



BILKENT UNIVERSITY

Computer Engineering Department

CS 353 DATABASE SYSTEMS FALL 2017

CASESWITCHERS

Collaborative Hypertext Dictionary

PROJECT DESIGN REPORT

Section 1 - Group 2

Eren Bilaloğlu

Afra Dömeke

Gülce Karaçal

Fırat Sivrikaya

Available online : <https://djcedrics.github.io/CaseSwitchers>

CONTENTS

REVISED E/R MODEL	3
RELATION SCHEMAS	5
User	5
Category	6
Entry	7
Post	8
Admin	9
Comment	10
BannedUsers	11
Subcategory	12
SubComments	13
Messages	14
Rates	15
Favorites	16
Owns	17
PostCategory	18
PostComments	19
Admin	20
FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES	21
FUNCTIONAL COMPONENTS	21
Use Cases	21
User	21
Admin	23
Algorithms	26
Logical Requirements	26
Entry Related Algorithms	26
Data Structures	26
USER INTERFACE DESIGN AND SQL STATEMENTS	27
Login Screen	27
Sign Up Screen	28
Category Specific Screen / Creating a New Post	29
Update Profile Information Screen	31
Post Specific Screen	32
Creating a Comment	32
Creating a Subcomment	33
Rate an Entry	33
Add an entry to favorites	34

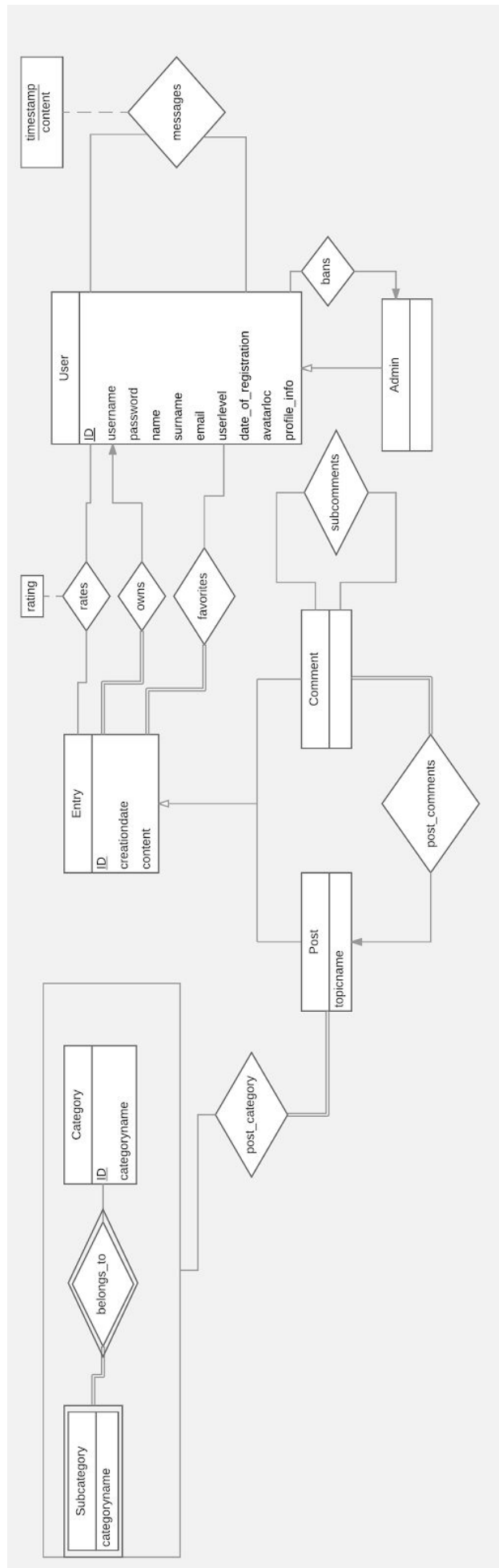
Message Screen	34
Sending a Message	35
Searching for a Message	35
User Profile Screen	36
Admin Panel	39
Edit Post	40
Delete Post	40
Edit Comment	41
Delete Comment	41
Ban User	41
Unban User	42
Create Category	42
Delete Category	42
Create Subcategory	42
Change Subcategory Parent	43
Update User Bio	43
ADVANCED DATABASE COMPONENTS	44
Views	44
Users Messages View	44
Stored Procedures	44
Get the Rating of a User by ID	44
Get the most recent 3 comments of a user	45
Get the most recent 3 posts of a user	45
Reports	46
Total Number of Posts	46
Total Number of Comments	46
Total Number of Categories	46
Total Number of Subcategories	46
Total Number of Users	46
Total Number of Posts under each Category	46
Total Ratings of Each User in Descending Order	46
Triggers	47
Constraints	47
IMPLEMENTATION PLAN	48

1. REVISED E/R MODEL

After getting feedback from our assistant, we changed our E/R model in order to have a better structure of the database system. The modifications made in the E/R model are as follows:

- We added a new relation called “subcomments” between two Comment entities so that in our system comments can have comments. The mapping cardinality constraint of this relation is defined as M-M because multiple comments can have multiple comments.
- We removed rating attribute of Entry entity. Instead, we defined “rates” relation between User entity and Entry entity. The mapping cardinality constraint of this relation is defined as M-M because multiple entries can be rated by multiple users and vice versa.
- We added a weak entity set called “Subcategory” which is related to Category by the “belongs_to” relation. Every subcategory is required to have a category which indicates total participation of subcategories into the relation. The mapping cardinality constraint of this relation is defined as 1-M since a category can have multiple subcategories but a subcategory only belongs to one category.
- We added aggregation in our model. Doing that, our model represents:
 - A subcategory belongs to a category.
 - A category, subcategory combination may have an associated post.

Our updated E/R diagram can be found below:



2. RELATION SCHEMAS

2.1. User

Relational Model:

User (ID, username, password, name, surname, email, userlevel, date_of_registration, avatarloc, profile_info)

Functional Dependencies:

ID → username, password, name, surname, email, userlevel, date_of_registration, avatarloc, profile_info

username → ID, password, name, surname, email, userlevel, date_of_registration, avatarloc, profile_info

email → ID, password, name, surname, userlevel, date_of_registration, avatarloc, profile_info

Candidate Keys:

{(ID), (username), (email)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`User` (  
  `ID` INT NOT NULL AUTO_INCREMENT,  
  `username` VARCHAR(45) NOT NULL,  
  `password` VARCHAR(45) NOT NULL,  
  `name` VARCHAR(45) ,  
  `surname` VARCHAR(45),  
  `email` VARCHAR(45) NOT NULL,  
  `userlevel` VARCHAR(45) NOT NULL,  
  `date_of_registration` TIMESTAMP NOT NULL,  
  `avatarloc` VARCHAR(45),  
  `profile_info` TINYTEXT,  
  PRIMARY KEY (`ID`),  
  UNIQUE INDEX `username_UNIQUE` (`username` ASC),  
  UNIQUE INDEX `email_UNIQUE` (`email` ASC),  
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC))
```

ENGINE = InnoDB

2.2. Category

Relational Model:

Category (ID, categoryname)

Functional Dependencies:

ID → categoryname

Candidate Keys:

{{ID}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Category` (  
  `ID` INT NOT NULL AUTO_INCREMENT,  
  `categoryname` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC)  
  UNIQUE INDEX `categoryname_UNIQUE` (`categoryname` ASC))  
ENGINE = InnoDB;
```

2.3. Entry

Relational Model:

Entry (ID, creationdate, content)

Functional Dependencies:

ID \rightarrow creationdate, content

Candidate Keys:

{{ID}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Entry` (  
  `ID` INT NOT NULL AUTO_INCREMENT,  
  `creationdate` TIMESTAMP NOT NULL,  
  `content` MEDIUMTEXT NOT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC))  
ENGINE = InnoDB;
```


2.4. Post

Relational Model:

Post (ID, topicname) FK : ID ref. Entry (ID)

Functional Dependencies:

ID → topicname

Candidate Keys:

{{ID}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Post` (  
  `ID` INT NOT NULL,  
  `topicname` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC),  
  CONSTRAINT `ID`  
    FOREIGN KEY (`ID`)  
      REFERENCES `mydb`.`Entry` (`ID`)  
  CONSTRAINT `creationdate`  
    FOREIGN KEY ()  
      REFERENCES `mydb`.`Entry` ())  
ENGINE = InnoDB
```

2.5. Admin

Relational Model:

Admin (ID) FK: ID ref. User (ID)

Functional Dependencies:

None

Candidate Keys:

{{ID}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Admin` (  
  `ID` INT NOT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC),  
  CONSTRAINT `ID`  
    FOREIGN KEY (`ID`)  
      REFERENCES `mydb`.`User` (`ID`))  
ENGINE = InnoDB
```

2.6. Comment

Relational Model:

Comment (ID)

FK: ID ref. Entry (ID)

Functional Dependencies:

None

Candidate Keys:

{{ID}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Comment` (  
  `ID` INT NOT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC),  
  CONSTRAINT `ID`  
    FOREIGN KEY (`ID`)  
      REFERENCES `mydb`.`Entry` (`ID`))  
ENGINE = InnoDB
```

2.7. BannedUsers

Relational Model:

BannedUsers (banned_id, admin_id)

FK: admin_id ref. Admin(ID)

FK: banned_id ref. User(ID)

Functional Dependencies:

banned_id → admin_id

Candidate Keys:

{{banned_id}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`BannedUsers` (  
  `banned_id` INT NOT NULL,  
  `admin_id` INT NOT NULL,  
  PRIMARY KEY (`banned_id`, `admin_id`),  
  INDEX `admin_id_idx` (`admin_id` ASC),  
  UNIQUE INDEX `admin_id_UNIQUE` (`admin_id` ASC),  
  CONSTRAINT `banned_id`  
    FOREIGN KEY (`banned_id`)  
      REFERENCES `mydb`.`User` (`ID`)  
  CONSTRAINT `admin_id`  
    FOREIGN KEY (`admin_id`)  
      REFERENCES `mydb`.`User` (`ID`))  
ENGINE = InnoDB
```

2.8. Subcategory

Relational Model:

Subcategory (sub_id, c_id, subcategoryname) FK: c_id ref. Category (ID)

Functional Dependencies:

sub_id, c_id → subcategoryname

Candidate Keys:

{(sub_id, c_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Subcategory` (  
  `sub_id` INT NOT NULL AUTO_INCREMENT,  
  `c_id` INT NOT NULL,  
  `subcategoryname` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`sub_id`, `c_id`),  
  UNIQUE INDEX `sub_id_UNIQUE` (`sub_id` ASC),  
  INDEX `c_id_idx` (`c_id` ASC),  
  UNIQUE INDEX `subcategoryname_UNIQUE` (`subcategoryname` ASC),  
  CONSTRAINT `c_id`  
    FOREIGN KEY (`c_id`)  
      REFERENCES `mydb`.`Category` (`ID`))  
ENGINE = InnoDB
```

2.9. SubComments

Relational Model:

SubComments (comment_id, subcomment_id) FK: comment_id ref. Comment (ID)
FK: subcomment_id ref. Comment (ID)

Functional Dependencies:

None

Candidate Keys:

{{comment_id, subcomment_id}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`SubComments` (  
  `comment_id` INT NOT NULL,  
  `subcomment_id` INT NOT NULL,  
  PRIMARY KEY (`comment_id`, `subcomment_id`),  
  INDEX `subcomment_id_idx` (`subcomment_id` ASC),  
  CONSTRAINT `comment_id`  
    FOREIGN KEY (`comment_id`)  
      REFERENCES `mydb`.`Comment` (`ID`),  
  CONSTRAINT `subcomment_id`  
    FOREIGN KEY (`subcomment_id`)  
      REFERENCES `mydb`.`Comment` (`ID`))  
ENGINE = InnoDB
```

2.10. Messages

Relational Model:

Messages (sender_id, receiver_id, timestamp, content) FK: sender_id ref. User (ID)
FK: receiver_id ref. User(ID)

Functional Dependencies:

sender_id, receiver_id, timestamp → content

Candidate Keys:

{(sender_id, receiver_id, timestamp)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Messages` (  
  `sender_id` INT NOT NULL,  
  `receiver_id` INT NOT NULL,  
  `timestamp` TIMESTAMP NOT NULL,  
  `content` MEDIUMTEXT NOT NULL,  
  PRIMARY KEY (`sender_id`, `receiver_id`, `timestamp`),  
  INDEX `receiver_id_idx` (`receiver_id` ASC),  
  CONSTRAINT `sender_id`  
    FOREIGN KEY (`sender_id`)  
    REFERENCES `mydb`.`User` (`ID`),  
  CONSTRAINT `receiver_id`  
    FOREIGN KEY (`receiver_id`)  
    REFERENCES `mydb`.`User` (`ID`))  
ENGINE = InnoDB
```

2.11. Rates

Relational Model:

Rates(e_id,u_id,rating) FK: e_id ref. Entry (ID)
FK: u_id ref. User(ID)

Functional Dependencies:

$$e_id, u_id \rightarrow rating$$

Candidate Keys:

$$\{(e_id, u_id)\}$$

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Rates` (  
  `e_id` INT NOT NULL,  
  `u_id` INT NOT NULL,  
  `rating` INT NOT NULL,  
  PRIMARY KEY (`e_id`, `u_id`),  
  INDEX `u_id_idx` (`u_id` ASC),  
  CONSTRAINT `e_id`  
    FOREIGN KEY (`e_id`)  
      REFERENCES `mydb`.`Entry` (`ID`),  
  CONSTRAINT `u_id`  
    FOREIGN KEY (`u_id`)  
      REFERENCES `mydb`.`User` (`ID`))  
ENGINE = InnoDB
```


2.12. Favorites

Relational Model:

Favorites(e_id, u_id) FK: e_id ref. Entry (ID)
FK: u_id ref. User(ID)

Functional Dependencies:

None

Candidate Keys:

$$\{(e_id, u_id)\}$$

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Rates` (  
  `e_id` INT NOT NULL,  
  `u_id` INT NOT NULL,  
  PRIMARY KEY (`e_id`, `u_id`),  
  INDEX `u_id_idx` (`u_id` ASC),  
  CONSTRAINT `e_id`  
    FOREIGN KEY (`e_id`)  
      REFERENCES `mydb`.`Entry` (`ID`),  
  CONSTRAINT `u_id`  
    FOREIGN KEY (`u_id`)  
      REFERENCES `mydb`.`User` (`ID`))  
ENGINE = InnoDB
```

2.13. Owns

Relational Model:

Owns(e_id, u_id) FK: e_id ref. Entry (ID)
 FK: u_id ref. User(ID)

Functional Dependencies:

None

Candidate Keys:

{(e_id, u_id)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Owns` (  
  `e_id` INT NOT NULL,  
  `u_id` INT NOT NULL,  
  PRIMARY KEY (`e_id`, `u_id`),  
  INDEX `u_id_idx` (`u_id` ASC),  
  UNIQUE INDEX `u_id_UNIQUE` (`u_id` ASC),  
  CONSTRAINT `e_id`  
    FOREIGN KEY (`e_id`)  
      REFERENCES `mydb`.`Entry` (`ID`),  
  CONSTRAINT `u_id`  
    FOREIGN KEY (`u_id`)  
      REFERENCES `mydb`.`User` (`ID`))  
ENGINE = InnoDB
```

2.14. PostCategory

Relational Model:

PostCategory(p_id, c_id, s_id)

FK: p_id ref. Post (ID)

FK: c_id ref. Category(ID)

FK: s_id ref. Subcategory(ID)

Functional Dependencies:

$p_id \rightarrow c_id, s_id$

Candidate Keys:

{{p_id}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`PostCategory` (  
  `p_id` INT NOT NULL,  
  `c_id` INT NOT NULL,  
  `s_id` INT NOT NULL,  
  PRIMARY KEY (`p_id`),  
  INDEX `p_id_idx` (`p_id` ASC),  
  UNIQUE INDEX `p_id_UNIQUE` (`p_id` ASC),  
  CONSTRAINT `p_id`  
    FOREIGN KEY (`p_id`)  
      REFERENCES `mydb`.`Post` (`ID`),  
  CONSTRAINT `c_id`  
    FOREIGN KEY (`c_id`)  
      REFERENCES `mydb`.`Category` (`ID`))  
  CONSTRAINT `s_id`  
    FOREIGN KEY (`s_id`)  
      REFERENCES `mydb`.`Subcategory` (`ID`))  
ENGINE = InnoDB
```

2.15. PostComments

Relational Model:

PostComments(p_id, c_id)

FK: p_id ref. Post (ID)

FK: c_id ref. Comment(ID)

Functional Dependencies:

None

Candidate Keys:

