Department of Computer Science & Engineering

Database Management Systems - UE18CS252

Project Report

Submitted By

PES2201800266 Akash Murthy

Database for Discussion Forum

INDEX

Introduction

Since the introduction to modern technology, online forums have changed the way people interact and find solutions. The ease to find an answer or get a doubt solved has exponentially increased.

Building an online community where like-minded people can share their ideas, views, and thoughts, can increase the depth and intensity of a conversation.

This is the motivation behind building an online platform for students, teacher or anyone with curiosity to learn more about a topic and get doubts cleared.

The front-end for is built on PHP (w/ Bootstrap, JavaScript - jQuery, CSS, and HTML). The backend for this platform is built on phpMyAdmin using the InnoDB engine offered by MySQLi.

This Database should be able to store admin details, user details, their posts & replies in listed topics and threads. Admin can modify all structures, and users can only add threads, posts, and replies.

Admin and user details to be kept separate, to protect moderator’s privacy. Hierarchy of forum: [Topics→Thread→Post→Reply]

- Topic A
    - Thread A
        - Post 1
            - Reply 1
        - Post 2
        - Post 3
    - Thread B
- Topic B

Database should have provisions to store user details like unique ID, name, username, profile picture path, email, password, address, and registration time.

Each thread, post or reply created by the user must have a unique ID associated with it, along with details as to title, description, views, posted by, and time of creation.

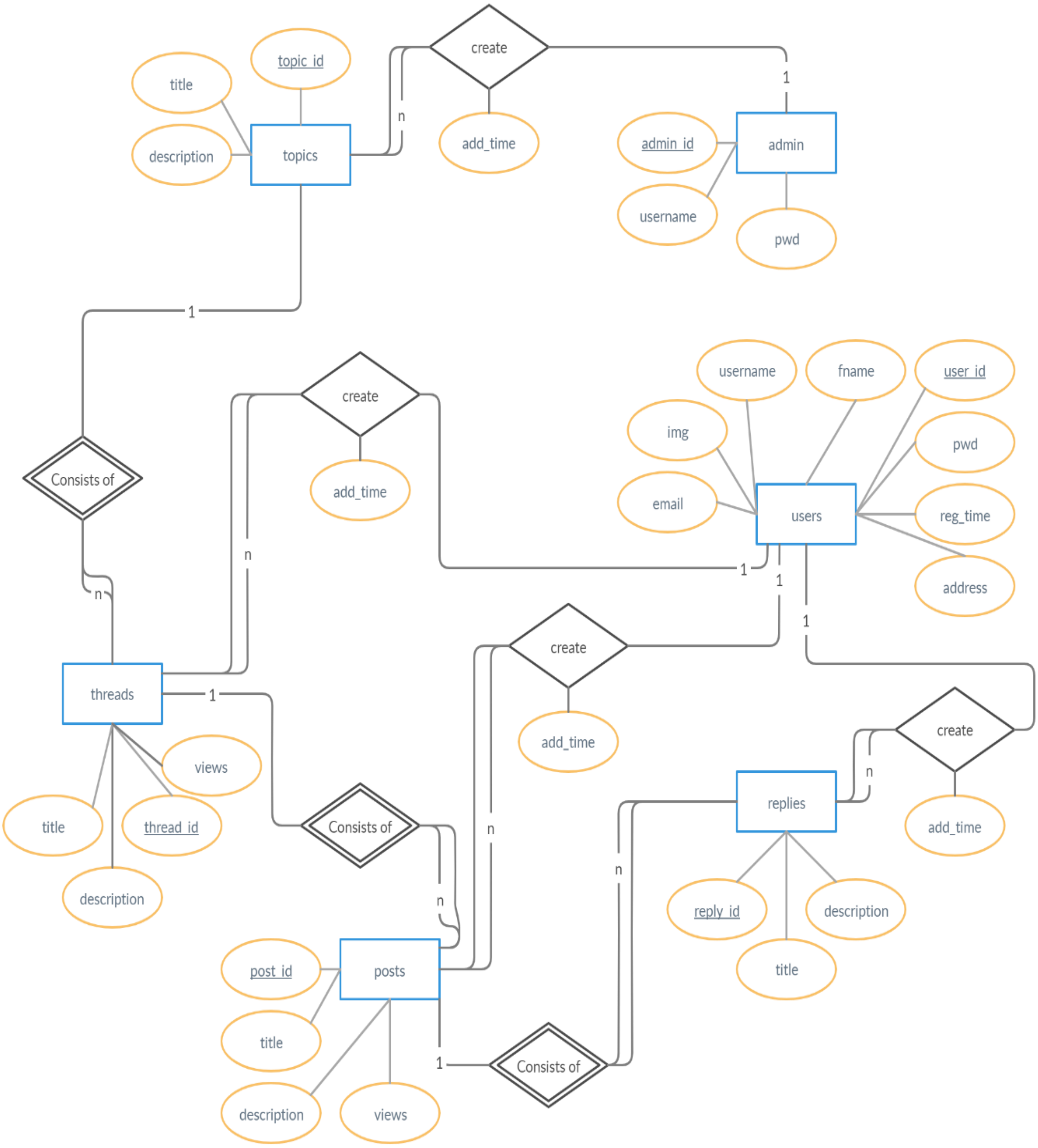
Topics can only be added and modified by admins. Details of admins not required to be stored. Only admin login details to be stored.

