Bilkent University

Department of Computer Engineering

CS 353 - Database Management Systems

Collaborative Hypertext Dictionary

Project Design

Group 03

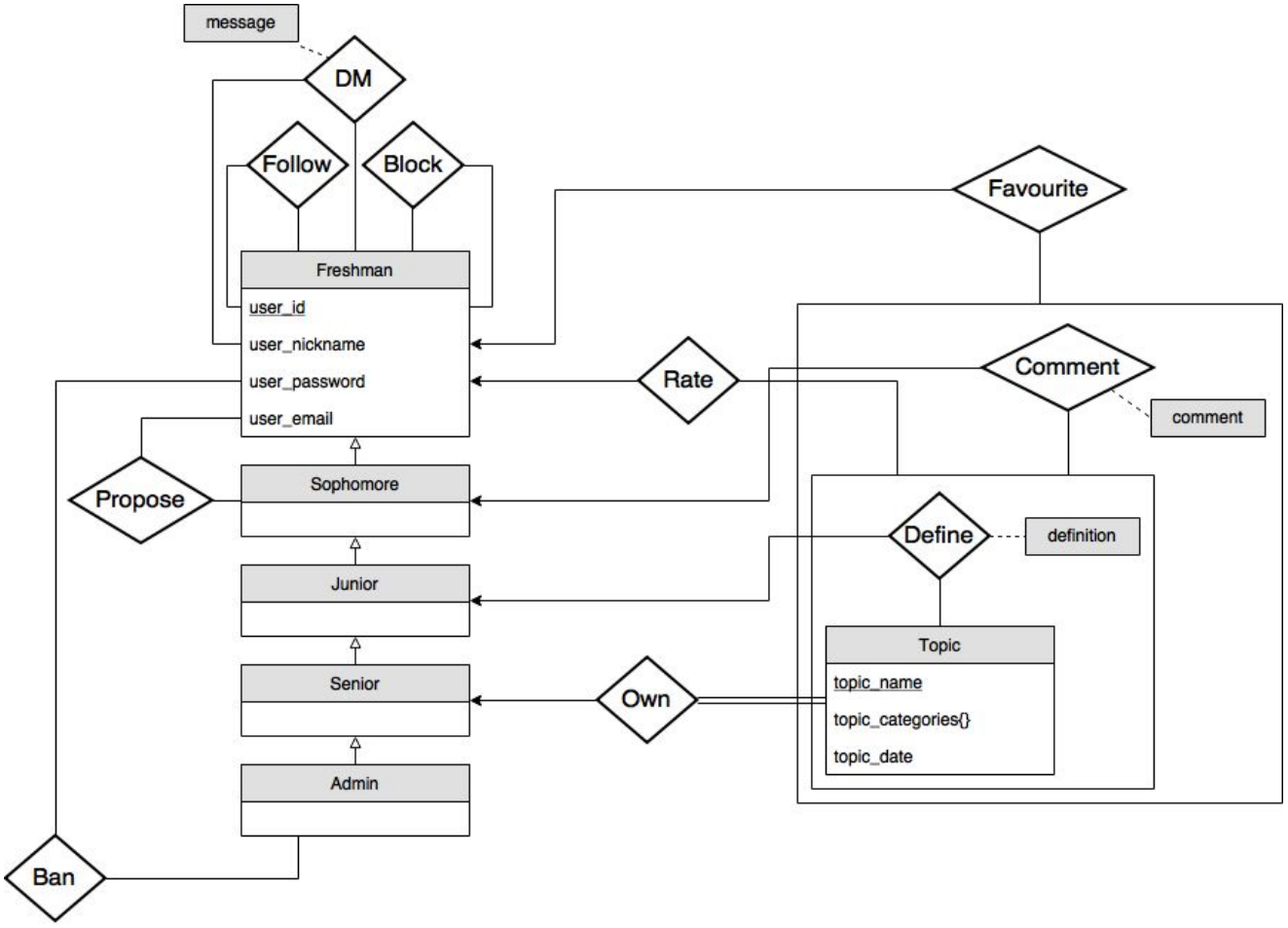
Rümeyza Dinçer	21400191
Furkan Arif Bozdağ	21401748
Mustafa Caner Çalışkaner	21401961
Ferhat Serdar Atalay	21300738

