Blogging Website for Diverse Voices

Database Project Report

Group-G5

Report submitted November, 2024

A project submitted to Dr. Rudra Pratap Deb Nath, Associate Professor, Department of Computer Science and Engineering, Chittagong University (CU) in partial fulfillment of the requirements for the Database Systems Lab course. The project is not submitted to any other organization at the same time.

Table 1: Details of Group-G5

Roll Id	Name	Sigature	Date	Supervisor Approval
22701035	Aryan Bin Ashraf	Aryan		

Contents

1 Requirement Gathering and Analysis

For the accomplishment of a project successfully, it is very important to gather all the requirements and analyze them properly. That is why, before beginning our work, I first gathered our requirements and analyzed them. Gathering requirements for the **Blogging Website for Diverse Voices** could be done by several methods such as documentation review, interviews with stakeholders, surveys, discussions, workshops, etc. Among these, I have chosen to use the web as a resource and discuss with people who are interested in sharing their perspective with others and get to know about some interesting topics. I have talked with some of my friends who are interested in blogging and some of them are frequent readers of blogs.

1.1 Process of Gathering Requirements

1.1.1 Discussion with friends

Below are some key points from the discussion.

- **Question 1: What are some challenges you face when sharing your perspectives online?**
  Friend: A big challenge is finding a platform that allows different voices and viewpoints. Many platforms cater to specific topics or groups, so it's hard to reach a wide range of people. It's also tough to make sure the community is friendly and inclusive.

- **Question 2: How important is it for you to receive reactions and comments on your posts?**
  Friend: Very important! Reactions help me understand how my posts are resonating with readers. Comments are especially valuable because they allow for more in-depth feedback and discussion. Sometimes the comments are as engaging as the post itself, so it's essential that the platform encourages constructive conversations.

- **Question 3: How would you like to connect with other bloggers or readers on the platform?**
  Friend: A feature to follow or connect with other users would be great. This way, we can build a community and keep track of other creators' new content.

1.1.2 Web as a Resource for Collecting Information on Blogging Websites

We collected information from several online resources to understand the essential features, user engagement strategies, and security practices for creating a blogging website. The following resources provided valuable insights:

- **HubSpot** – *What is a Blog?*
  https://blog.hubspot.com/marketing/what-is-a-blog

- **Infidigit** – *What is a Blog and Why Blogs are Important?*
  https://www.infidigit.com/blog/what-is-a-blog/

The information we collected from the above-mentioned resources is given below:

- **What is a Blog?**
  A blog is an online platform where individuals or groups share information, ideas, and personal insights on a specific topic.
- **Why is Blogging Important?**
  Blogging is a valuable tool for sharing knowledge, building personal or brand authority, and engaging with an audience. It fosters communication and allows readers to interact through comments, reactions, and sharing options.
- **What Features Should a Blogging Website Include?**
  A successful blogging platform should offer features like user profiles, post creation tools, and the ability to react and comment on posts.
- **What Are Key Features for User Engagement?**
  Blogging websites benefit from interactive features such as likes, shares, comments, and follow options. These features help build a community and keep readers engaged.
- **Is it Safe to Use a Blog Site?**
  Yes, if the website has security features like data encryption, password protection, and regular updates, it's generally safe for users.

1.2 Recognizing Stakeholders

For a blogging website, the stakeholders typically include:

- **Bloggers (Content Creators):** Individuals who create and publish blog posts. They rely on the platform's tools for writing.
- **Readers (Audience):** People who visit the site to read blog posts. They may interact with content by liking, commenting, or sharing posts and can also follow favorite bloggers.
- **Developers and Designers:** The technical team responsible for creating, maintaining, and updating the website. They ensure the site runs smoothly, looks appealing, and provides a good user experience.
- **Advertisers and Sponsors:** Brands or businesses that pay for ad placements on the site or sponsor content. They provide a source of revenue for the platform and may collaborate with bloggers for promotions.

1.3 Discovering Requirements

1.3.1 Key functional requirements for the platform include:

- **User Registration and Profile Management:** Let users create and manage profiles.
- **Content Creation and Editing:** Provide a rich text editor for creating content and adding media.
- **Social Interaction:** Allow users to comment, like, and follow content creators to encourage community engagement.

1.3.2 Important non-functional requirements for the system include:

- **System Reliability and Availability:** Ensure high uptime and smooth performance.
- **Data Security and Privacy:** Implement security measures to protect user data and privacy.
- **User-Friendly Interface:** Design an easy-to-use interface accessible to users of all technical levels.
- **Cross-Platform Compatibility:** Make sure the platform works across different devices and browsers.

