Project Report

Naqash Hussain
&
Mudasir Mujtaba

May 5th, 2024

—

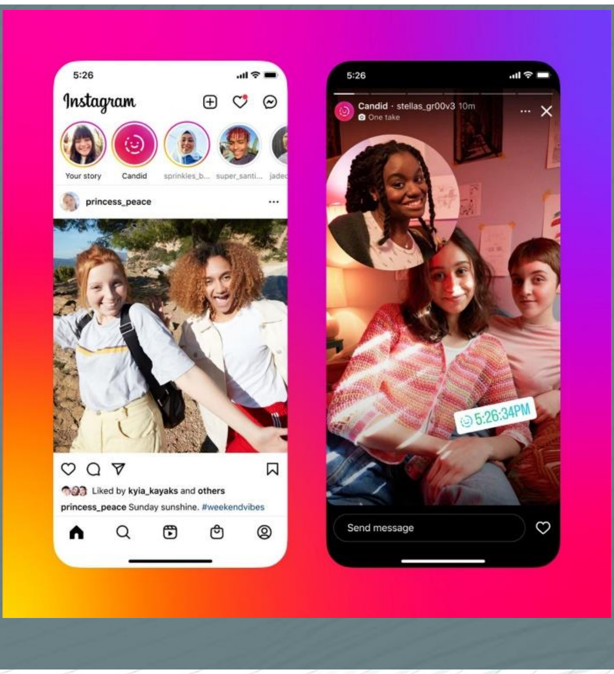
Database System

—

Lecturer Ms. Marina Gul

INTRODUCTION

Most of the time, we use social media and browse extensively. As database students, we thought it would be an interesting idea to build our own version of the famous social media platform Instagram database from scratch, based on our understanding.



PURPOSE

PROBLEM STATEMENT

The objective of this project is to design and implement a robust database system to replicate the functionalities of the Instagram platform. This entails creating an efficient database schema capable of managing user profiles, posts, comments, likes, and posts effectively. Challenges include ensuring scalability, reliability, and optimal performance while accommodating the dynamic nature of social media data. Furthermore, the project aims to develop SQL queries to derive valuable insights from the database, such as user engagement trends and popular content. Addressing these challenges will contribute to a comprehensive understanding of database management systems and their application in the context of social media platforms like Instagram.

Assumptions:

1. Data Consistency: It is assumed that data entered into the Instagram database will be accurate and consistent to maintain the integrity of the system.

2. User Authentication: Users accessing the database are assumed to have valid authentication credentials to ensure data security and privacy.

3. Regular Backup: Regular database backups are assumed to be in place to prevent data loss in case of system failures or emergencies.

Dependencies:

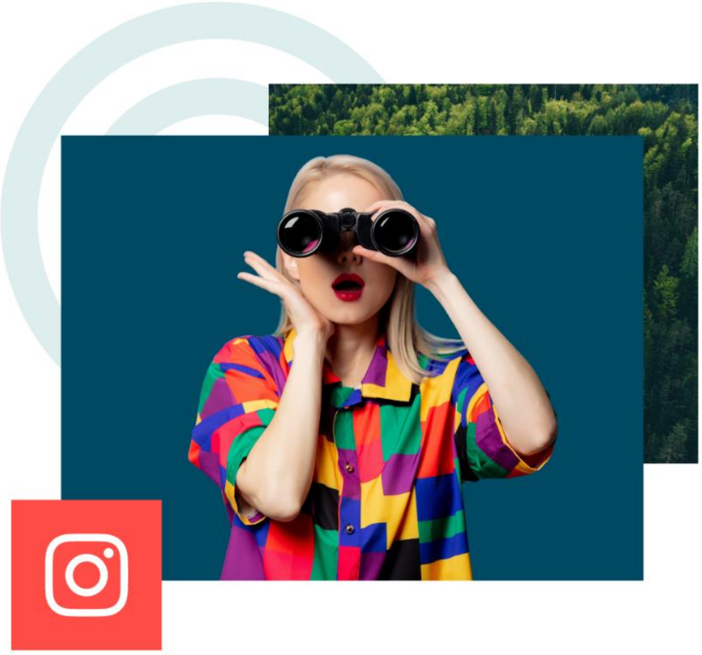
1. Database Management System (DBMS): The project depends on the chosen DBMS MySQL for database creation, management, and querying.