20.11.2017

Table of Contents

1. REVISED E/R MODEL

1.1. Previous E/R Model

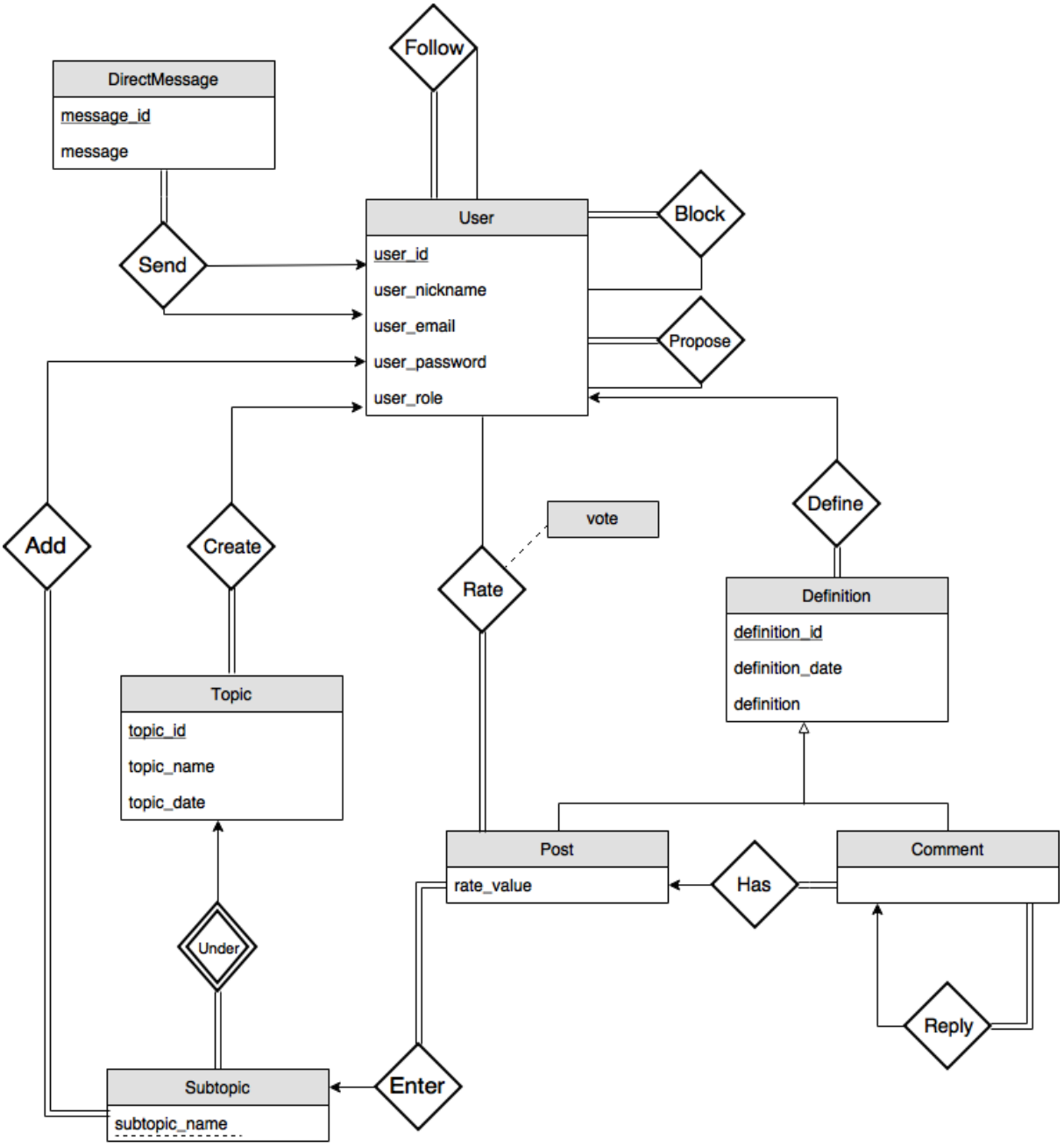


1.2. Changes Made in the Previous E/R Diagram

After receiving feedback for the E/R diagram that we proposed, the following changes have been made in order to implement our database appropriately :

- We revised our tables which consist of "Freshman, Sophomore, Junior, Senior, Admin" and relations "Follow, Block, DM, Propose, Ban". We decided to use a different design for this report since the previous one was very complicated.
  - We created a "User" entity instead of having different user tables and created "user_role" attribute which stands for "Freshman, Sophomore, Junior, Senior".
  - We realized that we do not need "Admin" in the E/R diagram and we removed "Admin" role and "Ban" relation from it.
  - We redesigned "Follow", "Propose", "Block" relations as recursive relations to "User" entity.
  - "DM" relation is substituted with "DirectMessage" entity and it is holding a unique message_id for each message between users.
    - 1-to-1 "Send" relation is added between "User" and "DirectMesaage" entities.
- We created "Definition", "Post", "Comment" entities where "Post" and "Comment" are subclasses of "Definition".
  - We made "Rate" relation, between "User" and "Post", many-to-many.
  - We added a 1-to-many recursive "Reply" relation to the "Comment" entity.
  - We added a many-to-many "Rate" relation with "vote" attribute between "User" and "Post" entities.
  - We added a 1-to-many "Has" relation between "Post" and "Comment" entities.
- We created a weak entity under "Topic" entity, which is called "SubTopic" with key "subtopic_name".
  - We added a 1-to-many "Create" relation between "User" and "Topic".
  - We added a 1-to-many "Enter" relation between "Post" and "Subtopic".
  - We added a 1-to-many "Add" relation between "User" and "Subtopic".

1.3. Revised E/R Diagram



2. RELATIONAL SCHEMAS

2.1. User

Relational Schema:

User(<u>user_id</u>, user_nickname, user_email, user_password, user_role)

Functional Dependencies:

user_id -> user_nickname, user_email, user_password, user_role

user_email -> user_id, user_nickname, user_password, user_role

Candidate Keys:

user_id or user_email

Selected Primary Key:

user_id

Normal Form:

BCNF

Table Definition:

create table user

(user_id	int(8) primary key,
user_nickname	varchar(16) not null,
user_email	varchar(245) not null,
user_password	varchar(32) not null,
user_role	int(1) not null);

2.2. Follow

Relational Schema:

Follow(<u>follower, followed</u>)

Functional Dependencies:

No nontrivial dependencies.

Candidate Keys:

{follower, followed}

Selected Primary Key:

follower, followed

Foreign Keys:

(follower, followed) ref. User(user_id)

Normal Form:

BCNF

Table Definition:

create table follow

(follower int(8),
followed varchar(8),
Primary Key(follower, followed)
Foreign Key(follower, followed) references user);

2.3. Block

Relational Schema:

Block(blocker, blocked)

Functional Dependencies:

No nontrivial dependencies.

Candidate Keys:

{blocker, blocked}

Selected Primary Key:

blocker, blocked

Foreign Keys:

(blocker, blocked) ref. User(user_id)

Normal Form:

BCNF

Table Definition:

create table block

(blocker int(8),
blocking varchar(8),
Primary Key(blocker, blocked),
Foreign Key(blocker, blocked) references user);

2.4. Propose

Relational Schema:

Propose(proposed, proposer)

Functional Dependencies:

No nontrivial dependencies.

Candidate Keys:

{proposed, proposer}

Selected Primary Key:

proposed, proposer

Foreign Keys:

(proposed, proposer) ref. User(user_id)

Normal Form:

BCNF

Table Definition:

create table propose

(proposed int(8),
proposer varchar(8),
Primary Key(proposed, proposer),
Foreign Key(proposed, proposer) references user);

2.5. DirectMessage

Relational Schema:

DirectMessage(message_id, message, sender_user, receiver_user)

Functional Dependencies:

message_id -> message

Candidate Keys:

message_id

Selected Primary Key:

message_id

Foreign Keys:

(sender_user, receiver_user) ref. User(user_id)

Normal Form:

BCNF

Table Definition:

```
create table directMessage
        (message_id         int(64) primary key,
         message            varchar(256) not null,
         sender_user        int(8) not null,
         receiver_user      int(8) not null,
         Foreign Key(sender_user) references user,
         Foreign Key(receiver_user) references user);
```

2.6. Topic

Relational Schema:

Topic(topic_id, topic_name, topic_date, creator_user)

Functional Dependencies:

topic_id -> topic_name, topic_date, creator_user

Candidate Keys:

topic_id

Selected Primary Key:

topic_id

Foreign Keys:

creator_user ref. User(user_id)

Normal Form:

BCNF

Table Definition:

```
create table topic
        (topic_id           int(64) primary key,
         topic_name         varchar(256) not null,
         topic_date         date not null,
         creator_user       int(8) not null,
         Foreign Key(creator_user) references user);
```

2.7. Subtopic

Relational Schema:

Subtopic(topic_id, subtopic_name, adder_user)

Functional Dependencies:

topic_id, subtopic_name-> adder_user

Candidate Keys:

{topic_id, subtopic_name}

Selected Primary Key:

topic_id, subtopic_name

Foreign Keys:

topic_id ref. Topic

adder_user ref. User(user_id)

Normal Form:

