpg'),
(10, 'src/main/resources/img/storage/nymphoftheforest.jpg'),
(11, 'src/main/resources/img/storage/vampirenight.jpg'),
(12, 'src/main/resources/img/storage/witchybrew.jpg'),
(13, 'src/main/resources/img/storage/trollbridge.jpg'),
(14, 'src/main/resources/img/storage/phoenixreborn.jpg'),
(15, 'src/main/resources/img/storage/elfscribe.jpg'),
(16, 'src/main/resources/img/storage/krakenbeast.jpg'),
(17, 'src/main/resources/img/storage/griffintamer.jpg'),
(18, 'src/main/resources/img/storage/mermaidpearl.jpg'),
(19, 'src/main/resources/img/storage/giantogre.jpg'),
(20, 'src/main/resources/img/storage/leprechaunluck.jpg');
```

```
INSERT INTO post_relationship (user_id, post_id) VALUES
(1, 5),
(2, 4),
(3, 7),
(4, 2),
(5, 1),
(6, 10),
(7, 3),
(8, 6),
(9, 9),
(10, 8),
(11, 11),
(12, 13),
(13, 12),
(14, 14),
(15, 15),
(16, 16),
(17, 18),
(18, 17),
(19, 19),
(20, 20);

INSERT INTO notification (notification_id, target_id, message) V
(1, 1, 'You have a new follower from the enchanted forest'),
(2, 2, 'Your magical post just got a new like'),
(3, 3, 'New comment on your craft'),
(4, 4, 'Your story has enchanted another reader'),
(5, 5, 'Dragon sightings reported near your last post'),
(6, 6, 'Battle cry appreciated by fellow orcs'),
(7, 7, 'Centaur race event invitation'),
(8, 8, 'Sirens have replied to your song'),
(9, 9, 'Your trick has been uncovered'),
(10, 10, 'New flowers bloom in your forest'),
(11, 11, 'Invitation to the vampire gathering'),
(12, 12, 'Your potion recipe has been featured'),
```

```
(13, 13, 'A goat tried to cross the bridge'),
(14, 14, 'Your phoenix tale has inspired many'),
(15, 15, 'Ancient scroll discoveries need your expertise'),
(16, 16, 'New depths have been explored'),
(17, 17, 'Griffin spotted near your location'),
(18, 18, 'Pearl diving contest awaits'),
(19, 19, 'Feast preparations are complete'),
(20, 20, 'Gold found at the end of the rainbow');

INSERT INTO user_relationship (source_id, destination_id) VALUES
(1, 2),
(1, 3),
(2, 4),
(2, 5),
(3, 6),
(4, 7),
(5, 8),
(6, 9),
(7, 10),
(8, 11),
(9, 12),
(10, 13),
(11, 14),
(12, 15),
(13, 16),
(14, 17),
(15, 18),
(16, 19),
(17, 20),
(18, 1);

INSERT INTO comment (comment_id, post_id, user_id, message, publ
(1, 1, 1, 'Behold the arcane secrets unveiled!', '2024-05-01'),
(2, 2, 2, 'A true masterpiece of eldritch lore.', '2024-05-02'),
(3, 3, 3, 'Such a mystical journey through the enchanted woods.
(4, 4, 4, 'This potion recipe has transformed my brew!', '2024-0
```

```
(5, 5, 5, 'The dragon’s flight was depicted with fiery passion!
(6, 6, 6, 'Your tale of the orc warlord was truly gripping.', '2
(7, 7, 7, 'The song of the siren echoed deep within my soul.',
(8, 8, 8, 'A treasure map that leads to ancient mysteries!', '2(
(9, 9, 9, 'The shadows of the vampire castle were chilling.', '2
(10, 10, 10, 'The lore of the ancient scrolls is enlightening.',
(11, 11, 11, 'A giants tale that shakes the very earth!', '2024-
(12, 12, 12, 'Bewitched by the witch’s latest concoction.', '202
(13, 13, 13, 'Your chronicle of the phoenix rebirth is inspiring
(14, 14, 14, 'The griffin’s majesty was captured splendidly.',
(15, 15, 15, 'The leprechauns tricks added a spark of mischief!
(16, 16, 16, 'Your exploration of the deep sea kraken was thril
(17, 17, 17, 'The centaur’s charge was a rush of adrenaline.',
(18, 18, 18, 'Mermaid tales from the deep blue are captivating.
(19, 19, 19, 'Your depiction of the trolls bridge was spot-on.',
(20, 20, 20, 'Goblins and their mischief never cease to amuse!',
```

Views