Constraints:

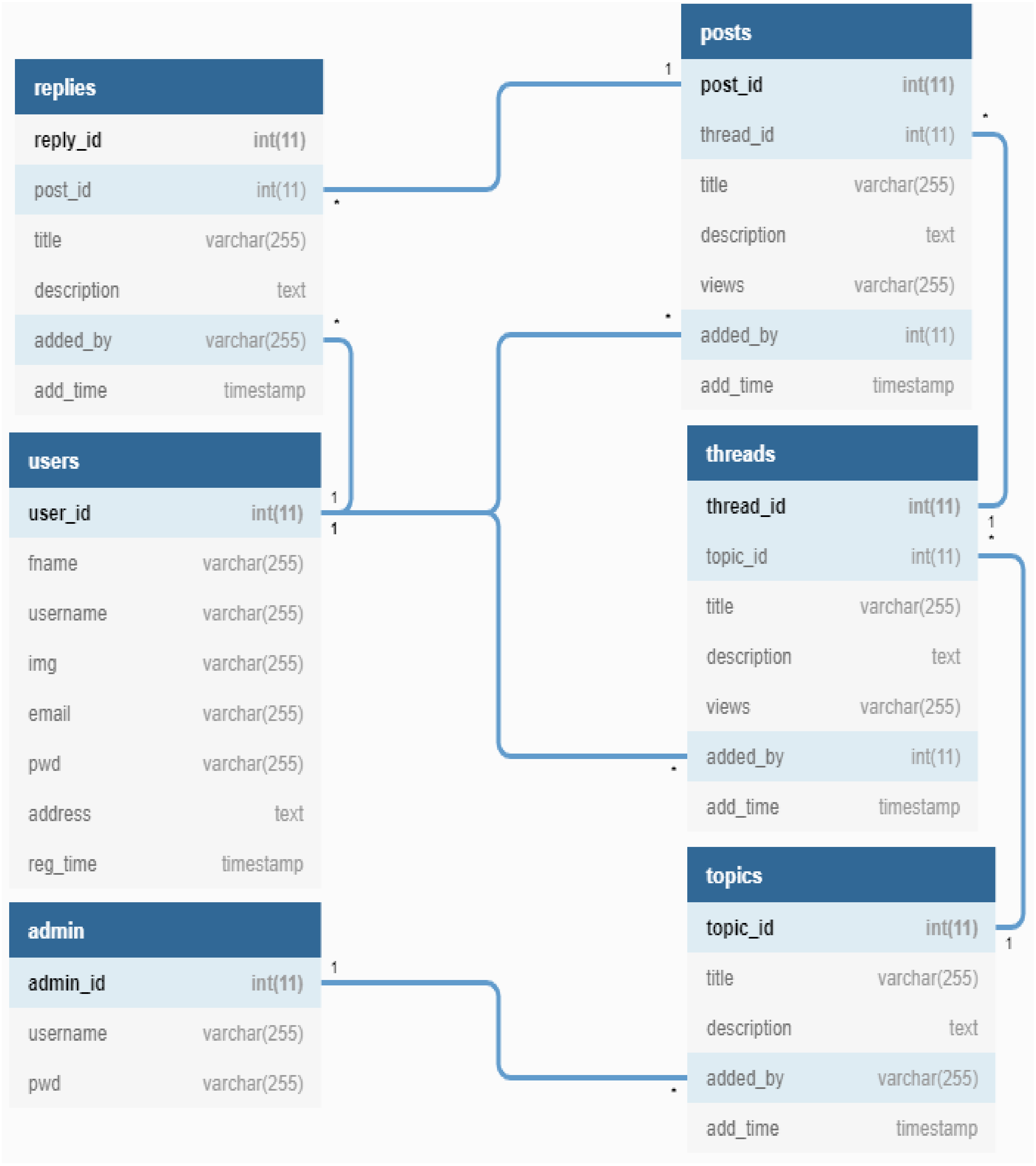
- The attribute ID for each entity must be unique and have auto increment
- Usernames and Passwords cannot be NULL
- The attribute view must have default value as 0
- Timestamp attribute for any entity must have default as current timestamp, on update should reflect current timestamp
- All attributes to be filled at the time of creation, cannot possess NULL