BCNF

Table Definition:

create table subtopic

(topic_id int(64),
subtopic_name varchar(256),
adder_user int(8) not null,
Primary Key(topic_id, subtopic_name),
Foreign Key(topic_id) references topic,
Foreign Key(adder_user) references user);

2.8. Definition

Relational Schema:

Definition(definition_id, definition_date, definition, definer_user)

Functional Dependencies:

definition_id->definition_date, definition, definer_user

Candidate Keys:

definition_id

Selected Primary Key:

definition_id

Foreign Keys:

definer_user ref. User(user_id)

Normal Form:

BCNF

Table Definition:

create table definition

(definition_id int(64) primary key,
definition_date date not null,
definer_user int(8) not null,
definition varchar(256) not null,
Foreign Key(definer_user) references user);

2.9. Post

Relational Schema:

Post(post_id, rate_value)

Functional Dependencies:

post_id->rate_value

Candidate Keys:

post_id

Selected Primary Key:

post_id

Foreign Keys:

post_id ref. Definition(definition_id)

Normal Form:

BCNF

Table Definition:

```
create table post
        (post_id            int(64) primary key,
         rate_value         int(64) not null,
         Foreign Key(post_id) references definition);
```

2.10. Enter

Relational Schema:

Enter(topic_id, subtopic_name, post_id)

Functional Dependencies:

No nontrivial dependencies

Candidate Keys:

{topic_id, subtopic_name, post_id}

Selected Primary Key:

topic_id, subtopic_name, post_id

Foreign Keys:

(topic_id, subtopic_name) ref. Subtopic

post_id ref. Definition(definition_id)

Normal Form:

BCNF

Table Definition:

create table enter

(topic_id int(64),
subtopic_name varchar(256),
post_id int(64) primary key,
Primary Key(topic_id, subtopic_name, post_id),
Foreign Key(topic_id, subtopic_name) references subtopic,
Foreign Key(post_id) references post);

2.11. Rate

Relational Schema:

Rate(user_id, post_id, vote)

Functional Dependencies:

user_id, post_id -> vote

Candidate Keys:

{user_id, post_id}

Selected Primary Key:

user_id, post_id

Foreign Keys:

user_id ref. User

post_id ref. Post

Normal Form:

BCNF

Table Definition:

create table rate

(user_id int(8),
post_id int(64),
vote int(1),
Primary Key(user_id, post_id),
Foreign Key(user_id) references user,
Foreign Key(post_id) references post);

2.12. Comment

Relational Schema:

Comment(comment_id, post_id)

Functional Dependencies:

No nontrivial dependencies

Candidate Keys:

{comment_id, post_id}

Selected Primary Key:

comment_id, post_id

Foreign Keys:

comment_id ref. Definition(definition_id)

post_id ref. Post

Normal Form:

BCNF

Table Definition:

create table comment

(comment_id int(64),
post_id int(64),
Primary Key(comment_id, post_id),
Foreign Key(comment_id) references definition,
Foreign Key(post_id) references post);

2.13. Reply

Relational Schema:

Reply(comment_id,replied_comment)

Functional Dependencies:

No nontrivial dependencies

Candidate Keys:

{comment_id,replied_comment}

Selected Primary Key:

comment_id, replied_comment

Foreign Keys:

(comment_id,replied_comment) ref. Comment(comment_id)

Normal Form:

BCNF

Table Definiton:

```
create table reply
        (comment_id          int(64),
         replied_comment     int(64),
         Primary Key(comment_id, replied_comment),
         Foreign Key(comment_id, replied_comment) references comment);
```

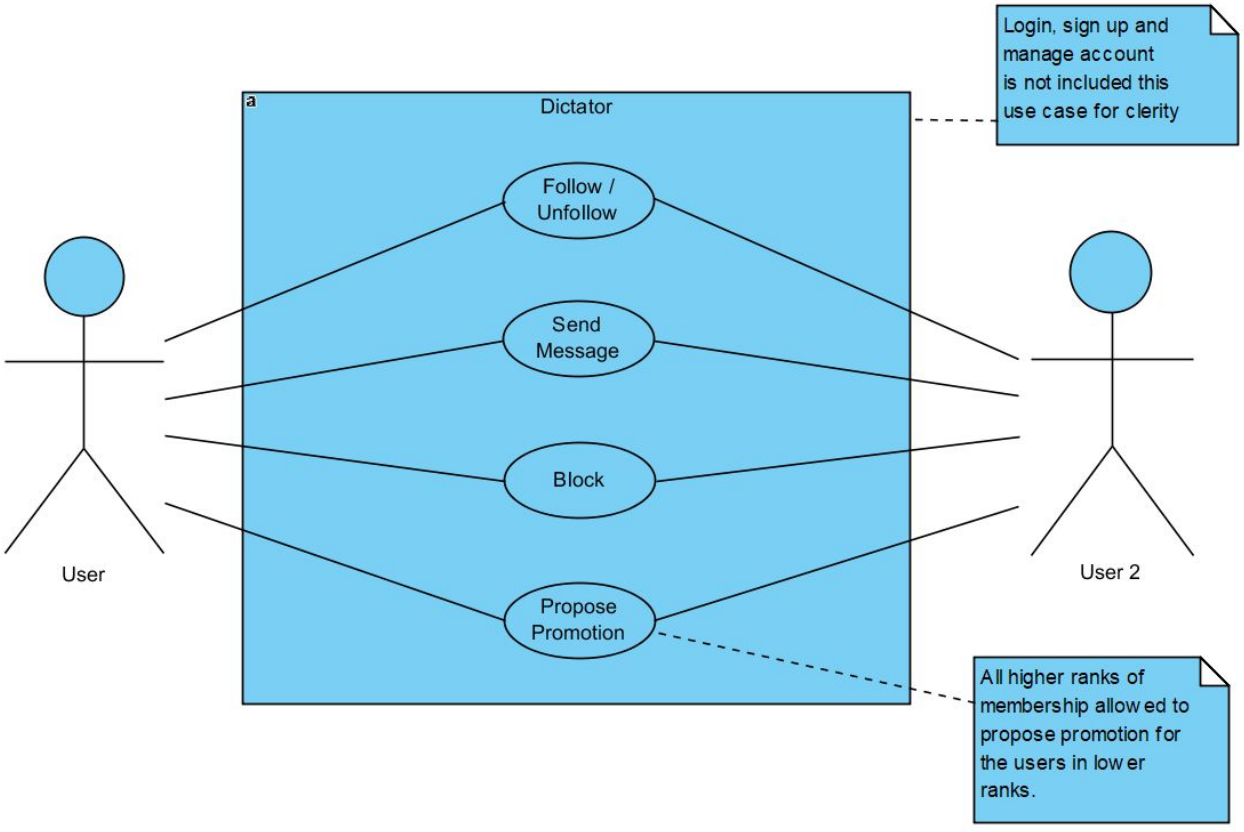
3. FUNCTIONAL COMPONENTS

3.1. Use Cases/Scenarios

Users in Dictator dictionary is categorised as **Freshman**, **Sophomore**, **Junior and Senior**. Therefore, their use cases differ from each other. Each category will be examined it this section. User use case is a use case diagram that shows actions every Dictator user can do regardless of their rank.

