SEVERUS SOCIAL NETWORK PLATFORM

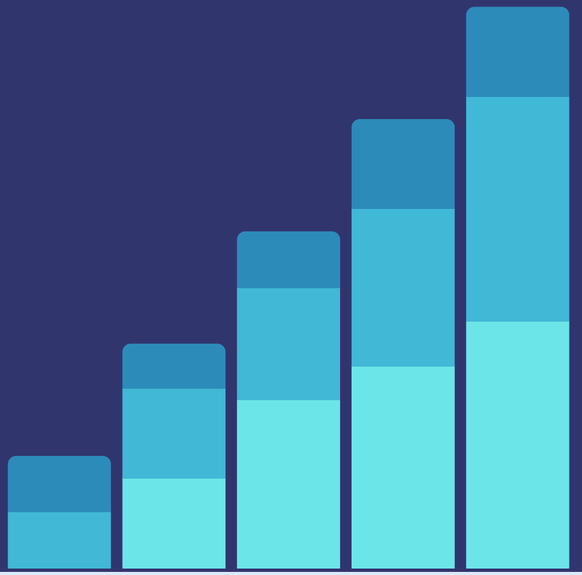
VU BINH MINH - V202100421
NGUYEN TUNG LAM - V202100571
BUI HUY LINH PHUC - V202100398

TABLE OF CONTENT

1. INTRODUCTION

Severus Social Network Platform: support core social networking features such as:

- user authentication
- media-rich content sharing
- real-time interaction.



Tools Overview: The project is built using a modern and robust technology stack

- **Next.js:** Frontend and server-side rendering.
- **Prisma ORM:** Facilitate secure and efficient database interactions with a MySQL relational database.
- **Clerk:** Enable user authentication and session management.
- **Cloudinary:** Manage and optimize user-uploaded images and videos, supporting rich media functionality that is essential for a social networking application.