{{p_id, c_id}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`PostComments` (  
  `p_id` INT NOT NULL,  
  `c_id` INT NOT NULL,  
  PRIMARY KEY (`p_id`, `c_id`),  
  INDEX `c_id_idx` (`c_id` ASC),  
  UNIQUE INDEX `c_id_UNIQUE` (`c_id` ASC),  
  CONSTRAINT `p_id`  
    FOREIGN KEY (`p_id`)  
      REFERENCES `mydb`.`Post` (`ID`),  
  CONSTRAINT `c_id`  
    FOREIGN KEY (`c_id`)  
      REFERENCES `mydb`.`Comment` (`ID`))  
ENGINE = InnoDB
```

2.16. Admin

Relational Model:

Admin(ID) FK: ID ref. User(ID)

Functional Dependencies:

None

Candidate Keys:

{{ID}}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE IF NOT EXISTS `mydb`.`Admin` (  
  `ID` INT NOT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE INDEX `ID_UNIQUE` (`ID` ASC),  
  CONSTRAINT `ID`  
    FOREIGN KEY (`ID`)  
      REFERENCES `mydb`.`User` (`ID`))  
ENGINE = InnoDB
```

3. FUNCTIONAL DEPENDENCIES AND NORMALIZATION OF TABLES

The Relation Schemas part of our design report contains all the functional dependencies and normal forms. Since the relations are all in Boyce-Codd Normal Form (BCNF), there is no need for any decomposition nor normalization.

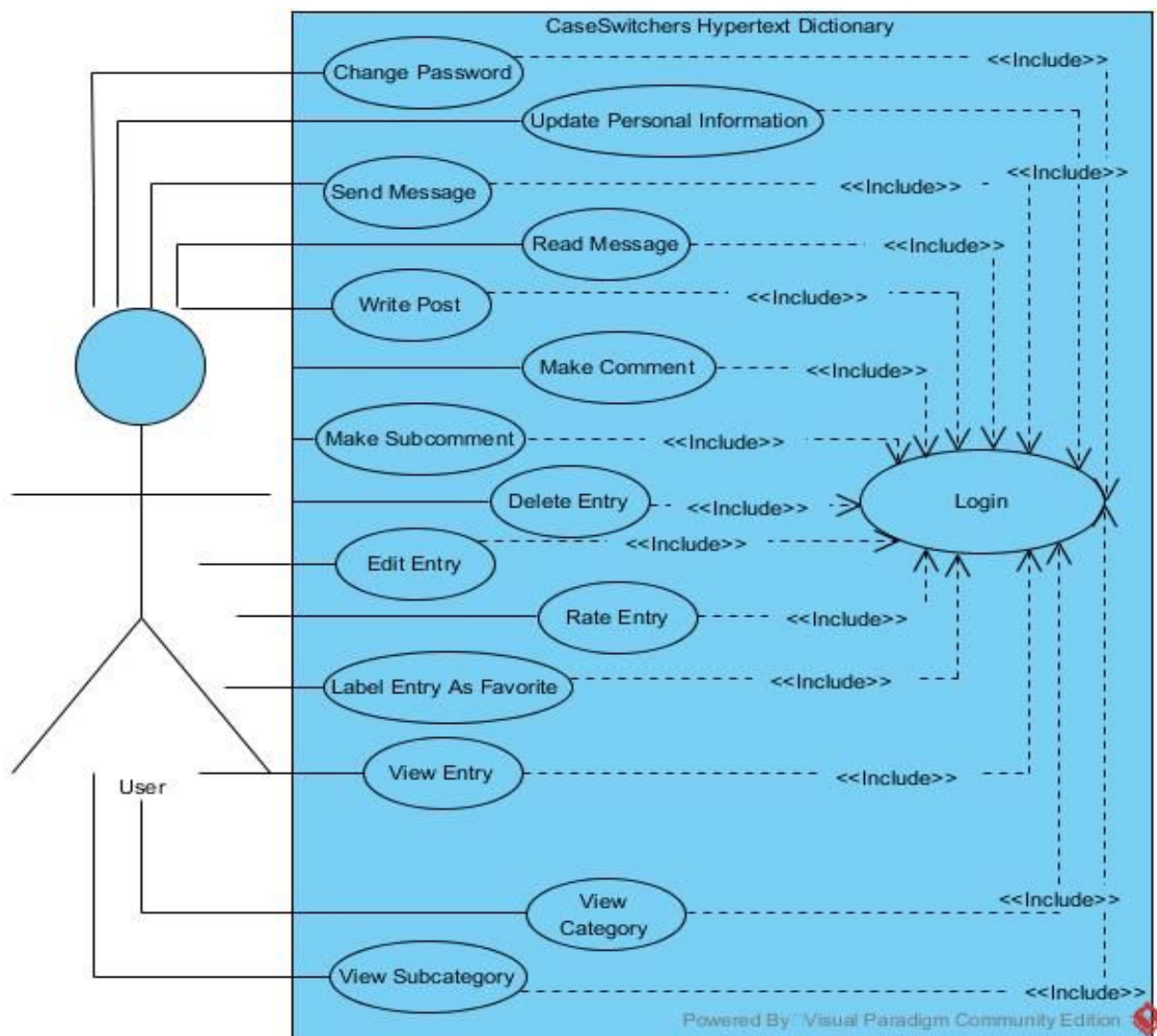
4. FUNCTIONAL COMPONENTS

4.1. Use Cases

4.1.1. User

- **Change Password:** If users want, they can change their passwords due to security issues.
- **Update Personal Information:** Users can update their account information such as username and e-mail. When changing the username and e-mail, it should be checked whether the new entered username and e-mail are unique or not.
- **Send Message:** Users can send direct messages to each other.
- **Read Message:** Users can read direct messages sent from other users.
- **Write Post:** Users can write posts about which category they desire.
- **Make Comment:** Users can make comment about other users' posts if they want.
- **Make Subcomment:** Users can make subcomments about other users' comments if they want.
- **Delete Entry:** Users can delete their past entries about any category.
- **Edit Entry:** Users can edit entries that they posted before.
- **Rate Entry:** Users can rate entries of others. If they like an entry, they can rate it as "up" or if they don't like it, they can rate it as "down".

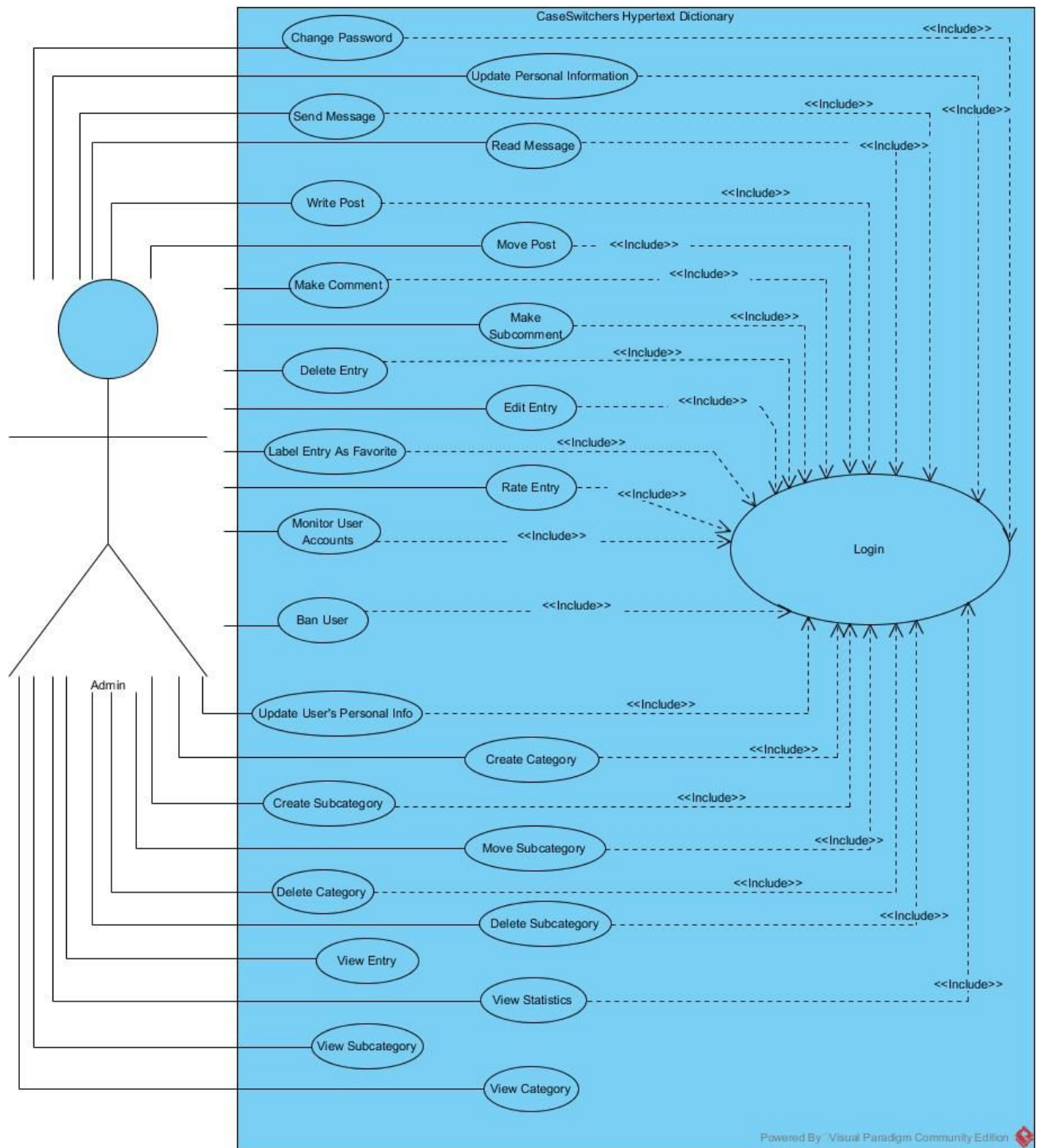
- **Label Entry as Favorite:** If users like an entry, they can label it as Favorite so that whenever they want, they can easily access the entry from Favorites part.
- **View Entry:** Users can view their entries as well as other users' entries.
- **View Category:** Users can view entries belongs to the selected category. In order to view a category, users are not required to log in to the system.
- **View Subcategory:** Users can view entries belongs to the selected subcategory of any categories. In order to view a subcategory, users are not required to log in to the system.