2 Conceptual Modelling

Conceptual modeling in database design is the process of creating a high-level representation of real-life objects and the relationships between them by translating user requirements. It defines the structure of a database and the relationships between different data elements. For conceptual design, we use a popular representation called the Entity-Relationship Model. Using the Entity-Relationship model, the conceptual schema specifies the entities represented in the database, the attributes of the entities, the relationships among the entities, and the constraints on entities and relationships.

2.1 Entity-Relationship Model

Entity Set

An **entity** is a real-world object distinct from all other things. For instance, every doctor at a hospital is a separate entity. An **entity set** is a collection of entities that belong to the same type and have similar properties or characteristics. For example, the entity set *doctor* is a collection of all individuals who work as doctors at a certain hospital. A group of entity sets may be contained in a database. They are represented by a rectangular shape.

Attributes

Each member of an entity set possesses descriptive properties known as **attributes**. For illustration, the *doctor* entity set may have properties such as *ID*, *name*, *salary*, *specialization*, *highest degree*, and *email.* Attributes can be categorized as follows:

1. **Simple Attribute:** A simple attribute is one that cannot be separated into other components.

2. **Composite Attribute:** A composite attribute has many components. For instance, *name* may be divided into *first name*, *middle name*, and *last name.*

3. **Single-Valued Attribute:** A single-valued attribute is one that only accepts one value per instance of the corresponding entity. Example: A student's *age.*

4. **Multi-Valued Attribute:** A multi-valued attribute stores several values for each instance of an object. Example: A student's *contact information.*

5. **Derived Attribute:** A derived attribute is one that has been computed or derived using other characteristics in the database.

Relationship Set

An **association** among different entities is referred to as a **relationship**. For example, a relationship *treats* may connect a doctor and a patient. A collection of connections of the same kind is referred to as a **relationship set**. There are four kinds of relationships in the Entity-Relationship diagram:

1. **One-to-One**
2. **One-to-Many**
3. **Many-to-One**
4. **Many-to-Many**

2.2 Entity Sets and Tables

(a) User Table

Attributes:

- **userid**: Primary Key
- **username**
- **name**
- **email**
- **password**
- **profilepic**
- **city**
- **website**

(b) Post Table

Attributes:

- **postid**: Primary Key
- **desc**
- **img**
- **createdAt**
- **userid**: Foreign Key referencing **User**

(c) Comments Table

Attributes:

- **commentsid**: Primary Key
- **desc**
- **createdAt**
- **userid**: Foreign Key referencing **User**
- **postid**: Foreign Key referencing **Post**

(d) Like Table

Attributes:

- **postid**: Foreign Key referencing **Post**
- **userid**: Foreign Key referencing **User**

(e) Followers Table

Attributes:

- **followerUserid**: Foreign Key referencing **User**
- **followedUserid**: Foreign Key referencing **User**

(f) Reply Table

Attributes:

- **replyid**: Primary Key
- **desc**
- **createdAt**
- **userid**: Foreign Key referencing **User**
- **commentsid**: Foreign Key referencing **Comments**

2.3 Relationship Set

User - Post Relationship

Description: Each user can create multiple posts, but each post is created by only one user.
Relationship Type: One-to-Many (1:N)
Foreign Key: `userid` in **Post** referencing **User**

User - Comments Relationship

Description: Each user can write multiple comments, but each comment is written by only one user.
Relationship Type: One-to-Many (1:N)
Foreign Key: `userid` in **Comments** referencing **User**

Post - Comments Relationship

Description: Each post can have multiple comments, but each comment is associated with only one post.
Relationship Type: One-to-Many (1:N)
Foreign Key: `postid` in **Comments** referencing **Post**

User - Like Relationship

Description: Each user can like multiple posts, and each post can be liked by multiple users.
Relationship Type: Many-to-Many (M:N)
Foreign Key: `userid` and `postid` in **Like** referencing **User** and **Post**

Post - Like Relationship

Description: Each post can receive multiple likes.
Relationship Type: One-to-Many (1:N)
Foreign Key: `postid` in **Like** referencing **Post**

User - Followers Relationship

Description: Each user can follow multiple other users, and each user can be followed by multiple users.
Relationship Type: Many-to-Many (M:N)
Foreign Key: `followerUserid` and `followedUserid` in **Followers** referencing **User**