2. SCHEMA IMPLEMENTATION

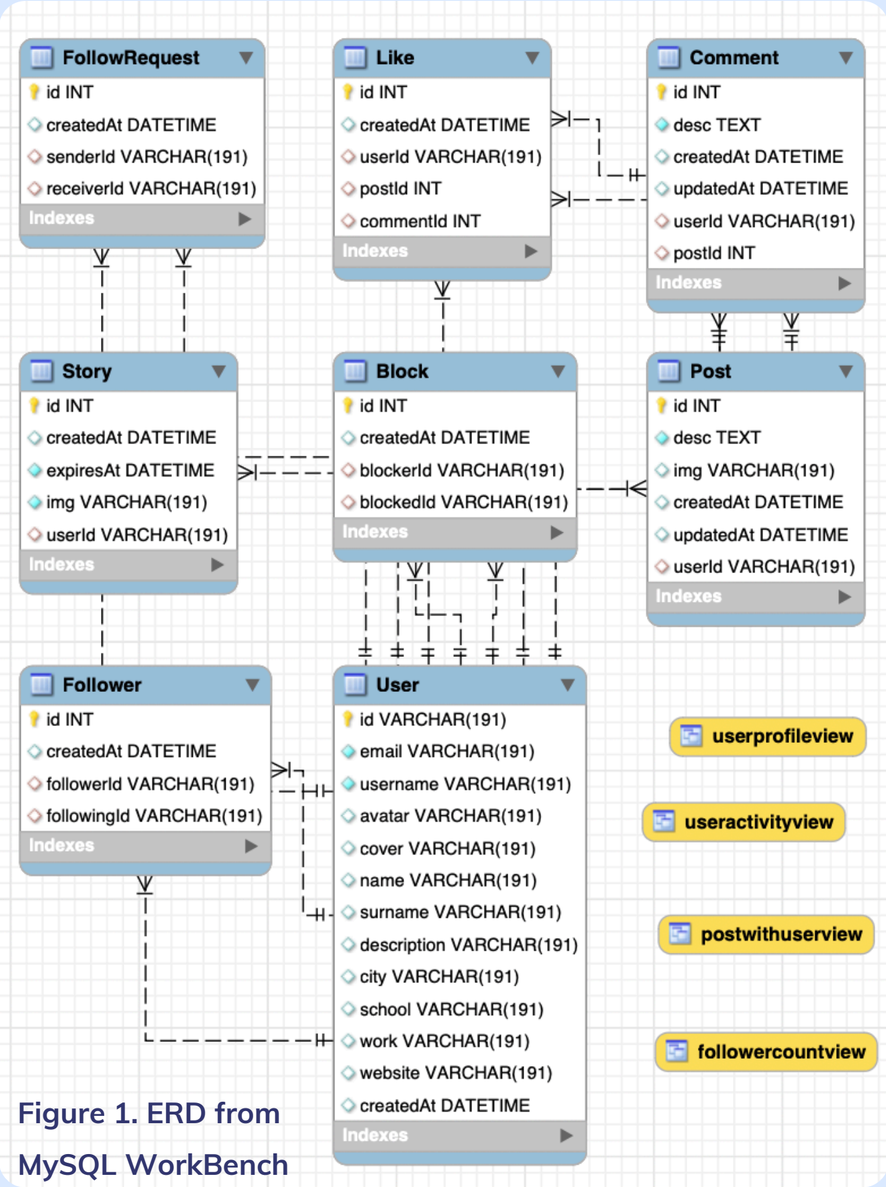


Figure 1. ERD from MySQL WorkBench

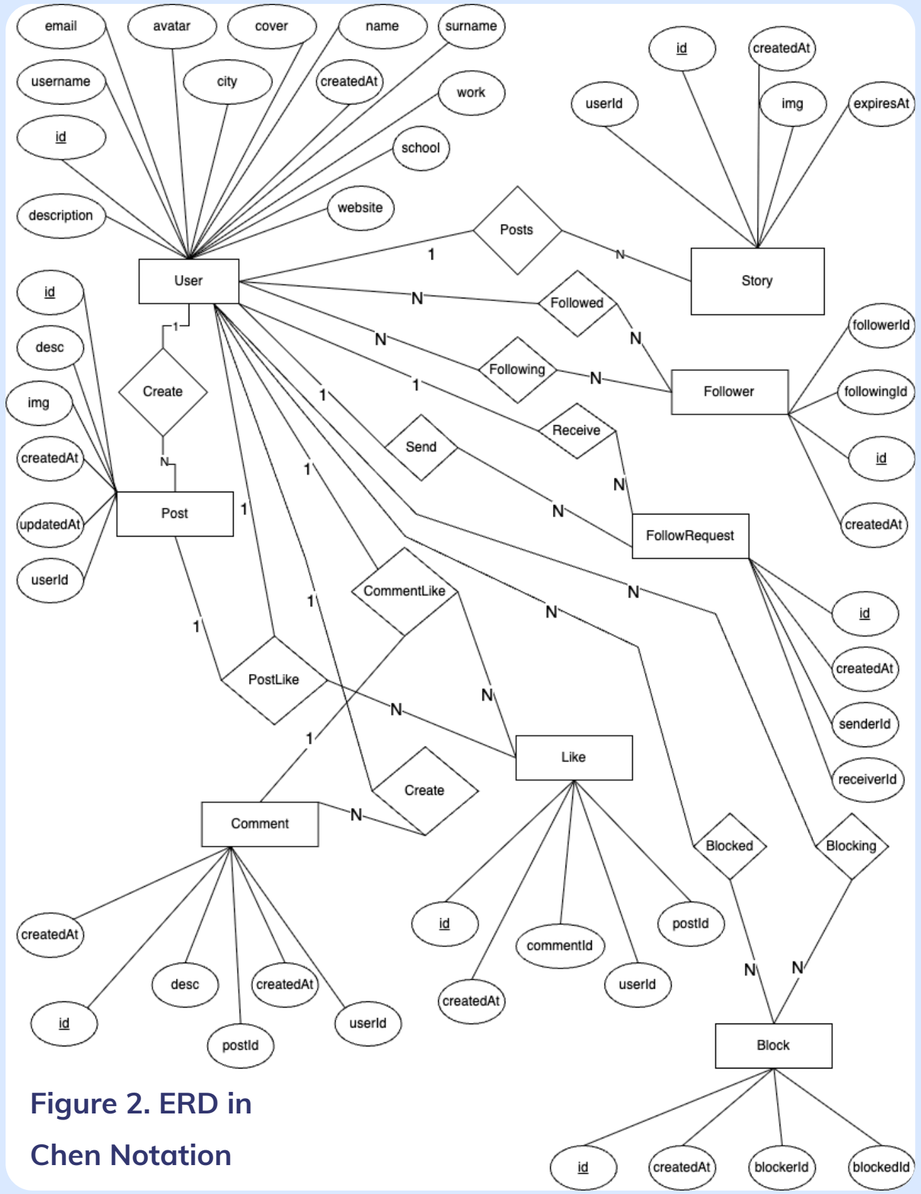


Figure 2. ERD in Chen Notation

3. SQL VIEW

```sql
-- UserProfileView
CREATE VIEW UserProfileView AS
SELECT id, username, name, surname, avatar, description, city, createdAt
FROM User;

-- PostWithUserView
CREATE VIEW PostWithUserView AS
SELECT p.id, p.desc, p.img, p.createdAt, p.updatedAt, u.username, u.avatar
FROM Post p
JOIN User u ON p.userId = u.id;

-- UserActivityView
CREATE VIEW UserActivityView AS
SELECT u.id, u.username,
       p.id AS postId, p.desc AS postDesc, p.createdAt AS postCreatedAt,
       c.id AS commentId, c.desc AS commentDesc, c.createdAt AS commentCreatedAt,
       l.postId AS likedPostId, l.createdAt AS likeCreatedAt
FROM User u
LEFT JOIN Post p ON u.id = p.userId
LEFT JOIN Comment c ON u.id = c.userId
LEFT JOIN `Like` l ON u.id = l.userId;

-- FollowerCountView
CREATE VIEW FollowerCountView AS
SELECT u.id, u.username,
       (SELECT COUNT(*) FROM Follower f WHERE f.followingId = u.id) AS followerCount,
       (SELECT COUNT(*) FROM Follower f WHERE f.followerId = u.id) AS followingCount
FROM User u;
```

These views streamline data retrieval by simplifying complex queries, improving performance for user profiles, feeds, activity tracking, and follower counts, while improving query maintainability.