4.1.2. Admin

- **Change Password:** Admin can change his password after login.
- **Update Personal Information:** Admin can update his personal information. When changing the username and e-mail, it should be checked whether the new entered username and e-mail are unique or not.
- **Send Message:** Admin can send messages to other users.
- **Read Message:** Admin can read his messages.
- **Write Post:** Admin can write a post as he is a user of the system.
- **Move Post:** Admin can move entries from one subcategory to another.
- **Make Comment:** Admin can make comment about other users' entries if he wants.
- **Make Subcomment:** Admin can make subcomment to other users' entries if he wants.
- **Delete Entry:** Admin can delete user's posts.
- **Edit Entry:** Admin can edit his own entries as well as with the other users entries.
- **Label Entry as Favorites:** If admin likes an entry, he can label it as "Favorite" so that whenever he wants, he can easily access the entry from his Favorites part.
- **Rate Entry:** Admin can rate entries of other users. If he likes an entry, he can rate it as "up" or if he doesn't like it, he can rate it as "down".
- **Monitor User Accounts:** Admin can view user's accounts if he wants.
- **Ban User:** Admin can ban users if he thinks user posts or comments improperly.
- **Update User's Personal Info:** Admin can also update user's personal information.

- **Create Category:** Admin can create categories.
- **Create Subcategory:** Admin can create subcategories if he wants.
- **Move Subcategory:** Admin can move a subcategory under another category.
- **Delete Category:** Admin can delete a category if he finds it improper or duplicate.
- **Delete Subcategory:** Admin can delete a subcategory if he finds it improper or duplicate.
- **View Entry:** Admin can view his entries as well as other users' entries. View entry doesn't require login, any guests who are not signed up to the system can view the entries.
- **View Statistics:** Admin can view total number of posts, comments and users. In addition to these, he can view total number of posts for all categories. Simply, this feature enables admin to display the general statistics about the site.
- **View Subcategory:** Admin can view subcategories belongs to the a category. View subcategory doesn't require login, any guests who are not signed up to the system can view the subcategories.
- **View Category:** Admin can view categories. View category doesn't need login, any guests who are not signed up to the system can view the categories.



4.2. Algorithms

4.2.1. Logical Requirements

In order to get true information from the system, we have to prevent logical errors. As logical errors occur with attributes of date and time types, our system should carefully check the boundary dates.

For example, in Entry table we have a creationdate attribute to store the creation date of each entry. Similarly, we have an attribute named date_of_registration in User table which keeps track of the registration date of the user. Considering these two attributes, if user owns an entry, the creationdate of entry cannot come before the date_of_registration of that user.

Similarly, in Messages relationship we have a timestamp attribute to specify the send dates of the messages. In this respect, timestamp of the messages which are send and received only by two users, cannot exceed the maximum of day_of_registration of two users. That is, if one user registered to the system on timestamp 2017-07-15 12:33:01 and the other one is on 2017-09-01 15:00:35, then the timestamp of the messages between these users cannot be earlier than 2017-09-01 15:00:35.

Such restrictions are required to prevent possible logical failures.

4.2.2. Entry Related Algorithms

Main contents in the website are kept as entries. When a post or comment is created, its data will be stored in Entry table and ID of it will also be copied to Post or Comment entity, depending on whether the entry is a comment or a post.

Deletion of entries also require an algorithm. When an entry is deleted by an admin, it would not be enough to simply delete the entry from Entry table. The entry's other dependent data are also deleted from the database. More specifically, if an entry is deleted, the tuples containing the ID of the entry will be deleted from Favorites, Rates and Owns. Furthermore, if the entry is a post, all of its comments and subcomments are also deleted. If the entry is a comment, parent post of it will not be touched but its subcomments are deleted.

4.3. Data Structures

The relation schemas we created uses Numeric Types, String types, Times and Date Types.

Numeric types are used in order to store numeric data such as ids and passwords. For numeric types, we used `INT`.

String types are used to store any attributes composed of characters such as emails, usernames and topic names. For String types, we used VARCHAR.

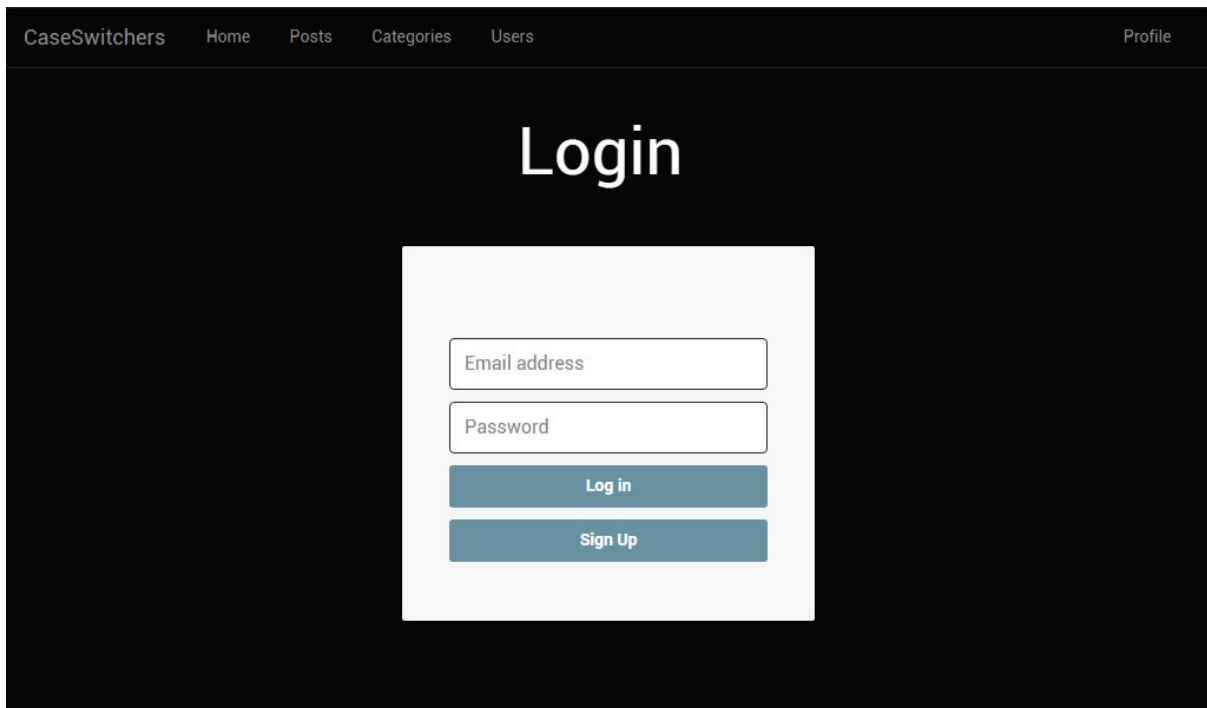
Date and Time types are used to keep time values. For Date and time types, we used TIMESTAMP.