```
create view user_behaviour as
    select u.user_id, coalesce(r.followers_count, 0) as follower
    from (select distinct user_id from user) u
    left join
        (select destination_id, count(destination_id) as followe
        from user_relationship ur2
        group by destination_id) r on u.user_id = r.destination_
    left join
        (select source_id, count(source_id) as followings_count
        from user_relationship ur
        group by source_id) f on u.user_id = f.source_id

create view content_popularity as
    select
        p.post_id,
        coalesce(l.like_count, 0) as like_count,
```

```
        coalesce (c.comment_count, 0) as comment_count
    from
        post p
    left join
        (select post_id, count(user_id) as like_count
        from post_relationship
        group by post_id) l on p.post_id = l.post_id
    left join
        (select post_id, count(comment_id) as comment_count
        from comment
        group by post_id) c on p.post_id = c.post_id

create view system_analytics as
    select publication_date as post_date, count(post_id) as post
    from post
    group by publication_date
    having count(post_id) >= 2
    order by post_date

INSERT INTO post (post_id, author_id, picture_id, title) VALUES
(1, 1, 1, 'The Hidden Glade'),
(2, 2, 2, 'Secrets of the Arcane'),
(3, 3, 3, 'An Ode to the Forge'),
(4, 4, 4, 'Magic in the Moonlight'),
(5, 5, 5, 'Flight of the Dragons'),
(6, 6, 6, 'War Songs of the Orcs'),
(7, 7, 7, 'Meadow Run at Dawn'),
(8, 8, 8, 'Songs from the Depths'),
(9, 9, 9, 'The Treasure Map'),
(10, 10, 10, 'Whispers of the Old Trees'),
(11, 11, 11, 'The Vampires Ball'),
(12, 12, 12, 'Potions and Hexes'),
(13, 13, 13, 'The Trolls Toll'),
(14, 14, 14, 'Rebirth of the Phoenix'),
(15, 15, 15, 'The Scrolls of Elders'),
(16, 16, 16, 'Mysteries of the Deep'),
```

```
(17, 17, 17, 'Griffin Flight Lessons'),
(18, 18, 18, 'Beneath the Waves'),
(19, 19, 19, 'The Ogres Feast'),
(20, 20, 20, 'The Leprechauns Pot of Gold');
```

Triggers

```
create function get_all_users()
returns decimal(5, 2) deterministic
return (select Count(user_id) from user);

create procedure get_follow_history(user_id int)
select source_id as user, destination_id as user_followed, follo
from user_relationship_history
where source_id = user_id

create procedure update_user_relationship_history(source_id int,
    insert into user_relationship_history (source_id, destinatio
    values (source_id, destination_id, followed, date_time);

create procedure validate_user_relationship_history()
SELECT history.follow_history_id
FROM user_relationship_history history
WHERE history.followed = 0 AND NOT EXISTS (
    SELECT 1
    FROM user_relationship_history previous
    WHERE previous.source_id = history.source_id
    AND previous.destination_id = history.destination_id
    AND previous.followed = 1
    AND previous.relationship_date < history.relationship_date
);

create procedure delete_invalid_user_relationship_history()
DELETE FROM user_relationship_history
WHERE followed = 0 AND follow_history_id IN (
```

```
    SELECT u1.follow_history_id
    FROM user_relationship_history u1
    WHERE u1.followed = 0 AND NOT EXISTS (
        SELECT 1
        FROM user_relationship_history u2
        WHERE u2.source_id = u1.source_id
        AND u2.destination_id = u1.destination_id
        AND u2.followed = 1
        AND u2.relationship_date < u1.relationship_date
    )
);

create trigger update_user_relationship_history_insert
after insert
on user_relationship
for each row
    call update_user_relationship_history(new.source_id, new.des

create trigger update_user_relationship_history_delete
after delete
on user_relationship
for each row
    call update_user_relationship_history(old.source_id, old.des

create trigger validate_user_relationship_history_insert
after insert
on user_relationship_history
for each row
    call delete_invalid_user_relationship_history();

create trigger validate_user_relationship_history_delete
after delete
on user_relationship_history
for each row
    call delete_invalid_user_relationship_history();
```