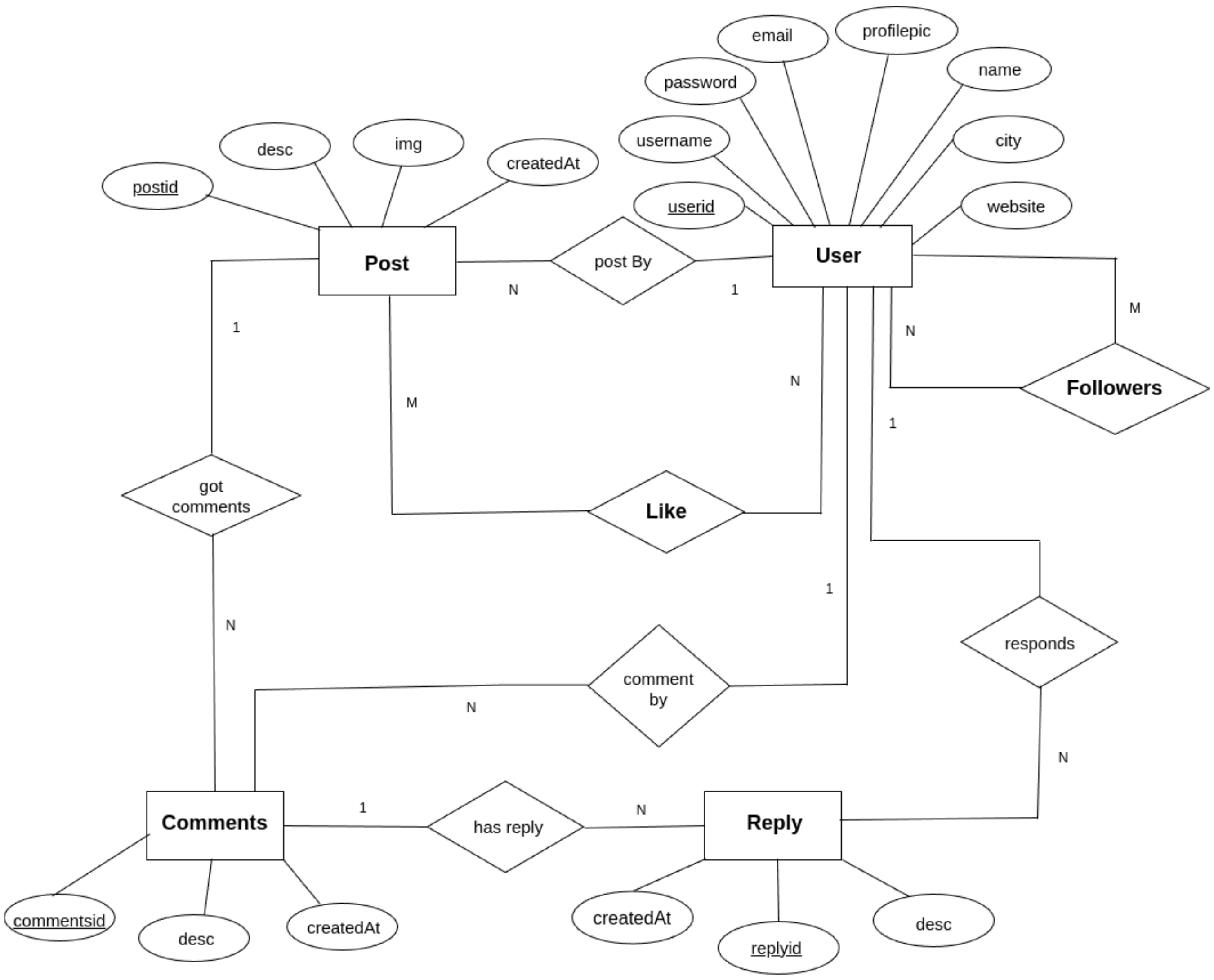
User - Reply Relationship

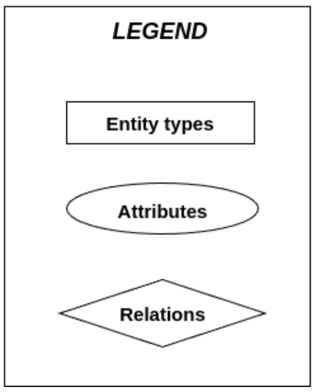
Description: Each user can write multiple reply to comments, but each reply of a comment is written by only one user.
Relationship Type: One-to-Many (1:N)
Foreign Key: `userid` in **Reply** referencing **User**

Comments - Reply Relationship

Description:Each comment can have multiple replies, but each reply is associated with only one comment.
Relationship Type: One-to-Many (1:N)
Foreign Key: `commentsid` in **Reply** referencing **Comments**

2.4 ER Diagram





3 Logical Modelling

After mapping onto the conceptual model in the previous section, I am going to map our logical data model. The structure of the data items and their relationships are specified using a logical data model. The logical model can be mapped in various ways. However, for our system, we are utilizing the relational model.