3.1.1. User Use Cases

The activities every Dictator user can do are sign up, login, manage account, follow/unfollow other users, send message to other user, block/unblock other users, and propose promotion for or report other users.

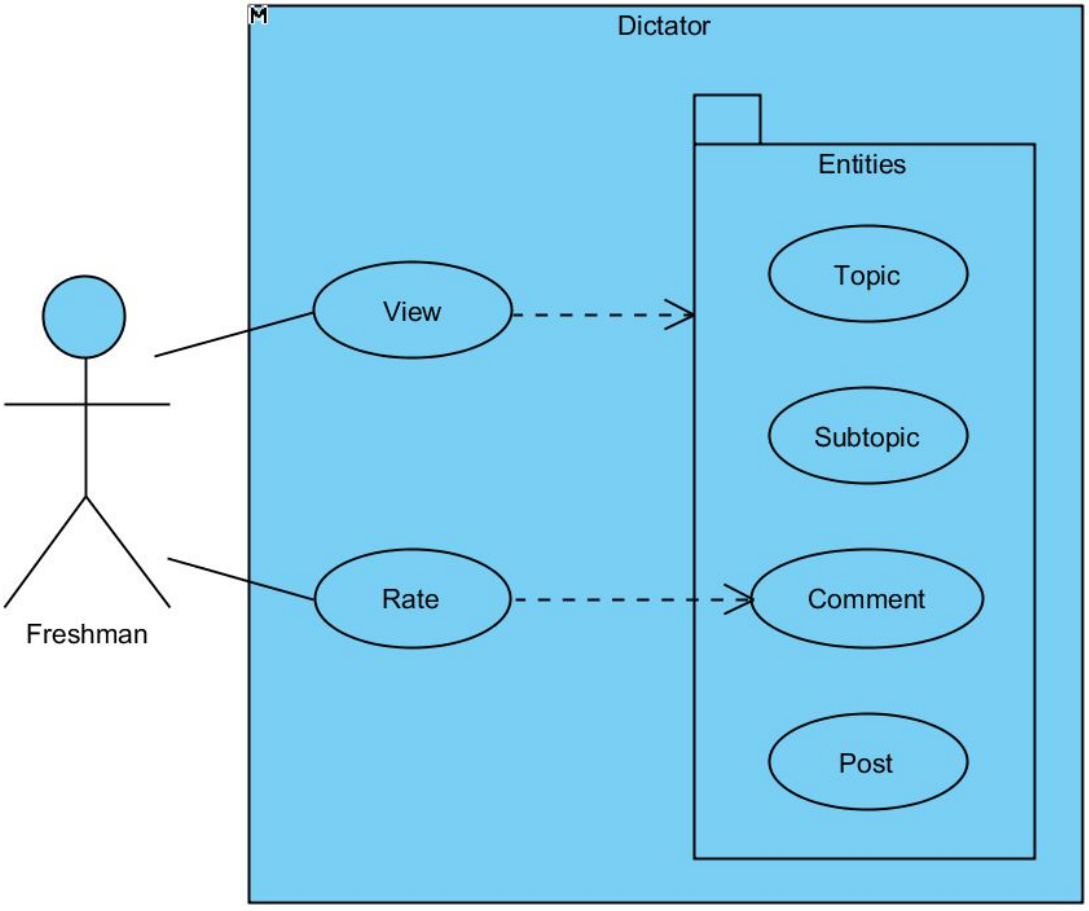


- **Login:** A user can login to the system using his/her email and password. Logging in to an account authorizes one to manage and update own account data and interact with the Dictator content and other users.

- **Sign Up:** A user can register to the Dictator dictionary by entering a unique user name, his/her email address and a password. Several accounts can not be registered using the same email address. Every users is ranked as Freshman at the registry.
- **Manage Account Credentials:** A user can update his/her username and password in this page. Username can only be changed with an available one in the system.
- **Follow / Unfollow:** A user can follow and after unfollow any user regardless of their rank. This will allow users to see their favorite write's posts to appear at their followings page.
- **Send Message:** A user can send message to another user by clicking the "Send Message" button that appears near their username. After this a new chat screen will be added to the messages page where the two can keep exchanging messages.
- **Block:** A user can block other users who he/she does not want to see the contents produced by the other user or does not want his/her contents to be seen by the other user. The user can later be unblocked by locating to that user's account via search option and clicking the unblock button appears in that specific users page after the block.
- **Propose Promotion:** All users except Freshman can propose promotions for the users with a lower rank from themselves. Every hundred proposal moves user to an upper rank. A Freshman cannot propose promotions since they do not have any user lower than their rank. A Senior cannot be proposed for promotion since there is no rank above.

3.1.1.1. Freshman Use Case

Every user signed up to the Dictator dictionary starts as a Freshman. A Freshman can do every common user activity which are login, manage account, follow/unfollow other users, send message to other user, block/unblock other users, and propose promotion for or report other users. The activities that can be done by the Freshman can be done by all the users above the Freshman rank. Every Freshman needs a hundred promotion proposals to become a Sophomore.