2. Programming Languages: Dependencies on programming languages such as SQL for query development and possibly java for automation tasks are anticipated.

3. Hardware Resources: The system's performance and scalability may depend on the hardware resources available, including memory, processing power, and storage capacity.

User scenario for database

A Journey Through Sarah's Len



A Journey Through Sarah's Lens

Sarah loves taking pictures and sharing them online. She's really into photography and finds beauty all around her. When she signed up for Instagram, it was like opening a door to a world where she could show off her creativity. She filled out her profile like an artist signing a masterpiece, making it uniquely hers.

She uses hashtags to connect with other photographers and artists who are just as passionate as she is. By following them, she gets to see their awesome pictures and share her own. Sarah's Instagram feed is like a gallery of her life, with each post telling a story.

From mountains to beaches, Sarah captures it all through her camera lens, sharing moments that mean a lot to her. Getting likes and comments feels good because it shows that people appreciate her work. But what really matters to her are the personal messages she gets, where she can connect one-on-one with people who share her interests.

Sarah also gives her followers a peek behind the scenes with Instagram stories. She likes adding fun filters and captions to make her posts more interesting.

Sometimes Sarah has to unfollow accounts that don't inspire her anymore. It's important for her to keep her feed filled with things that spark her creativity.

Through Instagram, Sarah isn't just sharing photos – she's creating a colorful world where art and community come together. It's her way of expressing herself and connecting with others who love photography as much as she does.

Functional Requirements

1. Users shall be able to register themselves on Instagram by inserting their (email address, username, password, phone number, first name, last name, date of birth, gender, account type, and bio data).
2. Users shall be able to log in to Instagram by providing their (email address/phone number, username, and password).
3. Users shall be able to follow others and be followed by other users.
4. Users shall be able to use hashtags.
5. Users shall be able to tag other users when uploading their photos/videos.
6. Users shall be able to save posts.
7. Users can upload or share posts in the format of (images, videos) along with optional text.
8. Users shall be able to like, comment on, and share posts.
9. Users shall be able to upload reels in video format.
10. Users shall be able to send/receive messages to/from other users.
11. Users shall receive notifications for likes, comments, and new followers.
12. Users shall be able to upload their stories in the format of images/text/videos.
13. The users shall be able to Unfollow other users.

Non-functional requirements

1. Usernames must be unique and cannot be null.
2. Email addresses must have domains of @gmail.com or @hotmail.com only.
3. Users must provide a unique and valid phone number with the country code.
4. Users should be able to select an account type from three available options: personal, business, creator.
5. Both the phone number/email and password attributes must not be null.
6. Remove the follower attribute if a user is unfollowed by others.
7. Hashtags should begin with the "#" symbol.
8. Tags in posts should include "@" followed by the username.
9. Image files must have extensions of .png, .jpeg, or .jpg only.
10. Video files must have extensions of .mp4 or .mov only.
11. The size of the reels should be 1080 x 1920 pixels.
12. Instagram story size should be 1080 x 1920 pixels.

Database Schema Design
Possible tables and Attributes.

1. Users

Columns:

user_id
email_address
username
password
phone_number
first_name
last_name
date_of_birth
gender
account_type
bio_data

2. Posts

Columns:

post_id
post_type (image or video)
post_content
upload_date

3. Followers

Columns:

follower_id

4. Hashtags

Columns:

hashtag_id
hashtag_name

5. Post_Hashtags

Columns:

6. Likes

Columns: like_id

7. Comments

Columns:

comment_id
comment_text
comment_date

8. Messages

Columns:

message_id
message_content
message_date

9. Notifications

Columns:

notification_id
notification_type
notification_content
notification_date

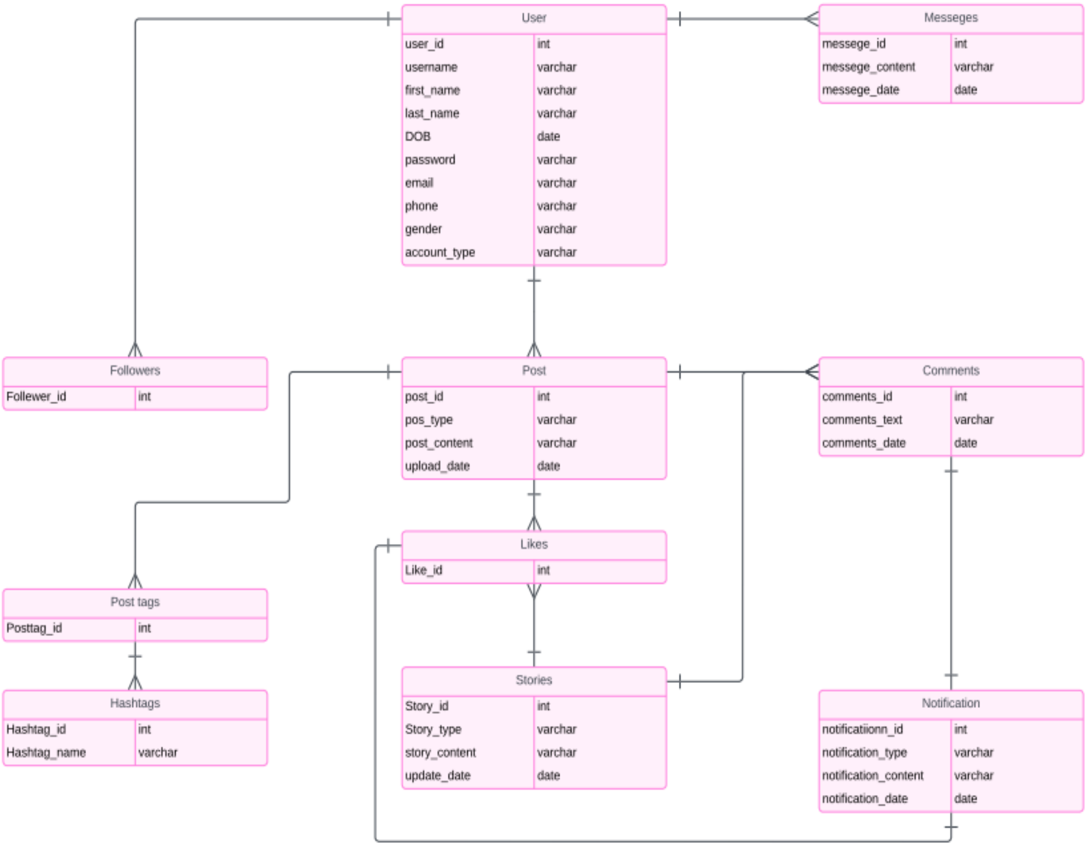
10. Stories

Columns:

story_id
story_content
upload_date

Diagrams

First Entity Relation Diagram



User	
user_id	int
username	varchar
first_name	varchar
last_name	varchar
DOB	date
password	varchar
email	varchar
phone	varchar
gender	varchar
account_type	varchar