In order to keep the contents of entries and messages, we used MEDIUMTEXT.

TINYTEXT is used to keep the content of profile infos of users.

5. USER INTERFACE DESIGN AND SQL STATEMENTS

5.1. Login Screen



Inputs: @email, @password

Process: Users and admin will login to their accounts by entering their email addresses and passwords.

SQL Statements:

SELECT ID

FROM User

WHERE email = @email **AND** password = @password

5.2. Sign Up Screen

The screenshot shows a web application interface with a dark blue header. The header contains navigation links: 'CaseSwitchers', 'Home', 'Posts', 'Categories', 'Users', and 'Profile'. The main content area is a light blue box with the title 'Sign Up' in large, bold, black text. Below the title are four input fields: 'Username', 'Email', 'Enter Password', and 'Re-enter Password'. The 'Username' and 'Email' fields have green checkmarks at the end, indicating they are valid. The 'Enter Password' field has a red warning triangle at the end, indicating it is invalid. The 'Re-enter Password' field is empty. Below the input fields is a small text link: 'You can set your other profile details after registration.' At the bottom of the form is a red 'Register' button.

Inputs: @ID, @username, @password, @repassword, @email, @date, @default_userlevel

Process: Users will be able to sign up by entering a username and password. Password will be confirmed by being asked twice.

SQL Statements:

INSERT INTO User

VALUES(@ID, '@username', '@password', NULL, @email, NULL , '@default_userlevel', '@date', NULL, NULL)

WHERE @password = @repassword

5.3. Category Specific Screen / Creating a New Post

[CaseSwitchers](#) [Home](#) [Posts](#) [Categories](#) [Users](#) [Profile](#)

Category: *Blockchain*

Entry	Author	Rating	Comments	Subcategory
Explain Blockchain in 3 words	cspro	55	122	Subcategory1
Future of Cryptocurrencies	jack30	43	22	Subcategory2
By 2040, There Will Be No World Without Bitcoin	h4x0r	55	21	Subcategory3
Bitcoin will hit \$10.000 by the end of 2017	aug30	11	403	Subcategory4
I've invested more than \$100.000 in Bitcoin, AMA!	kevin17	378	555	Subcategory5
The Lightning Network	cryptoislove	389	59	Subcategory6
Most profitable altcoins	chocho03	77	17	Subcategory7

New Post

Post Title

Select Category

Select Subcategory

Category Specific Screen

New Post Screen

Inputs: @ID, @creationdate, @content, @topicname, @c_id, @s_id
 * @ID is the post id, @c_id is category id, @s_id is subcategory id.

Process: Users will be able to display posts that are under a specific category and contribute to that category by submitting new posts. To submit a new post; users will enter a post title, select category for the post, optionally select a subcategory from the selected category for the post and write the content of their posts.

SQL Statements:

INSERT INTO Entry

VALUES(@ID, '@creationdate', '@content', '@topicname')

INSERT INTO Owns

VALUES(@ID, @u_id)

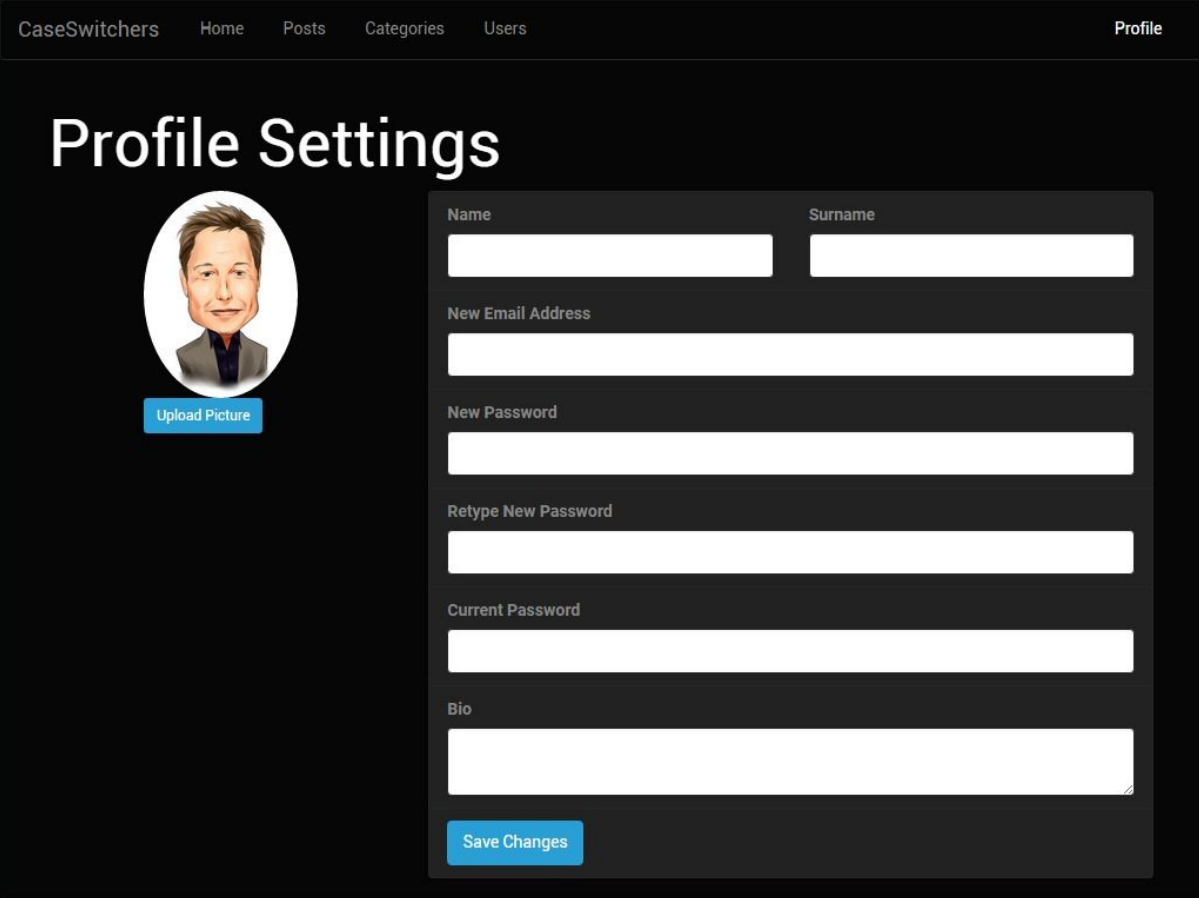
INSERT INTO Post

VALUES(@ID)

INSERT INTO PostCategory

VALUES (@ID, @c_id, @s_id)

5.4. Update Profile Information Screen



Inputs: @ID, @name, @surname, @newpassword, @renewpassword, @newemail @curpassword, @bio
*@ID is the user id.

Process: Users will be able to set and update their personal information in profile settings screen. Users can enter their name and surname, write their bio and change their passwords through this screen. In order to save the changes, users need to enter their current password correctly, otherwise the changes will not be saved.