3.1 Relational Model

In a relational model, a group of linked tables serves as a representation of the data and relationships. Each table consists of a collection of columns and rows, where each column denotes an attribute of an entity and each row a record. The relational schema describes how a relation is organized and is composed of the relation name and a list of attribute names.

3.2 ER Model to Relational Schema

I followed the following steps for mapping our relational schema:

1. Each entity type will become a table and each attribute of the entity type will turn into a column.
2. For each table, the primary key will be identified. Any table row must be uniquely identified by the primary key. The primary key can be a single or set of attributes.
3. If a many-to-many relationship occurs between entity types, then the relationship will become a table whose primary key is a composite of the primary keys of the linked tables or entities. If the relationship has its own attribute then it will be added as a column.

3.3 List of Relational Schema for Our System

- **User**
  `User(userid, username, name, password , email, profilepic, city, website)`
  Primary Key: userid
- **Post**
  `Post(postid, desc, img, createdAt, userid)`
  Primary Key: postid
  Foreign Key: userid → User(userid)

- **Comments**
  `Comments(commentsid, desc, createdAt, userid, postid)`
  Primary Key: commentsid
  Foreign Keys:
  userid → User(userid)
  postid → Post(postid)

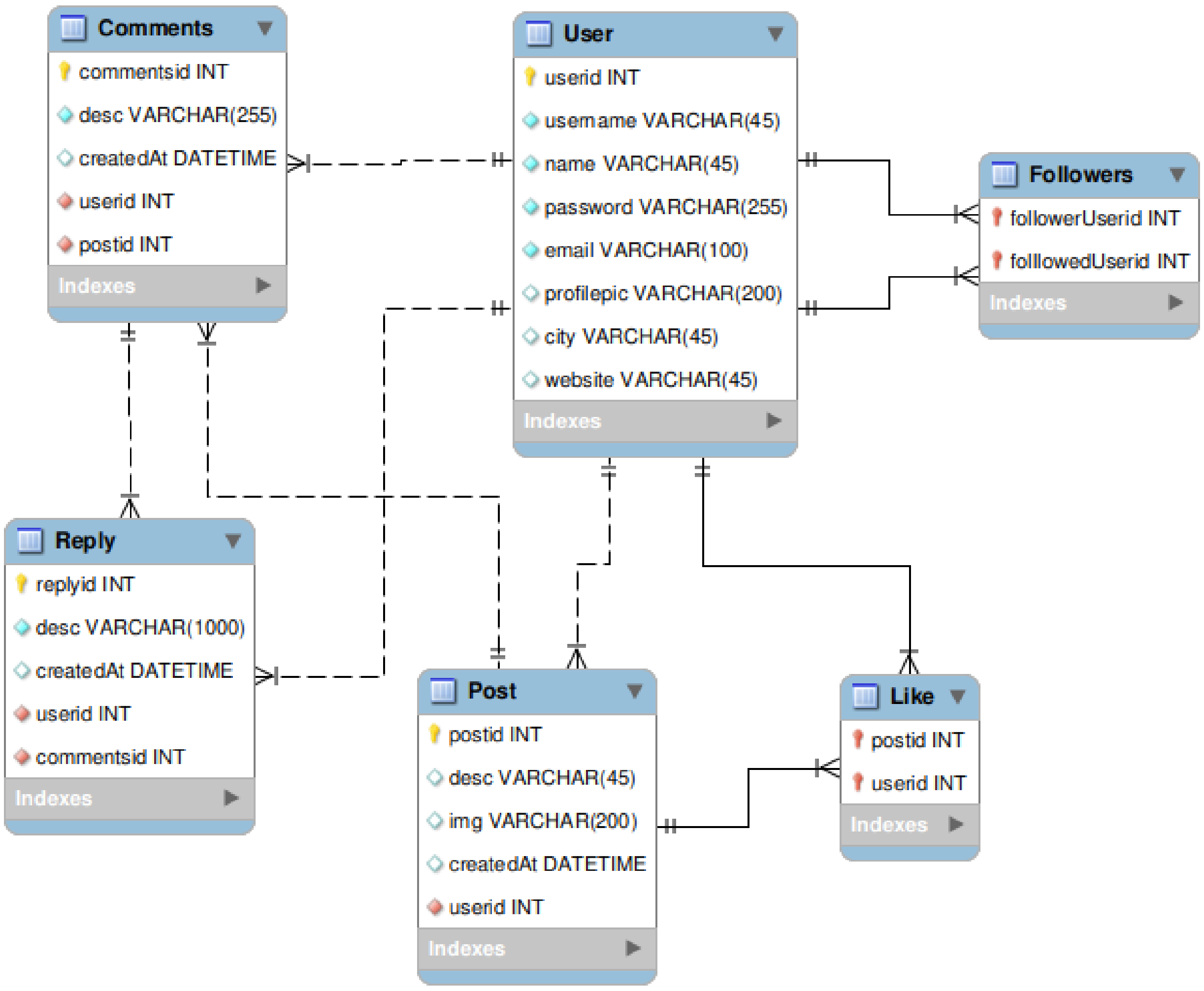
- **Like**
  `Like(postid, userid)`
  Primary Key: (postid,userid)
  Foreign Keys:
  postid → Post(postid)
  userid → User(userid)