Messeges	
messege_id	int
messege_content	varchar
messege_date	date

Followers	
Follewer_id	int

Post	
post_id	int
pos_type	varchar
post_content	varchar
upload_date	date

Comments	
comments_id	int
comments_text	varchar
comments_date	date

Likes	
Like_id	int

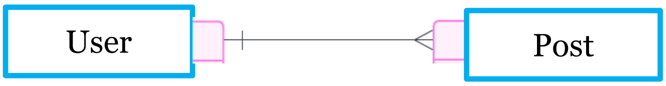
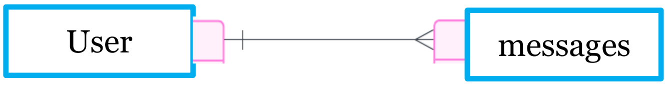
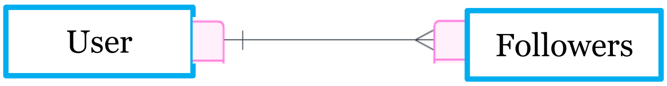
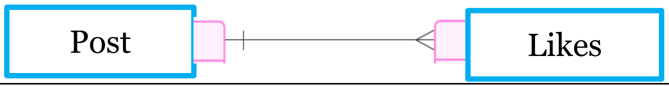
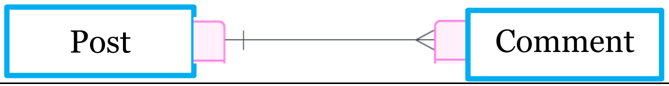
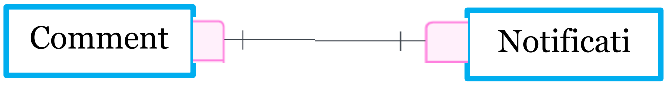
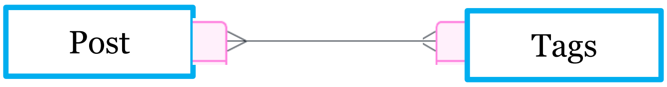
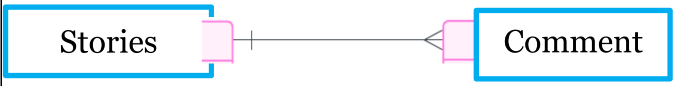
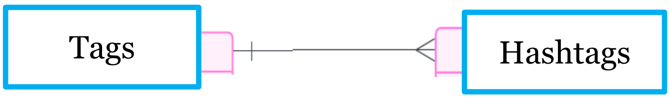
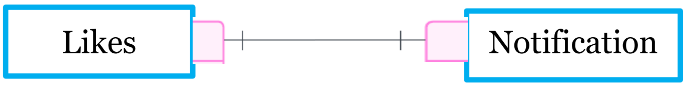
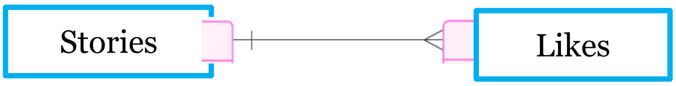
Post tags	
Posttag_id	int

Hashtags	
Hashtag_id	int
Hashtag_name	varchar

Stories	
Story_id	int
Story_type	varchar
story_content	varchar
update_date	date

Notification	
notificatiionn_id	int
notification_type	varchar
notification_content	varchar
notification_date	date

Translation of Cardinalities

	Each User can post multiple posts, while Each post belongs to one specific user.
	Each User can send multiple messages, while Each message belongs to one specific user.
	A single user can have many followers, while Each follower record is associated with one user they follow.
	A post can be liked by many users, while Each like record indicates which post it's for.
	A post can be the subject of many comments, while Each comment is tied to a single post.
	Each comment is associated with only one notification and viceversa.
	A single post can be associated with many tags, while A single tag can be applied to many posts.
	Each story can have many comments, while each comment is associated with only one story.
	Each posttags can be multiple hashtags, while hashtag is only associated with only one posttags
	Each like can have one notification, while notification can be for the one like.
	Each story can have many likes, while each like is associated with only one story.

New revised Tables
Relational Schema

- User (user_id, user_name, first_name, last_name, dob, email, password, phone, gender, account_type)
  Primary key: user_id

- Messages (message_id, message_content, message_date, sender_user_id, receiver_user_id)
  Primary key: message_id
  Foreign key: sender_user_id, receiver_user_id

- Post (post_id, post_type, post_content, upload_date, user_id)
  Primary key: post_id
  Foreign key: user_id

- Followers (follower_id, follower_user_id)
  Primary key: follower_id
  Foreign key: follower_user_id

- Likes (like_id, user_id)
  Primary key: like_id
  Foreign key: post_id

- Comments (comment_id, comment_text, comments_date, post_id, story_id)
  Primary key: comment_id
  Foreign key: post_id, story_id

- Notifications (notification_id, notification_type, notification_content, notification_date, comment_id, like_id)
  Primary key: notification_id
  Foreign key: comment_id, like_id

- Stories (story_id, story_type, story_content, update_date)
  Primary key: story_id

- Tags (tag_id, tag_name, tag_date)
  Primary key: tag_id

- Posttags (post_tag_id, post_id, tag_id)
  primary key: post_tag_id
  Foreign key: post_id, tag_id