Data Model

- ER Diagram in Chen's Notation



- Relational Schema



replies

reply_id	int(11)
post_id	int(11)
title	varchar(255)
description	text
added_by	varchar(255)
add_time	timestamp

users

user_id	int(11)
fname	varchar(255)
username	varchar(255)
img	varchar(255)
email	varchar(255)
pwd	varchar(255)
address	text
reg_time	timestamp

admin

admin_id	int(11)
username	varchar(255)
pwd	varchar(255)

posts

post_id	int(11)
thread_id	int(11)
title	varchar(255)
description	text
views	varchar(255)
added_by	int(11)
add_time	timestamp

threads

thread_id	int(11)
topic_id	int(11)
title	varchar(255)
description	text
views	varchar(255)
added_by	int(11)
add_time	timestamp

topics

topic_id	int(11)
title	varchar(255)
description	text
added_by	varchar(255)
add_time	timestamp

Functional Dependencies of Established Relations

i.e. A → B, where A determines B. Which means A can uniquely identify B.

```
1. admin_id     → {username, pwd}
2. topic_id     → {title, description, added_by, add_time}
3. thread_id    → {topic_id, title, description, views, added_by, add_time}
4. post_id      → {thread_id, title, description, views, added_by, add_time}
5. reply_id     → {title, description, views, added_by, add_time}
6. user_id      → {fname, username, img, email, pwd, address, reg_time}
7. {added_by, add_time}     → post_id               [Redundant]
8. {added_by, add_time}     → topic_id              [Redundant]
9. {added_by, add_time}     → reply_id              [Redundant]
10.{added_by, add_time}     → thread_id             [Redundant]
```

[Multivalued Dependencies – Can be ignored due Transitive Nature]

[Trivial FD - Subset & Transitivity Law: reply_id → post_id → thread_id → topic_id]

Normalization

As this database was developed using ER Model and schema approach, the set of relations formed are normalized.

Here are some cases were the second normal form could be violated:

- Adding username to create relationship table
- Joining tables under the Consists of relationship table
  [i.e. Having replies and posts table together]
- Including users table with any other table
- Including topics with admin table
- Introducing an admin column in users table

The third normal form would be violated if any additional attributes can reference the prime attributes. As this is a case of transitive dependency.

Another Example:

The posts table would have violated the normal forms if it had a replies column, as replies section is placed in its own table to include all its attributes. It would have been difficult to handle if it were in the posts table.