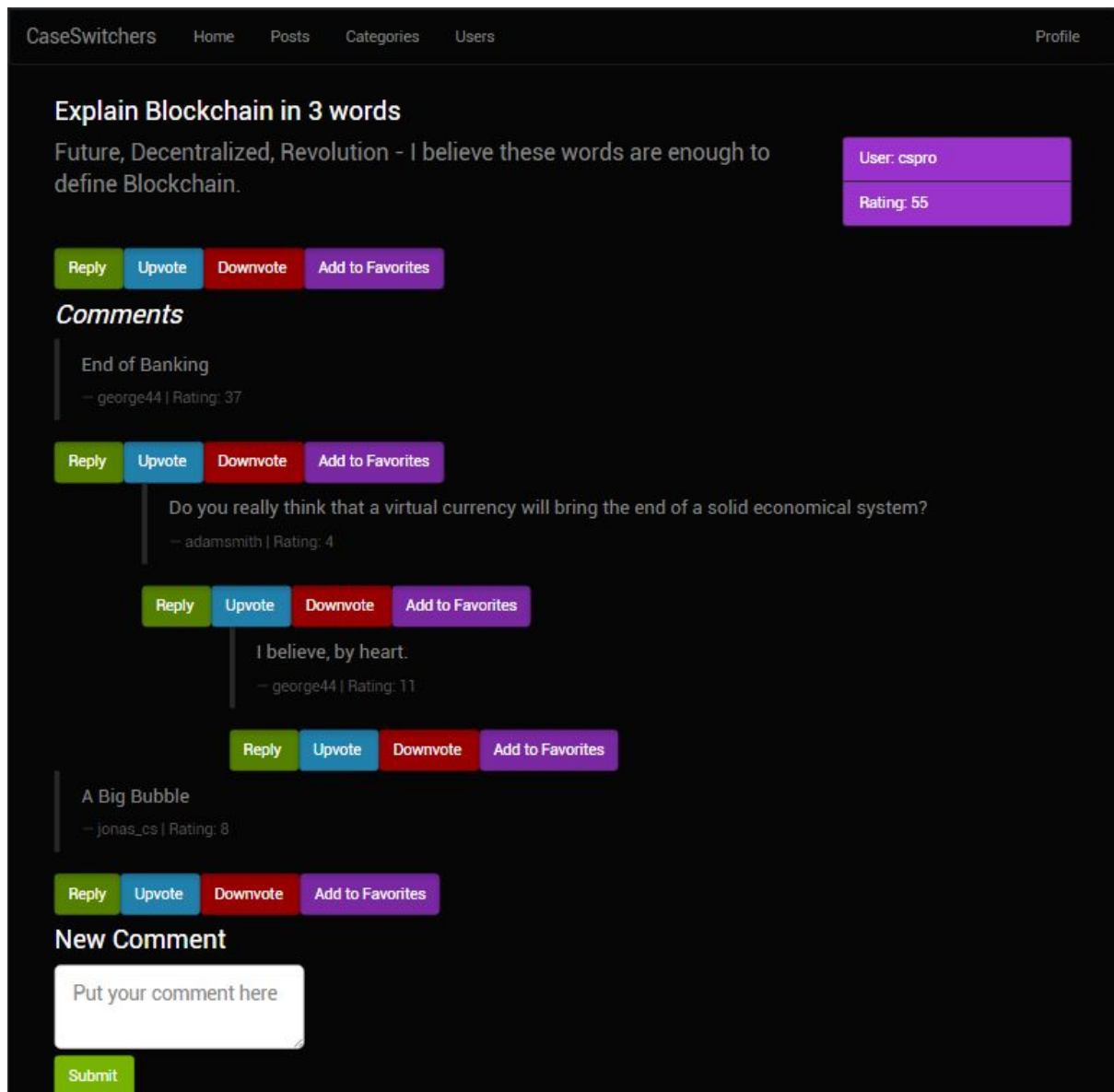
SQL Statement:

UPDATE User

SET name = @name, surname = @surname, newpassword = @newpassword, renewpassword = @renewpassword, email = @newemail, bio = @bio

WHERE ID = @ID **AND** password = @curpassword

5.5. Post Specific Screen



5.5.1. Creating a Comment

Inputs: @ID, @creationdate, @content, @topicname, @p_id

*@ID is the comment id, @p_id is the post id, @u_id is the user id.

Process: Users will be able to make comment on posts. They will simply enter a content for the comment and submit it.

SQL Statements:

INSERT INTO Entry

VALUES(@ID, '@creationdate', '@content', '@topicname')

INSERT INTO Owns

VALUES(@ID, @u_id)

INSERT INTO Comment

VALUES(@ID)

INSERT INTO PostComments **VALUES** (@p_id, @ID)

5.5.2. Creating a Subcomment

Inputs: @ID, @creationdate, @content, @topicname, @c_id

*@ID is the subcomment id, @p_id is the post id, @c_id is the comment id, @u_id is the user id.

Process: Users will be able to make subcomment on a comment. They will simply enter a content for the subcomment and submit it.

SQL Statements:

INSERT INTO Entry

VALUES(@ID, '@creationdate', '@content', '@topicname')

INSERT INTO Owns

VALUES(@ID, @u_id)

INSERT INTO SubComments

VALUES (@c_id, @ID)

5.5.3. Rate an Entry

Inputs: @e_id, @u_id, @vote

*@e_id is the entry id, @u_id is the user id.

Process: Users will be able to rate entries. They can rate comments or posts by upvoting or downvoting them.

SQL Statements:

INSERT INTO Rates **VALUES** (@e_id, @u_id, @vote)

5.5.4. Add an entry to favorites

Inputs: @e_id, @u_id

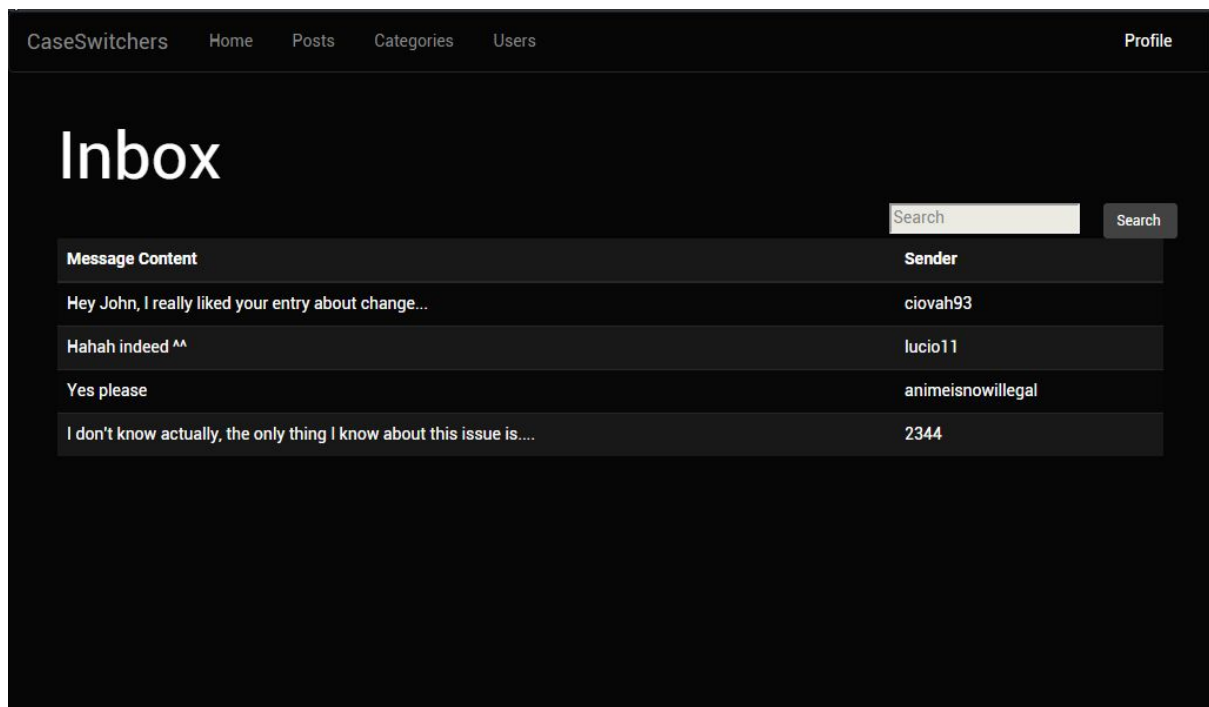
*@e_id is the entry id, @u_id is the user id.

Process: Users will be able to add an entry to their favorites.

SQL Statements:

INSERT INTO Favorites **VALUES** (@e_id, @u_id)

5.6. Message Screen



5.6.1. Sending a Message

Inputs: @sender_id, @receiver_id, @timestamp, @content

Process: Users will be able to send direct messages to each other.