- Hashtag (hash_tag_id, hash_tag_name, post_tag_id)
  Primary key: hash_tag_id
  Foreign key: post_tag_id

Sample Data collection

Users

message_id	message_content	message_date	sender_user_id	receiver_user_id
1	Hello!	2024-04-01	1	2
2	Hi there	2024-04-02	2	1
3	How are you?	2024-04-03	3	1

Messages

Post

post_id	post_type	post_content	upload_date	user_id
1	Text	Hello World!	2024-04-01	1
2	Image	[Image URL]	2024-04-02	2
3	Video	[Video URL]	2024-04-03	3

Followers

follower_id	follower_user_id
1	1
2	2
3	3

Likes

like_id	user_id
1	1
2	2
3	3

Comments

comment_id	comment_text	comments_date	story_id	post_id
1	Nice post!	2024-04-01	1	NULL
2	Love it!	2024-04-02	NULL	2
3	Great video!	2024-04-03	3	NULL

Notification

notification_id	notification_type	notification_content	notification_date	comment_id	like_id
1	Like	John Doe liked your post	2024-04-25 10:20:00	NULL	1
2	Comment	Jane Doe commented on your post	2024-04-27 18:50:00	2	NULL

Stories

story_id	story_type	story_content	update_date
1	Text	Story text goes here	2024-04-25 10:00:00
2	Image	[Image file]	2024-04-26 14:30:00
3	Video	[Video file]	2024-04-27 18:45:00

Tags

tag_id	tag_name	tag_date
1	nature	2024-04-25 10:15:00
2	travel	2024-04-26 14:35:00
3	fun	2024-04-27 18:50:00

Post Tag

post_tag_id	post_id	tag_id
1	1	1
2	2	2
3	3	3

Hash tag

hash_tag_id	hash_tag_name	post_tag_id
1	beach	1
2	mountains	2
3	adventure	3

Normalization

Functional Dependencies

User Table

user_id > user_name, first_name, last_name, dob, email, password, phone, gender, account_type

Candidate Key: user_id
Primary Key: user_id

Messages Table

message_id > message_content, message_date, sender_user_id, receiver_user_id
sender_user_id > user_name, first_name, last_name, dob, email, password, phone, gender, account_type
receiver_user_id > user_name, first_name, last_name, dob, email, password, phone, gender, account_type

Candidate Key: message_id
Primary Key: message_id

Post Table

post_id > post_type, post_content, upload_date, user_id
user_id > user_name, first_name, last_name, dob, email, password, phone, gender, account_type

Candidate Key: post_id
Primary Key: post_id

Followers Table

follower_id > follower_user_id
follower_user_id > user_name, first_name, last_name, dob, email, password, phone, gender, account_type

Candidate Key: follower_id
Primary Key: follower_id

Likes Table
like_id > user_id
user_id > user_name, first_name, last_name, dob, email, password, phone, gender, account_type

Candidate Key: like_id
Primary Key: like_id

Comments Table
(comment_id, story_id) > comment_text, comments_date, user_id
user_id > user_name, first_name, last_name, dob, email, password, phone, gender, account_type
story_id > post_id, post_type, post_content, upload_date

Candidate Key: (comment_id)
Primary Key: comment_id

Notifications Table
notification_id > notification_type, notification_content, notification_date, comment_id, like_id

Candidate Key: notification_id
Primary Key: notification_id

Stories Table
story_id > story_type, story_content, update_date

Candidate Key: story_id
Primary Key: story_id

Tags Table
tag_id > tag_name, tag_date

Candidate Key: tag_id
Primary Key: tag_id

Posttags Table
post_tag_id > post_id, tag_id
post_id > post_type, post_content, upload_date, user_id
tag_id > tag_name, tag_date

Candidate Key: post_tag_id
Primary Key: post_tag_id

Hashtag Table
hash_tag_id > hash_tag_name, post_tag_id

Candidate Key: hash_tag_id
Primary Key: hash_tag

1NF (First Normal Form)

First, let's ensure that each **table** has atomic values and no repeating groups.