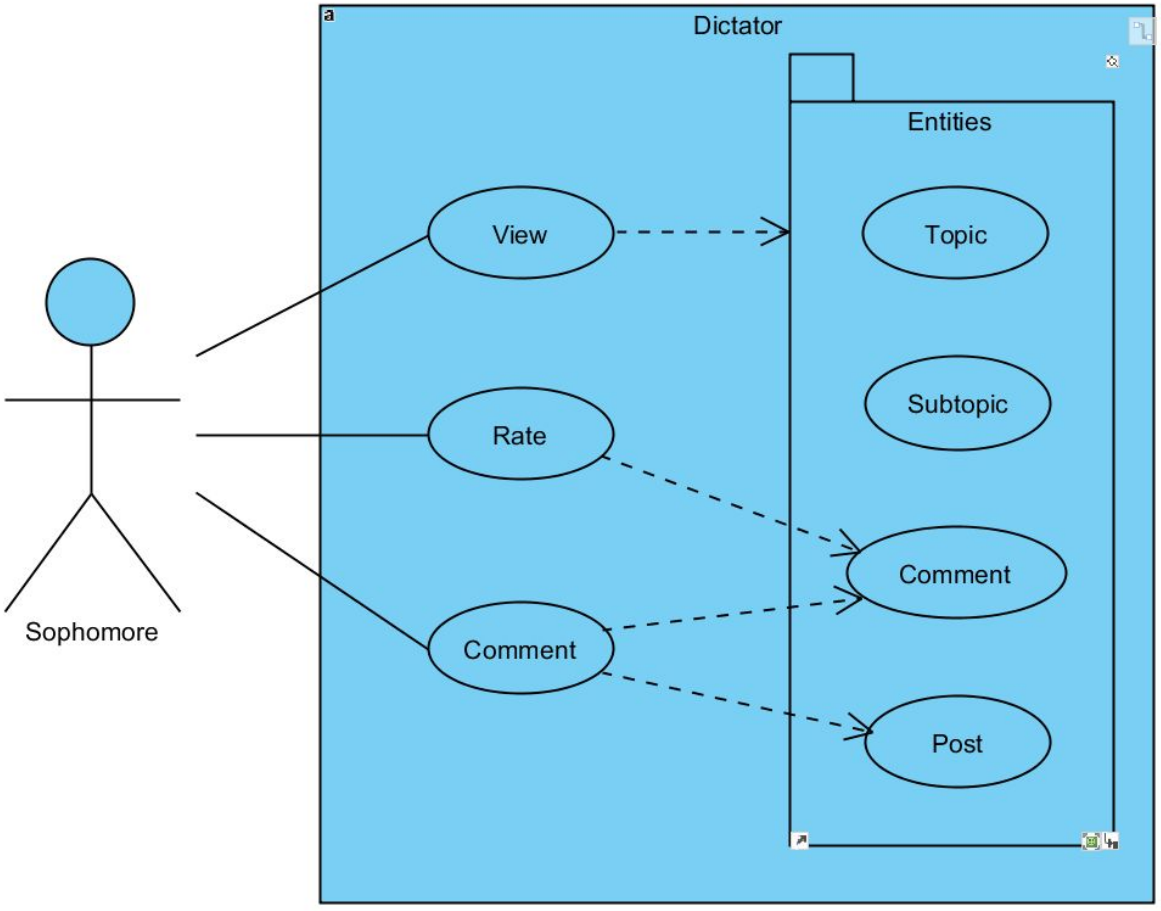


- **View:** All Freshman users can view topics, subtopics, posts and comments on them without any restrictions unless they are not blocked by the owner of the content. A content of the Dictator dictionary can be reached through the content url, links in the pages of the Dictator web site or the search bar.
- **Rate:** All Freshman users can rate the posts inside a topic by either liking or disliking that post. The likes and dislikes determines the rate value of the post. While each like

adds one point to the popularity rate of the post, each dislikes subtracts one point from it.

3.1.1.2. Sophomore Use Case

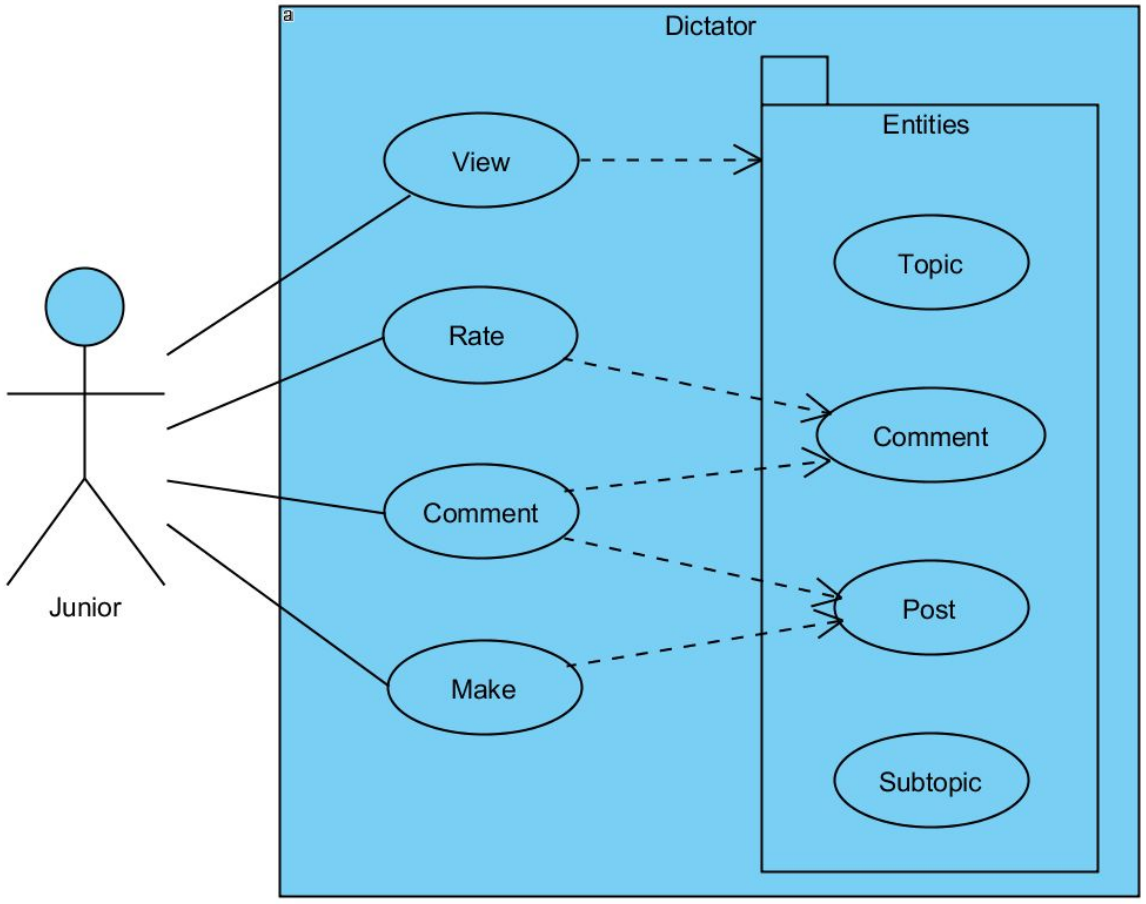
A Sophomore can do every common user action and all the actions a Freshman can do which are login, view contents, rate posts, manage account, follow/unfollow other users, send message to other user, block/unblock other users, and propose promotion for or report other users. Additionally, a Sophomore can write comments on the posts and the other comments under a post. The activities that can be done by the Sophomore can be done by all the users above the Sophomore rank. Every Sophomore needs a hundred promotion proposals to become a Junior.



- **Comment:** A Sophomore can comment on the posts inside a subtopic and also the other comments under the posts. This comments are shown when the specific post is clicked on as the post box expands downwards. Every post and comment can have several sub comments that are made by any Sophomore, Junior or Senior.