3. SQL PROCEDURES & TRIGGER

```
# Procedure: Follow a user
DELIMITER $$

CREATE PROCEDURE FollowUser(
    IN p_followerId VARCHAR(191),
    IN p_followingId VARCHAR(191)
)
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM Block WHERE (blockerId = p_followingId AND blockedId = p_followerId)
    ) THEN
        INSERT IGNORE INTO Follower (followerId, followingId)
        VALUES (p_followerId, p_followingId);
    END IF;
END $$

DELIMITER ;
```

Procedure for following a user

```
# Procedure: Accept FollowRequest
DELIMITER $$

CREATE PROCEDURE AcceptFollowRequest(
  IN senderId VARCHAR(191),
  IN receiverId VARCHAR(191)
)
BEGIN
  DELETE FROM FollowRequest
  WHERE senderId = senderId AND receiverId = receiverId;

  INSERT IGNORE INTO Follower(followerId, followingId)
  VALUES (senderId, receiverId);
END $$

DELIMITER ;
```

Procedure for accepting follow requests

3. SQL PROCEDURES & TRIGGER

```sql
# Procedure: Block a user
DELIMITER $$

CREATE PROCEDURE BlockUser(
  IN blockerId VARCHAR(191),
  IN blockedId VARCHAR(191)
)
BEGIN
  DELETE FROM Follower
  WHERE (followerId = blockerId AND followingId = blockedId)
     OR (followerId = blockedId AND followingId = blockerId);

  INSERT IGNORE INTO Block(blockerId, blockedId)
  VALUES (blockerId, blockedId);
END $$

DELIMITER ;
```