User Table: No repeating groups. Each attribute contains atomic value.
Messages Table: No repeating groups. Each attribute contains atomic values.
Post Table: No repeating groups. Each attribute contains atomic values.
Followers Table: No repeating groups. Each attribute contains atomic values.
Likes Table: No repeating groups. Each attribute contains atomic values.
Comments Table: No repeating groups. Each attribute contains atomic values.
Notifications Table: No repeating groups. Each attribute contains atomic values.
Stories Table: No repeating groups. Each attribute contains atomic values.
Tags Table: No repeating groups. Each attribute contains atomic values.
Posttags Table: No repeating groups. Each attribute contains atomic values.
Hashtag Table: No repeating groups. Each attribute contains atomic values.

2NF (Second Normal Form)

In the 2NF, we ensure that nonkey attributes are fully dependent on the primary key.

User Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (user_id).
Messages Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (message_id).
Post Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (post_id).
Followers Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (follower_id).
Likes Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (like_id).
Comments Table: Already in 2NF. All nonkey attributes are fully dependent on the composite primary key (comment_id, story_id).
Notifications Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (notification_id).
Stories Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (story_id).
Tags Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (tag_id).
Posttags Table: Already in 2NF. All nonkey attributes are fully dependent on the composite primary key (post_tag_id, post_id, tag_id).
Hashtag Table: Already in 2NF. All nonkey attributes are fully dependent on the primary key (hash_tag_id).

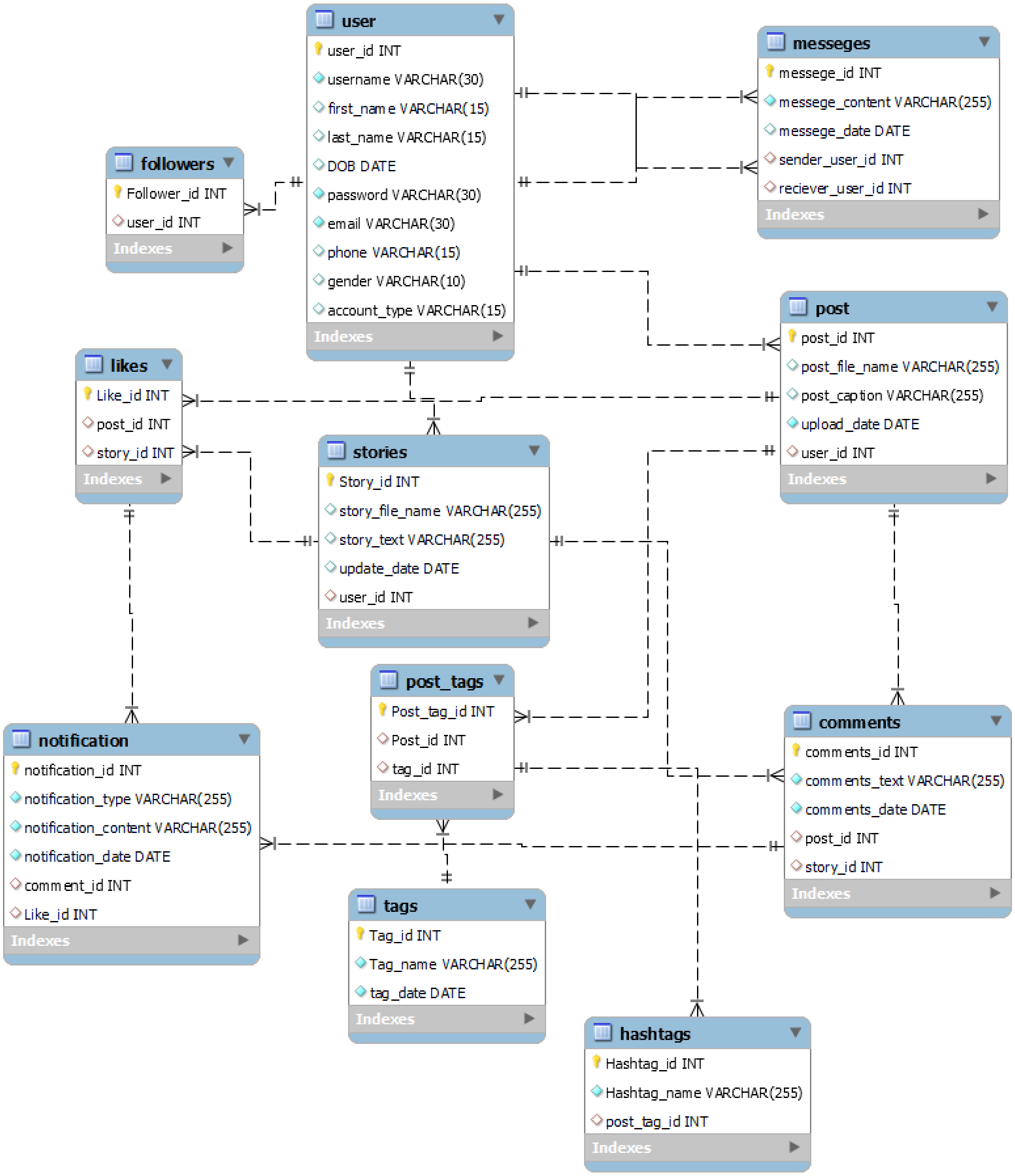
3NF (Third Normal Form)