SQL Statement:

```
INSERT INTO Messages VALUES(@sender_id, @receiver_id, @timestamp,
'@content')
```

5.6.2. Searching for a Message

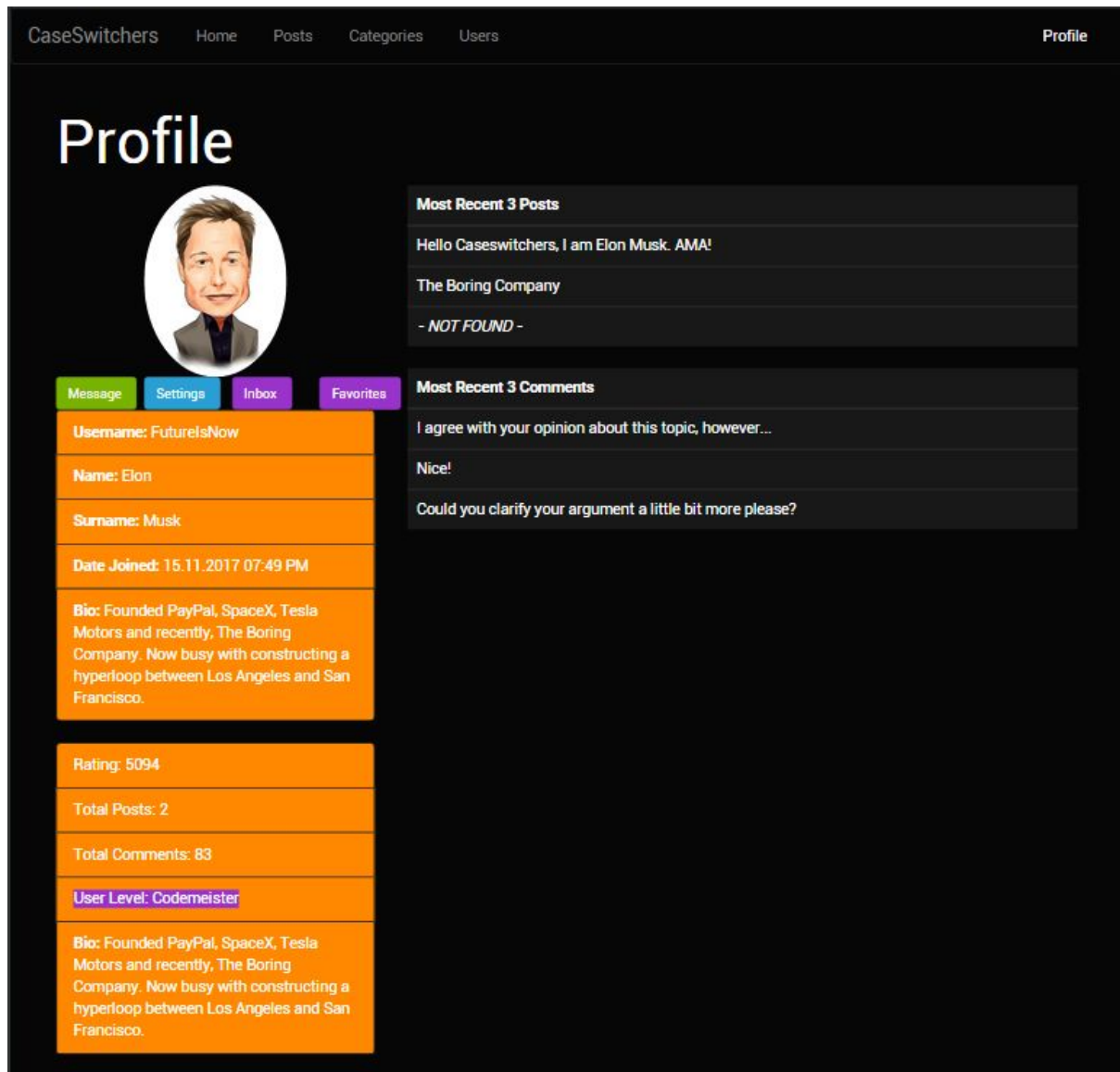
Inputs: @searchkey, @u_id

Process: Users will be able to search for the messages from a specific user by typing the sender's username to the search bar.

SQL Statement:

```
SELECT* FROM Messages WHERE receiver_id = u_id AND searchkey = sender_id
```

5.7. User Profile Screen



Inputs: @user_id

Process: Users will be able to display their own profile by clicking on Profile tab or any other user's profile by clicking on their usernames. User profile screen will display user information as well as the most recent 3 posts and the most recent 3 comments from the user.

SQL Statements:

//displays username

SELECT username **FROM** User **WHERE** ID = @user_id

//displays name

SELECT name **FROM** User **WHERE** ID = @user_id

//displays surname

SELECT surname **FROM** User **WHERE** ID = @user_id

//displays date of registration

SELECT date_of_registration **FROM** User **WHERE** ID = @user_id

//displays bio

SELECT bio **FROM** User **WHERE** ID = @user_id

//displays rating

WITH userentries

AS (**SELECT** e_id, u_id **FROM** Entry, Owns, **WHERE** (Owns.e_id = Entry.ID **AND** u_id = @user_id))

SELECT u_id, **sum**(rating) **FROM** Rates, userentries **WHERE** (Rates.e_id = userentries.e_id **AND** u_id = @user_id)

//displays number of posts

WITH userposts

AS (**SELECT** Post.ID, u_id **FROM** Entry, Owns, Post **WHERE** (Owns.e_id = Entry.ID **AND** Post.ID = Entry.ID **AND** u_id = @user_id))

SELECT **count**(*) **FROM** userposts

//displays number of comments

WITH usercomments

AS (**SELECT** Comment.ID, u_id **FROM** Entry, Owns, Comment **WHERE** (Owns.e_id = Entry.ID **AND** Comment.ID = Entry.ID **AND** u_id = @user_id))

SELECT **count**(*) **FROM** usercomments

//displays userlevel

SELECT userlevel **FROM** User **WHERE** ID = @u_id --displays user level

//displays most recent 3 posts of the user

WITH userposts

AS (SELECT Post.ID, u_id, content, creationdate FROM Entry, Owns, Post WHERE (Owns.e_id = Entry.ID AND Post.ID = Entry.ID AND u_id = @user_id))
SELECT content FROM userposts ORDER BY creationdate DESC LIMIT 3

//displays most recent 3 comments of the user

WITH usercomments

AS (SELECT Comment.ID, u_id, content, creationdate FROM Entry, Owns, Comment WHERE (Owns.e_id = Entry.ID AND Comment.ID = Entry.ID AND u_id = @user_id))
SELECT content FROM userposts ORDER BY creationdate DESC LIMIT 3

5.8. Admin Panel

CaseSwitchers

Home

Posts

Categories

Users

Profile

Admin Panel

Total Posts: 12903

Total Comments: 45940

Total Categories: 23

Total SubCategories: 53

Total Users: 65038

Edit Post

Post ID

Select Category

Select Subcategory

Put your post here

Submit

Delete Post

Post ID

Submit

Edit Comment

Comment ID

Put your comment here

Submit

Delete Comment

Comment ID

Submit

Ban User

User ID

Submit

Unban User

User ID

Submit

Create Category

Category Name

Submit

Delete Category

Category ID

Submit

Create Subcategory

Select Category

Subcategory Name

Submit

Change Subcategory Parent

Select Category

Subcategory ID

Submit

Update User Bio

User ID

Name

Surname

Put user bio here

Submit

Admin panel will display the total number of posts, total number of comments, total number of categories, total number of subcategories, total number of users.

SQL Statements:

//Displays total number of posts

SELECT count(*) FROM Entry NATURAL JOIN Post

//Displays total number of users

SELECT count(*) FROM Users

//Displays total number of comments

SELECT count(*) FROM Entry NATURAL JOIN Comments

//Displays total number of categories

SELECT count(*) FROM Category

//Displays total number of subcategories

SELECT count(*) FROM Subcategory

Admin panel will allow admin to perform following operations:

5.8.1. Edit Post

Inputs: @p_id , @content, @category, @subcategory

*@p_id is post id.

Process: Admin will be able to edit any post by changing its content, its category and subcategory. Admin needs to specify the id of the post that will be edited.

SQL Statement:

UPDATE Entry

SET content = @content, category = @category, subcategory = @subcategory

WHERE ID = @p_id

5.8.2. Delete Post

Inputs: @p_id

*@p_id is post id.

Process: Admin will be able to delete any post. Admin needs to specify the id of the post that will be deleted.

SQL Statement:

DELETE FROM Entry

WHERE ID = @p_id

DELETE FROM Post
WHERE ID = @p_id

5.8.3. Edit Comment

Inputs: @c_id , @content
*@c_id is comment id.

Process: Admin will be able to edit any comment by changing its content. Admin needs to specify the id of the comment that will be edited.

SQL Statement:

UPDATE Entry
SET content = @content
WHERE ID = @c_id

5.8.4. Delete Comment

Inputs: @c_id
*@c_id is comment id.

Process: Admin will be able to delete any comment. Admin needs to specify the id of the comment that will be deleted.

SQL Statement:

DELETE FROM Entry
WHERE ID = @c_id

DELETE FROM Comment
WHERE ID = @c_id

5.8.5. Ban User

Inputs: @u_id, @a_id.
*@u_id is the user id, @a_id is the admin id.