- **Followers**
  `Followers(followerUserid, followedUserid)`
  Primary Key: (followerUserid,followedUserid)
  Foreign Keys:
  followerUserid → User(userid)
  followedUserid → User(userid)

- **Reply**
  `Reply(replyid, desc, createdAt, userid, commentsid)`
  Primary Key: (replyid)
  Foreign Keys:
  userid→ User(userid)
  commentsid → Comments(commentsid)

3.4 Relational Schema Diagram

The relational model of my system is given below.



4 Normalization

Normalization is a process used to minimize or remove data redundancy from a set of relations, aiming to eliminate anomalies. Anomaly refers to a database's fault or inconsistency, which may occur during insert, update, or delete operations. We follow normalization rules to make our database redundancy-less, lossless, and to preserve dependencies. There are many types of normal forms in the normalization process.

Normalization Rules

Normalization rules can be divided into the following categories:

- **First Normal Form:**
  - It should only have single (atomic) valued attributes/columns.
  - Values stored in a column should be of the same domain.
  - All the columns in a table should have unique names.
  - The order in which data is stored does not matter.
- **Second Normal Form:**
  - It should be in the First Normal form.
  - It should not have Partial Dependency.
- **Third Normal Form:**
  - It should be in the Second Normal form.
  - It should not have Transitive Dependency.

4.1 Normalizing User Table

Relational Schema: User(userid, username, name, password, email, profilepic, city, website)
Functional Dependency:

- userid → { username, name, password, email, profilepic, city, website }
- email → { userid, username, name, password, profilepic, city, website }

Candidate Key: { userid, email }
Prime Attributes: { userid, email }
Non-Prime Attributes: { username, name, password, profilepic, city, website }

Data in User Table

userid	username	name	password	email	profilepic	city	website
1	jdoe	John Doe	hashed_password_1	jdoe@example.com	profile1.jpg	New York	johndoe.com
2	asmith	Alice Smith	hashed_password_2	asmith@example.com	profile2.jpg	Los Angeles	None
3	mlee	Mark Lee	hashed_password_3	mlee@example.com	None	Chicago	marklee.com

Table 2: Data in User Table

Normalization Steps

1NF: Table contains only atomic values, so it is in 1NF.
2NF: No partial dependency exists, because a proper subset of the candidate key doesn't determine any non-prime attribute. Therefore, the table is in 2NF.
3NF: No transitive dependency exists, because each non-prime attribute is fully functionally dependent on the candidate key. Therefore, the table is in 3NF.

Normal Form	Normalized?
1NF	YES
2NF	YES
3NF	YES

Table 3: Normalization of User table

4.2 Normalizing Post Table

Relational Schema: Post(postid, desc, img, createdAt, userid)
Functional Dependency:

- postid → { desc, img, createdAt, userid }

Candidate Key: { postid }
Prime Attribute: { postid }
Non-Prime Attributes: { desc, img, createdAt, userid }

Data in Post Table

postid	desc	img	createdAt	userid
1	First blog post	image1.jpg	2024-11-14 10:30:00	1
2	Exploring the city	image2.jpg	2024-11-14 12:45:00	2
3	A day in the life	image3.jpg	2024-11-14 15:00:00	3

Table 4: Data in Post Table

Normalization Steps

1NF: Table contains only atomic values, so it is in 1NF.
2NF: No partial dependency exists because there is only one candidate key (‘postid‘). Therefore, the table is in 2NF.
3NF: No transitive dependency exists because each non-prime attribute is fully dependent on the candidate key (‘postid‘). Therefore, the table is in 3NF.