In the 3NF, we ensure that there are no transitive dependencies.

User Table: Already in 3NF. No transitive dependencies exist.
Messages Table: Already in 3NF. No transitive dependencies exist.
Post Table: Already in 3NF. No transitive dependencies exist.
Followers Table: Already in 3NF. No transitive dependencies exist.
Likes Table: Already in 3NF. No transitive dependencies exist.
Comments Table: Already in 3NF. No transitive dependencies exist.
Notifications Table: Already in 3NF. No transitive dependencies exist.
Stories Table: Already in 3NF. No transitive dependencies exist.
Tags Table: Already in 3NF. No transitive dependencies exist.
Posttags Table: Already in 3NF. No transitive dependencies exist.

Hashtag Table: Already in 3NF. No transitive dependencies exist.

Final ERD (Entity Relationship Diagram)



Implementation

```
Create Database if not exists Instagram;

CREATE TABLE User (
   user_id  int Primary Key,
   username   varchar(30) unique not null ,
   first_name   varchar(15),
   last_name   varchar(15),
   DOB  date,
   password   varchar(30) not null,
   email   varchar(30) unique not null,
   phone   varchar(15) unique,
   gender   varchar(10),
   account_type   varchar(15) default 'Personal'
);


CREATE TABLE Post (
  post_id INT PRIMARY KEY,
  post_type VARCHAR(255) default 'text',
  post_content VARCHAR(255) not null,
  upload_date DATE not null,
  user_id INT,
 constraint User_id_FK FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE TABLE Messeges (
  messege_id INT PRIMARY KEY,
  messege_content VARCHAR(255) not null,
  messege_date DATE,
  sender_user_id INT,
  reciever_user_id INT,
  FOREIGN KEY (sender_user_id) REFERENCES User(user_id) ON DELETE CASCADE ON
UPDATE CASCADE,
  FOREIGN KEY (reciever_user_id) REFERENCES User(user_id) ON DELETE CASCADE ON
UPDATE CASCADE
);

CREATE TABLE Followers (
  Follower_id INT PRIMARY KEY,
  user_id INT,
  FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE Stories (
  Story_id INT PRIMARY KEY,
  Story_type VARCHAR(255) default 'image',
  story_content VARCHAR(255) not null,
  update_date DATE
```

```
);

CREATE TABLE Comments (
  comments_id INT PRIMARY KEY,
  comments_text VARCHAR(255) not null,
  comments_date DATE not null,
  post_id INT,
  story_id INT,
  FOREIGN KEY (post_id) REFERENCES Post(post_id) ON DELETE CASCADE ON UPDATE
CASCADE,
  FOREIGN KEY (story_id) REFERENCES Stories(story_id) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE Likes (
  Like_id INT PRIMARY KEY,
  post_id INT,
  story_id INT,
  FOREIGN KEY (post_id) REFERENCES Post(post_id) ON DELETE CASCADE ON UPDATE
CASCADE,
  FOREIGN KEY (story_id) REFERENCES Stories(story_id) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE Notification (
  notification_id INT PRIMARY KEY,
  notification_type VARCHAR(255) not null,
  notification_content VARCHAR(255) not null,
  notification_date DATE not null,
  comment_id INT,
  Like_id INT,
  FOREIGN KEY (comment_id) REFERENCES Comments(comments_id) ON DELETE CASCADE
ON UPDATE CASCADE,
  FOREIGN KEY (Like_id) REFERENCES Likes(Like_id) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE Tags (
  Tag_id INT PRIMARY KEY,
  Tag_name VARCHAR(255) not null,
  tag_date DATE not null
);

CREATE TABLE Post_tags (
  Post_tag_id INT PRIMARY KEY,
  Post_id INT,
  tag_id INT,
  FOREIGN KEY (Post_id) REFERENCES Post(post_id) ON DELETE CASCADE ON UPDATE
CASCADE,
  FOREIGN KEY (tag_id) REFERENCES Tags(Tag_id) ON DELETE CASCADE ON UPDATE
CASCADE
);

CREATE TABLE Hashtags (
  Hashtag_id INT PRIMARY KEY,
  Hashtag_name VARCHAR(255) not null,
  post_tag_id INT,
  FOREIGN KEY (post_tag_id) REFERENCES Post_tags (Post_tag_id) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

TESTING

- Retrieve all users who have a premium account type.