3.1.1.3. Junior Use Case

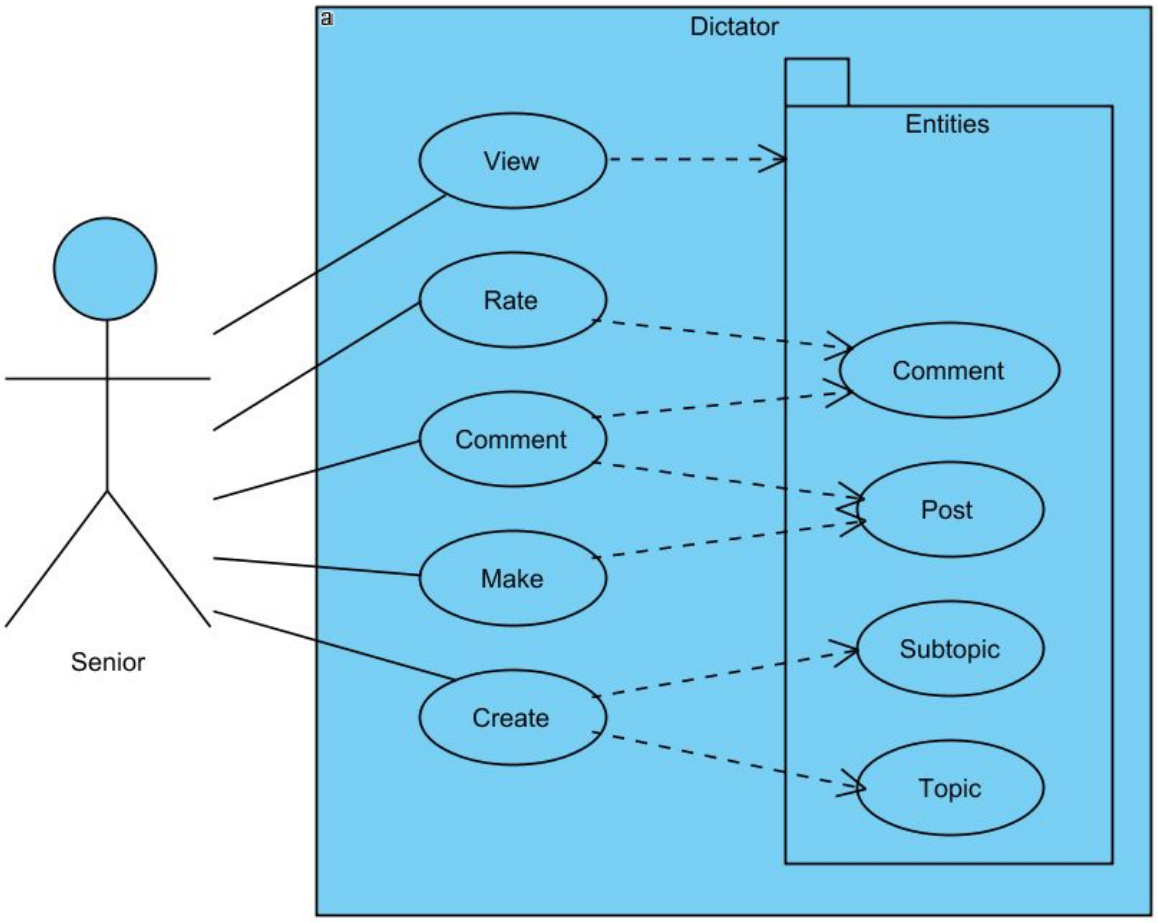
A Junior can do every common user action and all the actions a Freshman and Sophomore can do which are login, view contents, rate posts, comment on posts, comment on other comments, manage account, follow/unfollow other users, send message to other user, block/unblock other users, and propose promotion for or report other users. Additionally, a Junior can make posts on the subtopics. The activities that can be done by the Junior can be done by all the Senior rank users. Every Junior needs a hundred promotion proposals from Seniors to become a Senior.



- **Make Post:** A Junior can make posts under the subtopics created by the Seniors. These posts can be viewed at the subtopic page of the topics and rated by all the users. Each subtopic can have several posts made by any Junior or Senior.

3.1.1.4. Senior Use Case

A Senior has the maximum access to the activity flow of the Dictator dictionary. The Senior can do every common user action and all the actions the Freshman, Sophomore and Junior can do. These are login, view contents, make post, comment on posts, rate posts, comment on other comments, manage account, follow/unfollow other users, send message to other user, block/unblock other users, and propose promotion for or report other users. Additionally, a Senior can create new topics and subtopics.



- **Create Topic:** A Senior can create new topics. All topics need subtopics in order to have posts from other users in them. The popularity of a topic determined by the sum of the likes and dislikes -likes count as plus one and dislikes count as minus one- of the all posts under itself. These topics listed in the main page of the Dictator dictionary according to their popularity.

- **Create Subtopic:** A Senior can create several subtopics under a topic. Subtopics are fields were all the other Senior and Junior users can makes their posts about the topic. These subtopics are listed inside the topic page in the creational order for each topic.

3.2. Algorithms

3.2.1. Account Credentials Related Algorithms

A user has to have a unique user_nickname and user_email. Therefore, their name and emails will be checked against User table before given permission. The same restrictions will be applied on the change of the user_nickname later. Change for user_email for an account is not permitted.

3.2.2. User Interactions Related Algorithms

All users can follow a user once which means no entries for Follow table permitted for a user following other more than once. Users cannot unfollow users they did not followed in the first place.

All users can send messages to other users any number of times if they are not in the blocked list of that users.

All users can block other users once which means no entries for Block table permitted for a user to block the other more than once. Users cannot unblock users they did not blocked in the first place.

Users can propose promotion only with a lower rank from themselves. Every user gets only one right of proposal for each rank of a specific user.

3.2.3. User and Site Content Related Algorithms

All users can like or dislike each post only once which means no entries for rate table permitted for more than once.

3.2.4. Popularity Related Algorithms

3.2.4.1. Topic Popularity Algorithm

The popularity of a topic is calculated by the sum of rate_value attributes of all the posts under the topics' subtopics. The rate_value of a post goes up by one point for each like and goes down one point for each dislike.