Procedure for blocking a user

```sql
# Trigger: Prevent FollowRequest if Block
DELIMITER $$

CREATE TRIGGER PreventBlockedFollowRequest
BEFORE INSERT ON FollowRequest
FOR EACH ROW
BEGIN
  IF EXISTS (
    SELECT 1 FROM Block
    WHERE blockerId = NEW.receiverId AND blockedId = NEW.senderId
  ) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Cannot send follow request: blocked by this user.';
  END IF;
END $$

DELIMITER ;
```

The trigger blocking follow requests from blocked users, ensuring data consistency and reducing application-layer validation overhead.

3. PERFORMANCE OPTIMIZATIONS

```sql
CREATE INDEX idx_user_name ON User(name);
CREATE INDEX idx_post_userId ON Post(userId);
CREATE INDEX idx_post_createdAt ON Post(createdAt);
CREATE INDEX idx_comment_userId ON Comment(userId);
CREATE INDEX idx_comment_postId ON Comment(postId);
CREATE INDEX idx_comment_createdAt ON Comment(createdAt);
CREATE INDEX idx_like_userId_postId ON `Like`(userId, postId);
CREATE INDEX idx_like_userId_commentId ON `Like`(userId, commentId);
```

These indexes optimize query performance for user searches, feed retrieval, and interaction tracking, balancing read efficiency with write overhead.

3. SECURITY CONFIGURATIONS

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'secure_admin_pass';
CREATE USER 'normal_user'@'localhost' IDENTIFIED BY 'secure_user_pass';

GRANT ALL PRIVILEGES ON social.* TO 'admin'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON social.`User` TO 'normal_user'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON social.Post TO 'normal_user'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON social.Comment TO 'normal_user'@'localhost';
GRANT SELECT, INSERT, DELETE ON social.`Like` TO 'normal_user'@'localhost';
GRANT SELECT, INSERT, DELETE ON social.Follower TO 'normal_user'@'localhost';
GRANT SELECT, INSERT, DELETE ON social.FollowRequest TO 'normal_user'@'localhost';
GRANT SELECT, INSERT, DELETE ON social.Block TO 'normal_user'@'localhost';
GRANT SELECT, INSERT ON social.Story TO 'normal_user'@'localhost';
```

The admin@localhost user has full access to the social database for administrative tasks, restricted to local connections for safety. The normal_user@localhost is granted access that align with typical user actions while minimizing risks.

4. WEB INTERFACE

Let me show you a live demonstration of our web interface, or you can watch a recorded video at your convenience via the following link:

https://drive.google.com/drive/folders/1Sx2HUlMeDND6cXkO3WhyoX66Ci35jEDt?usp=sharing

5. TESTING RESULTS

Variable_name	Value
Handler_read_first	1
Handler_read_key	1
Handler_read_last	0
Handler_read_next	0
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	100002

Before indexing

Variable_name	Value
Handler_read_first	0
Handler_read_key	1
Handler_read_last	0
Handler_read_next	11
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0

After indexing

Without the index, the query needs to scan through the entire table row by row, resulting in a full table scan and significantly higher resource usage. After creating the index, we can efficiently locate matching records, drastically reducing the number of rows accessed and improving query performance.

5. SAMPLE QUERIES

```
DROP INDEX idx_user_name on User;

FLUSH STATUS;
SELECT * FROM User WHERE name LIKE 'Name1234%';
SHOW STATUS
LIKE 'Handler_read%';

EXPLAIN SELECT * FROM User WHERE name LIKE 'Name1234%';

CREATE INDEX idx_user_name ON User(name);

SHOW INDEX FROM User;

FLUSH STATUS;
SELECT * FROM User WHERE name LIKE 'Name1234%';
SHOW STATUS
LIKE 'Handler_read%';

EXPLAIN SELECT * FROM User WHERE name LIKE 'Name1234%';
```

THANK YOU

ANY QUESTIONS?