Normal Form	Normalized?
1NF	YES
2NF	YES
3NF	YES

Table 5: Normalization of Post table

4.3 Normalizing Comments Table

Relational Schema: Comments(commentsid, text, createdAt, postid, userid)
Functional Dependency:

- commentsid → { text, createdAt, postid, userid }

Candidate Key: { commentsid }
Prime Attribute: { commentsid }
Non-Prime Attributes: { text, createdAt, postid, userid }

Data in Comments Table

commentsid	desc	createdAt	postid	userid
1	Great post! Very informative.	2024-11-14 10:45:00	1	2
2	Loved this article. Keep it up!	2024-11-14 12:50:00	2	3
3	Very inspiring, thank you for sharing!	2024-11-14 15:05:00	3	1

Table 6: Data in Comments Table

Normalization Steps

1NF: Table contains only atomic values, so it is in 1NF.
2NF: No partial dependency exists because there is only one candidate key (‘commentsid‘). Therefore, the table is in 2NF.
3NF: No transitive dependency exists because each non-prime attribute is fully dependent on the candidate key (‘commentsid‘). Therefore, the table is in 3NF.

Normal Form	Normalized?
1NF	YES
2NF	YES
3NF	YES

Table 7: Normalization of Comments table

4.4 Normalizing Like Table

Relational Schema: Like(postid, userid)

Functional Dependency:

- { postid, userid } → all attributes

Candidate Key: { postid, userid }
Prime Attributes: { postid, userid }
Non-Prime Attributes: None

Data in Like Table

postid	userid
1	2
2	3
3	1

Table 8: Data in Like Table

Normalization Steps

1NF: Table contains only atomic values, so it is in 1NF.
2NF: No partial dependency exists as both 'postid' and 'userid' form the candidate key, and there are no non-prime attributes. Therefore, the table is in 2NF.
3NF: No transitive dependency exists as there are no non-prime attributes. Therefore, the table is in 3NF.

Normal Form	Normalized?
1NF	YES
2NF	YES
3NF	YES

Table 9: Normalization of Like table

4.5 Normalizing Followers Table

Relational Schema: Followers(follower_id, followed_id)

Functional Dependency:

- { follower_id, followed_id } → all attributes

Candidate Key: { follower_id, followed_id }
Prime Attributes: { follower_id, followed_id }
Non-Prime Attributes: None

Data in Followers Table

follower_id	followed_id
1	2
2	1
1	3

Table 10: Data in Followers Table

Normalization Steps

1NF: The table contains only atomic values, so it is in 1NF.
2NF: No partial dependency exists because both 'follower_id' and 'followed_id' form the candidate key, and there are no non-prime attributes. Therefore, the table is in 2NF.
3NF: No transitive dependency exists because there are no non-prime attributes. Therefore, the table is in 3NF.

Normal Form	Normalized?
1NF	YES
2NF	YES
3NF	YES

Table 11: Normalization of Followers table

4.6 Normalizing Reply Table

Relational Schema: Reply(replyid, desc, createdAt, userid, commentsid)

Functional Dependencies:

- { replyid } → { desc, createdAt, userid, commentsid }
- { userid, commentsid } → { replyid, desc, createdAt }

Candidate Key: { replyid }
Prime Attributes: { replyid }
Non-Prime Attributes: { desc, createdAt, userid, commentsid }

Data in Reply Table

replyid	desc	createdAt	userid	commentsid
1	Thanks for the feedback!	2024-11-14 12:00:00	1	1
2	I appreciate it!	2024-11-14 12:01:00	2	2
3	Interesting perspective!	2024-11-14 12:02:00	1	2

Table 12: Data in Reply Table

Normalization Steps

1NF: The table contains only atomic values (no repeating groups or arrays), so it is in 1NF.
2NF: There are no partial dependencies because ‘replyid‘ is the primary key, and the non-key attributes depend on it entirely. Therefore, the table is in 2NF.
3NF: There are no transitive dependencies because there are no non-prime attributes depending on other non-prime attributes. Therefore, the table is in 3NF.

Normal Form	Normalized?
1NF	YES
2NF	YES
3NF	YES

Table 13: Normalization of Reply table