3.2.4.2. User Popularity Algorithm

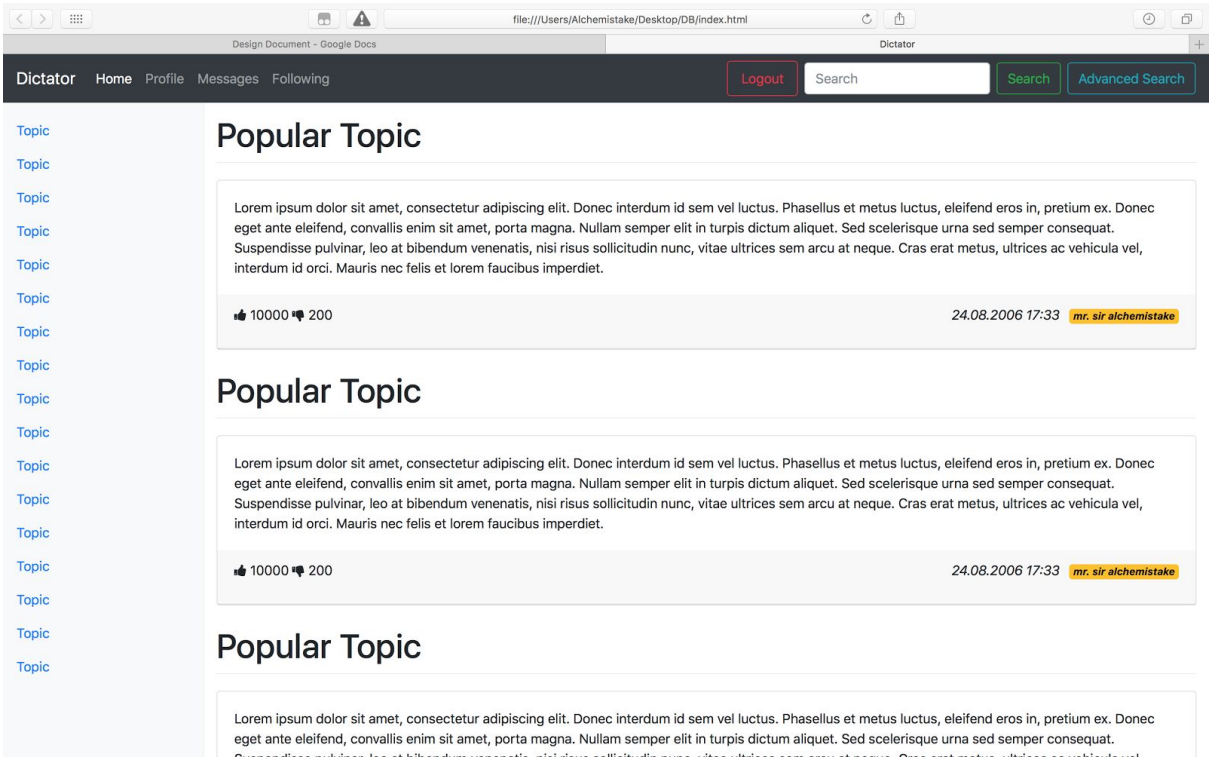
The popularity of a user is determined by the number of his/her followers summed with the number of promotion proposals they have and the user_role minus one times hundred. The last part of this algorithm is needed since the proposal number is reseted for every rank up and for each rank up every user needs a hundred proposals. The user_role minus one is needed since the lowest rank Freshman has a user_role value of one and cannot be multiplied by hundred since he/she did not go up rank before.

3.3. Data Structures

In our relational schema numeric type, string type, date type is used. Numeric type is used for user_id, message_id, definition_id, topic_id, vote and rate_value attributes. String type is user for user_nickname, user_email, user_password, definition, topic_name, subtopic_name and message attributes. Date type is necessary for definition_date and topic_date attributes.

4. USER INTERFACE DESIGN and CORRESPONDING SQL STATEMENTS

4.1. Home Page



Home page contains the popular topics of that day based on their ratings.

SQL Queries:

select topic_name, definition_date, definition, definer_user, rate_value

from topic, enter, post, definition

where post.post_id == enter.topic_id and post.post_id == definition.definition_id

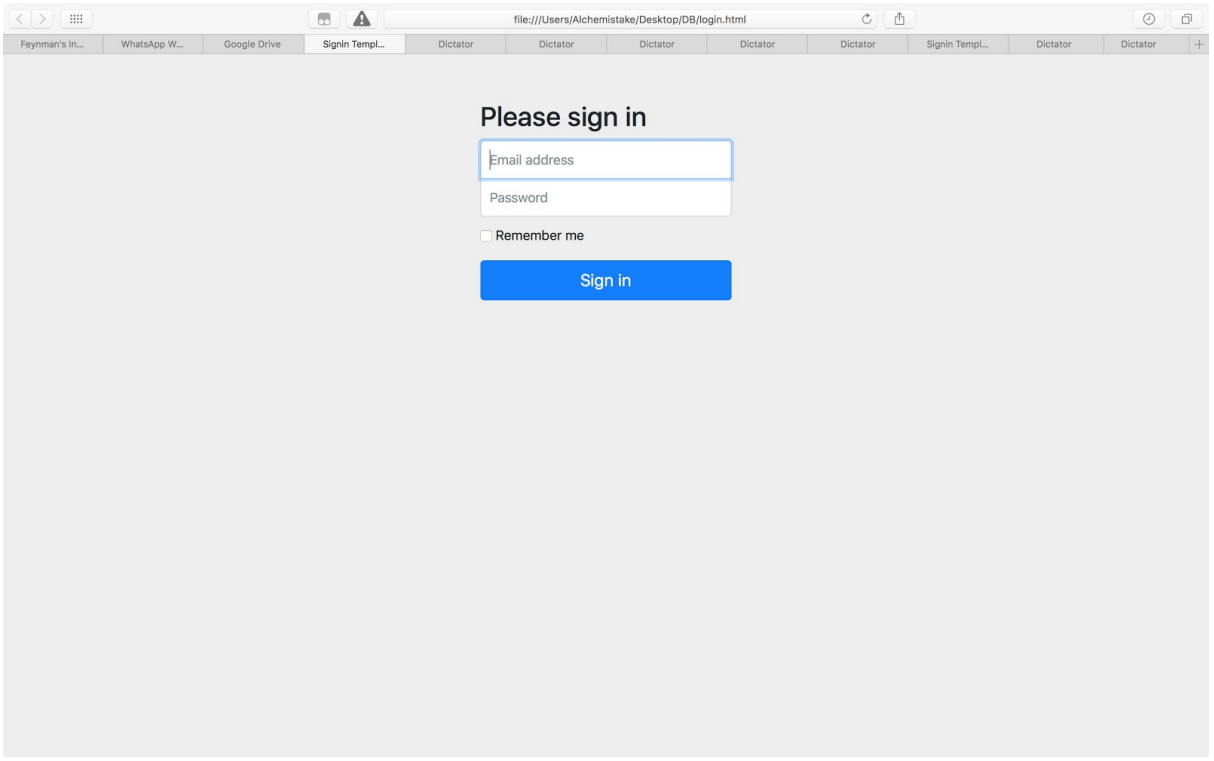
order by rate_value desc

limit 100

4.2. Signup/Login Navigation Bar

When a visitor is not logged in there will be a different navigation bar to indicate to them. This component does not use any SQL queries. It is detected by browser cookies.

4.3. Login Page



When a user wants to login they can click to the “Login” button from 4.2. Then they can enter their email address and password to login.