Process: Admin will be able to ban any user. Admin needs to specify the user id that will be banned.

SQL Statement:

INSERT INTO BannedUsers **VALUES** (@u_id, @a_id)

5.8.6. Unban User

Inputs: @u_id, @a_id

*@u_id is the user id, @a_id is the admin id.

Process: Admin will be able to remove ban of the any banned user. Admin needs to specify the user id whose ban will be removed.

SQL Statement:

DELETE FROM BannedUsers **WHERE** banned_id = @u_id **AND** admin_id = @a_id

5.8.7. Create Category

Inputs: @c_id, @c_name

*@c_id is the category id and @c_name is the category name.

Process: Admin will be able to create a new category by submitting its name.

SQL Statement:

INSERT INTO Category **VALUES** (@c_id, '@c_name')

5.8.8. Delete Category

Inputs: @c_id

*@c_id is the category id.

Process: Admin will be able to delete any category by specifying its id.

SQL Statement:

DELETE FROM Category **WHERE** ID = @c_id

5.8.9. Create Subcategory

Inputs: @sub_id, @c_name, @sub_name

*@sub_id is the subcategory id, c_name is the category name and @sub_name is the subcategory name.

Process: Admin will be able to create a new subcategory by submitting its name and selecting its parent category from the category list.

SQL Statement:

WITH c_id **AS** (**SELECT** ID **FROM** Category **WHERE** categoryname = @c_name)

INSERT INTO Subcategory **VALUES** (@sub_id, c_id, '@sub_name')

5.8.10. Change Subcategory Parent

Inputs: @sub_id, @c_name

*@sub_id is the subcategory id, c_name is the category name.

Process: Admin will be able to change a subcategory's parent category by submitting subcategory id and selecting a parent category from the category list.

SQL Statement:

WITH new_c_id **AS** (**SELECT** ID **FROM** Category **WHERE** categoryname = @c_name)

UPDATE Subcategory

SET c_id = new_c_id

WHERE sub_id = @sub_id

5.8.11. Update User Bio

Inputs: @u_id, @u_name, @u_surname, @bio

*@u_id is the user id, @u_name is the user's name, @u_surname is the user's surname.

Process: Admin will be able to update the bio of any user, user's name and surname by specifying the user id.

SQL Statement:

UPDATE User

SET bio = @bio, name = @u_name, surname = @u_surname

WHERE ID = @u_id

6. ADVANCED DATABASE COMPONENTS

6.1. Views

6.1.1. Users Messages View

Users can only access to messages which are sent to or sent by them. They cannot access any other messages which concerns other users.

```
CREATE VIEW user_messages as
  SELECT * from Messages
  WHERE sender_id = @u_id OR receiver_id = @u_id
```

*@u_id is ID of the user who is logged in.

6.2. Stored Procedures

Using stored procedures simplify the design of the database system. Instead of writing a long query each time we want to use, we can instead define a procedure which executes the desired query when called. They work like methods in Java, and they are pretty useful especially in the cases when we need to execute a query quite often and do not want to repeatedly fill the code space with long queries. We will simply pass the required parameters and call the stored procedure as we need.

6.2.1. Get the Rating of a User by ID

```
USE mydb;
DROP PROCEDURE IF EXISTS getRatingByUserID;
delimiter //
CREATE DEFINER='root'@'localhost' PROCEDURE `getRatingByUserID` (param
INT)
BEGIN
SET @s = 'WITH userentries
  AS (SELECT e_id, u_id FROM Entry, Owns, WHERE (Owns.e_id =
Entry.ID AND u_id = @user_id))
```

```

        SELECT u_id, sum(rating) FROM Rates, userentries WHERE
        (Rates.e_id = userentries.e_id AND u_id = @user_id);
SELECT @s;
PREPARE stmt FROM @s;
EXECUTE stmt;
END
delimiter ;

```

6.2.2. Get the most recent 3 comments of a user

```

USE mydb;
DROP PROCEDURE IF EXISTS getMostRecentCommentsOfUser;
delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE
`getMostRecentCommentsOfUser` (param INT)
BEGIN
SET @s = 'WITH usercomments
        AS (SELECT Comment.ID, u_id, content, creationdate FROM Entry,
        Owns, Comment WHERE (Owns.e_id = Entry.ID AND Comment.ID =
        Entry.ID AND u_id = @user_id))
        SELECT content FROM userposts ORDER BY creationdate DESC
        LIMIT 3';
SELECT @s;
PREPARE stmt FROM @s;
EXECUTE stmt;
END
delimiter ;

```

6.2.3. Get the most recent 3 posts of a user

```

USE mydb;
DROP PROCEDURE IF EXISTS getMostRecentPostsOfUser;
delimiter //
CREATE DEFINER=`root`@`localhost` PROCEDURE `getMostRecentPostsOfUser`
(param INT)
BEGIN
SET @s = 'WITH userposts
        AS (SELECT Post.ID, u_id, content, creationdate FROM Entry, Owns,
        Post WHERE (Owns.e_id = Entry.ID AND Post.ID = Entry.ID AND u_id
        = @user_id))
        SELECT content FROM userposts ORDER BY creationdate DESC
        LIMIT 3';
SELECT @s;
PREPARE stmt FROM @s;
EXECUTE stmt;

```

END
delimiter ;

6.3. Reports

6.3.1. Total Number of Posts

```
SELECT count(*)  
FROM Entry NATURAL JOIN Post
```

6.3.2. Total Number of Comments

```
SELECT count(*)  
FROM Entry NATURAL JOIN Comments
```

6.3.3. Total Number of Categories

```
SELECT count(*)  
FROM Category
```

6.3.4. Total Number of Subcategories

```
SELECT count(*)  
FROM Subcategory
```

6.3.5. Total Number of Users

```
SELECT count(*)  
FROM Users
```

6.3.6. Total Number of Posts under each Category

```
SELECT categoryname, count(p_id)  
FROM Category, PostCategory  
WHERE Category.ID = PostCategory.c_id  
GROUP BY Category.ID
```

6.3.7. Total Ratings of Each User in Descending Order

```
WITH userentries  
AS ( SELECT e_id, u_id FROM Entry, Owns, WHERE Owns.e_id = Entry.ID )  
SELECT u_id, sum(rating)  
FROM Rates, userentries  
WHERE Rates.e_id = userentries.e_id
```

GROUP BY u_id
ORDER BY sum(rating) DESC

6.4. Triggers

- When a post is deleted, the comments belong to that post will be deleted.
- When a post or comment is deleted, the total number of both posts and comments in the system will be decreased.
- When a post or comment is deleted, the favorites part of the users who favorite that entry will be updated.
- When a post or comment is deleted, ratings of it will also be deleted.
- When a category is deleted, the posts under it will be deleted.
- When a subcategory is deleted, the posts under it will be assigned to a *NULL* subcategory.

6.5. Constraints

- Users have to log in the system in order to create, rate, favorite entries and send messages to other users.
- Date of registration of the users are kept as a timestamp of instantaneous server time at the time of registration. Thus, it cannot be earlier or later than the exact registration time.
- Posts must have exactly one category.
- Posts can have at most one subcategory.
- Posts cannot be assigned to a subcategory which belongs to a category different than the post's category.
- Categories cannot be removed if there are one or more subcategories which belong to them. In order to remove a category, all of its subcategories must be removed first.
- Users cannot post empty entries.
- Users cannot send empty messages.
- Users cannot send messages to themselves.
- Users cannot rate their own entries or comments.
- Users can rate an entry only once as upward or downward. That is, users can change their ratings between upward or downward but they cannot rate the same entry twice as upwards or downwards.
- There can be at most 100000 registered users in the system.
- There can be at most 1000000 entries registered in the system.
- There can be at most 500 categories in the system.

- Topicname of a post, name of a category and subcategory, cannot be longer than 45 characters.
- Username, password, name, surname, email, userlevel and avatar location of a user cannot exceed 45 characters.
- Contents of messages and entries cannot exceed 3000 characters.
- Profile info of users can be at most 500 characters.

7. IMPLEMENTATION PLAN

We will implement a MySQL database system for the system. We are going to use MySQL Workbench to simplify the implementation of the database part. Back-end will be implemented in Java. We are going to use MySQL Connector/J library to connect the database to the back-end part of the system. We may also use other libraries if needed. For the front-end, we are planning to use PHP and Javascript to create the controllers; Bootstrap to create the view. Models will be stored in JSON objects which are generated by back-end.