```
SELECT * FROM User WHERE account_type = 'Premium';
```

- Retrieve all posts uploaded by a specific user (e.g., user with user_id = 3).

```
SELECT * FROM Post WHERE user_id = 3;
```

- Retrieve all messages sent by a specific user (e.g., user with user_id = 1).

```
SELECT * FROM Messages WHERE sender_user_id = 1;
```

- Retrieve all followers of a specific user (e.g., user with user_id = 7).

```
SELECT * FROM Followers WHERE user_id = 7;
```

- Retrieve all stories uploaded after a certain date (e.g., '2024-04-05').

```
SELECT * FROM Stories WHERE update_date > '2024-04-05';
```

- Retrieve all comments made on a specific post (e.g., post with post_id = 3)

```
SELECT * FROM Comments WHERE post_id = 3;
```

- Retrieve all likes on a specific story (e.g., story with story_id = 1)

```
SELECT * FROM Likes WHERE story_id = 1;
```

- Retrieve all notifications of type 'Like' after a certain date (e.g., '2024-04-03').

```
SELECT * FROM Notification WHERE notification_type = 'Like' AND notification_date > '2024-04-
03';
```

- Retrieve all tags associated with a specific post (e.g., post with post_id = 6).

```
SELECT Tags.Tag_name FROM Tags INNER JOIN Post_tags ON Tags.Tag_id = Post_tags.tag_id
WHERE Post_tags.Post_id = 6;
```

- Retrieve all posts associated with a specific hashtag (e.g., hashtag with hashtag_name = 'SampleHashtag3').

```
SELECT Post.post_id FROM Post INNER JOIN Post_tags ON Post.post_id = Post_tags.Post_id
INNER JOIN Tags ON Post_tags.tag_id = Tags.Tag_id INNER JOIN Hashtags ON
Post_tags.Post_tag_id = Hashtags.post_tag_id WHERE Hashtags.Hashtag_name =
'SampleHashtag3';
```

- Retrieve all users who have not uploaded any posts.

```
SELECT * FROM User WHERE user_id NOT IN (SELECT DISTINCT user_id FROM Post);
```

- Retrieve all posts uploaded on a specific date (e.g., '2024-04-04').

```
SELECT * FROM Post WHERE upload_date = '2024-04-04';
```

- Retrieve all messages exchanged between two specific users (e.g., user_id 1 and user_id 3).

```
SELECT * FROM Messages WHERE (sender_user_id = 1 AND receiver_user_id = 3) OR
(sender_user_id = 3 AND receiver_user_id = 1);
```

- Retrieve all followers of users who have a premium account type.

```
SELECT * FROM Followers WHERE user_id IN (SELECT user_id FROM User WHERE account_type =
'Premium');
```

- Retrieve all stories uploaded by users with a regular account type.

```
SELECT * FROM Stories WHERE Story_id IN (SELECT Story_id FROM User WHERE account_type =
'Regular');
```