From a practical standpoint having replies embedded in the post table would result in multiple replies, making the table less space efficient while accessing from the server side, as this is run on PHP with scripts and not manually.

1NF	
2NF	
3NF	
BCNF	admin, users, topics, threads, posts, replies

DDL

```
-- Database: `forum`

-- Table structure for table `admin`
CREATE TABLE `admin` (
  `admin_id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) NOT NULL,
  `pwd` varchar(255) NOT NULL,
  PRIMARY KEY (`admin_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;


-- Table structure for table `posts`
CREATE TABLE `posts` (
  `post_id` int(11) NOT NULL AUTO_INCREMENT,
  `thread_id` int(11) NOT NULL,
  `title` varchar(255) NOT NULL,
  `description` text NOT NULL,
  `views` varchar(255) NOT NULL DEFAULT '0',
  `added_by` int(11) NOT NULL FOREIGN KEY REFERENCES users(user_id),
  `add_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`post_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;


-- Table structure for table `replies`
CREATE TABLE `replies` (
  `reply_id` int(11) NOT NULL AUTO_INCREMENT,
  `post_id` int(11) NOT NULL,
  `title` varchar(255) NOT NULL,
  `description` text NOT NULL,
  `added_by` varchar(255) NOT NULL FOREIGN KEY REFERENCES users(user_id),
  `add_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`reply_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;


-- Table structure for table `threads`
CREATE TABLE `threads` (
  `thread_id` int(11) NOT NULL AUTO_INCREMENT,
  `topic_id` int(11) NOT NULL,
  `title` varchar(255) NOT NULL,
  `description` text NOT NULL,
  `views` varchar(255) NOT NULL DEFAULT '0',
  `added_by` int(11) NOT NULL FOREIGN KEY REFERENCES users(user_id),
  `add_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`thread_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=5 ;
```

```
-- Table structure for table `topics`
CREATE TABLE `topics` (
  `topic_id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `description` text NOT NULL,
  `added_by` varchar(255) NOT NULL DEFAULT '0' FOREIGN KEY REFERENCES
admin(admin_id),
  `add_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`topic_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=8 ;


-- Table structure for table `users`
CREATE TABLE `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `fname` varchar(255) NOT NULL,
  `username` varchar(255) NOT NULL,
  `img` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `pwd` varchar(255) NOT NULL,
  `address` text NOT NULL,
  `reg_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;


ALTER TABLE `admin`
ADD CONSTRAINT PASSWORD_CHK CHECK (LEN(pwd) >= 4);

ALTER TABLE `users`
ADD CONSTRAINT PASSWORD_CHK CHECK (LEN(pwd) >= 4);
```

Lossless Join – A Brief

Consider a case where some users were given an option to moderate the forum under the same username with admin privileges. This would mean admin and user table could be combined. But this would lead to a lot of redundancies and repetition of data, which results in data redundancy.

Therefore, one would advise to split or decompose the admin - user table.
If we were to join the said admin and user tables back together, we would get the original table. This is a lossless join, were none of the original data was lost.

Triggers [Default Delimiter ;]

- **Here are some triggers to update the total number of users and admins**

```
CREATE TABLE `total` (`user` int(11), `admin` int(11))
INSERT INTO `total` VALUES((SELECT COUNT(*) FROM users),(SELECT COUNT(*)
FROM admin))

CREATE TRIGGER total_user_delete
AFTER DELETE
      ON users FOR EACH ROW
UPDATE total
set user=(SELECT COUNT(*) FROM users);

CREATE TRIGGER total_user_insert
AFTER INSERT
      ON users FOR EACH ROW
UPDATE total
set user=(SELECT COUNT(*) FROM users);

CREATE TRIGGER total_admin_delete
AFTER DELETE
      ON admin FOR EACH ROW
UPDATE total
set admin=(SELECT COUNT(*) FROM admin);

CREATE TRIGGER total_admin_insert
AFTER INSERT
      ON admin FOR EACH ROW
UPDATE total
set admin=(SELECT COUNT(*) FROM admin);
```

- **Let's make an audit trail of deleted users**

```
CREATE TABLE `users_deleted` (
  `user_id` int(11) NOT NULL,
  `fname` varchar(255) NOT NULL,
  `username` varchar(255) NOT NULL,
  `img` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `pwd` varchar(255) NOT NULL,
  `address` text NOT NULL,
  `reg_time` timestamp NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TRIGGER deleted_users
AFTER DELETE
ON users FOR EACH ROW
AS
BEGIN
DECLARE id INT;
DECLARE reg TIMESTAMP;
DECLARE fn, us, im, em, pd, ad VARCHAR(255);
SELECT OLD.user_id INTO id;
SELECT OLD.fname INTO fn;
SELECT OLD.username INTO us;
SELECT OLD.img INTO im;
SELECT OLD.email INTO em;
SELECT OLD.pwd INTO pd;
SELECT OLD.address INTO ad;
INSERT INTO users_deleted
VALUES (id, fn, us, im, em, pd, ad, reg);
END;
```

SQL Queries

Write SQL retrieval queries to:

Display details of user who have created the highest number of replies

```
SELECT * FROM (SELECT added_by AS user_id, COUNT(*) as nReplies
                   FROM replies
                   GROUP BY added_by
                   ORDER BY nReplies desc limit 0,1) AS result
NATURAL JOIN users
```

Display details of user are inactive (No posts, replies or threads since 1st Jan 2020 00:00:00)

```
SELECT * FROM users
WHERE users.user_id NOT IN
(
SELECT added_by AS user_id FROM replies
WHERE add_time BETWEEN '2020-01-01 00:00:00' and TIMESTAMP(CURRENT_DATE())
UNION
SELECT added_by AS user_id FROM posts
WHERE add_time BETWEEN '2020-01-01 00:00:00' and TIMESTAMP(CURRENT_DATE())
UNION
SELECT added_by AS user_id FROM threads
WHERE add_time BETWEEN '2020-01-01 00:00:00' and TIMESTAMP(CURRENT_DATE())
)
```

Display the number of replies created by each user and their names

```
SELECT * FROM
(SELECT added_by AS user_id, COUNT(*) AS nReplies FROM replies GROUP BY user_id)
AS result
LEFT JOIN (SELECT user_id, fname, username FROM users) as u
ON result.user_id=u.user_id
```

Display the all user details along with the threads they created

```
SELECT * FROM threads AS t
LEFT JOIN users AS u ON t.added_by = u.user_id
UNION
SELECT * FROM threads AS t
RIGHT JOIN users AS u ON t.added_by = u.user_id
```

Display only the username, first name and user ID along with the posts they created

```
SELECT * FROM posts AS t
LEFT JOIN (select user_id, fname, username from users) AS u
      ON t.added_by=u.user_id
UNION
SELECT * FROM posts as t
RIGHT JOIN (select user_id, fname, username from users) AS u
      ON t.added_by=u.user_id
```

Calculate the average number of replies created by each user

```
SELECT * FROM
(SELECT count/(SELECT COUNT(*) as count FROM `posts`) as avg_posts, user_id
 FROM
   (SELECT COUNT(*) as count, added_by AS user_id FROM `posts` GROUP BY added_by)
   AS counts
)
AS t
LEFT JOIN
(SELECT username, fname, user_id FROM users) AS u
ON t.user_id = u.user_id
```

Conclusion

This database stores data about users and the posts they create. It is organised, making maintenance of records easy. The database is used for an online forum for discussions. Different auto increment values have been implemented to randomize ID values.

The complexity of this database is low and anyone with basic understanding of SQL will find writing queries easy. Currently, this database supports multiple user access. Manipulation of data can be done with multiple logins.

However, this system can have further improvements, such as storing additional information like date of birth, age, etc.
It can also include a rating system or experience rewards to further enhance user interaction at front-end level. An email trigger to welcome new users to the forum can be added.
Query optimisations is another aspect to be considered.