SQL Queries:

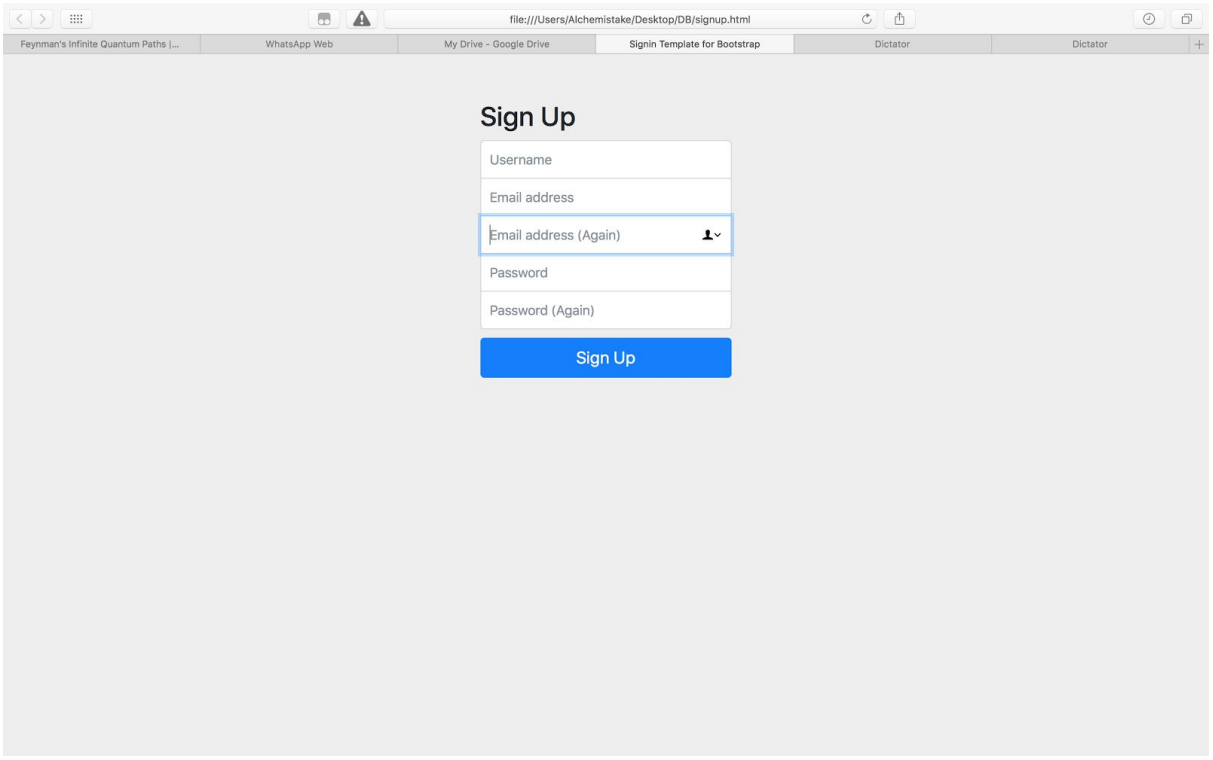
select *

from user

where user_email == *cur_email* and user_password == *cur_password*

cur_email, cur_password are passed from the Web Server component.

4.4. Signup Page



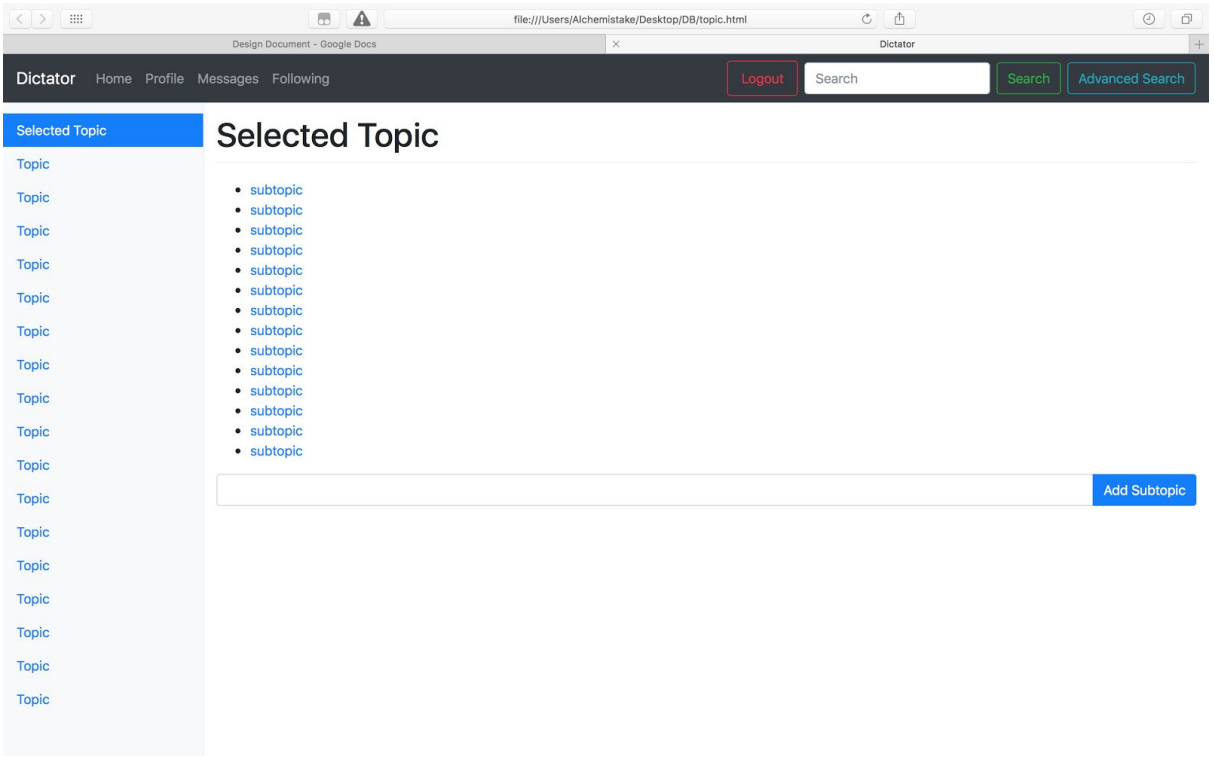
When a user wants to sign up they can click to the “Sign Up” button from 4.2. Then they can enter their email address, username and password to sign up for the site.

SQL Queries:

insert user(user_nickname, user_email, user_password, user_role) values (*cur_nickname, cur_email, cur_password*, 1)

cur_nickname, cur_email, cur_password are passed from the Web Server component.

4.5. Topic Page



When a topic is selected it shows all of the possible subtopics that can be explored. If the user has enough privileges to add subtopics, there will be a "Add Subtopic" part.

SQL Queries:

select *

from subtopic

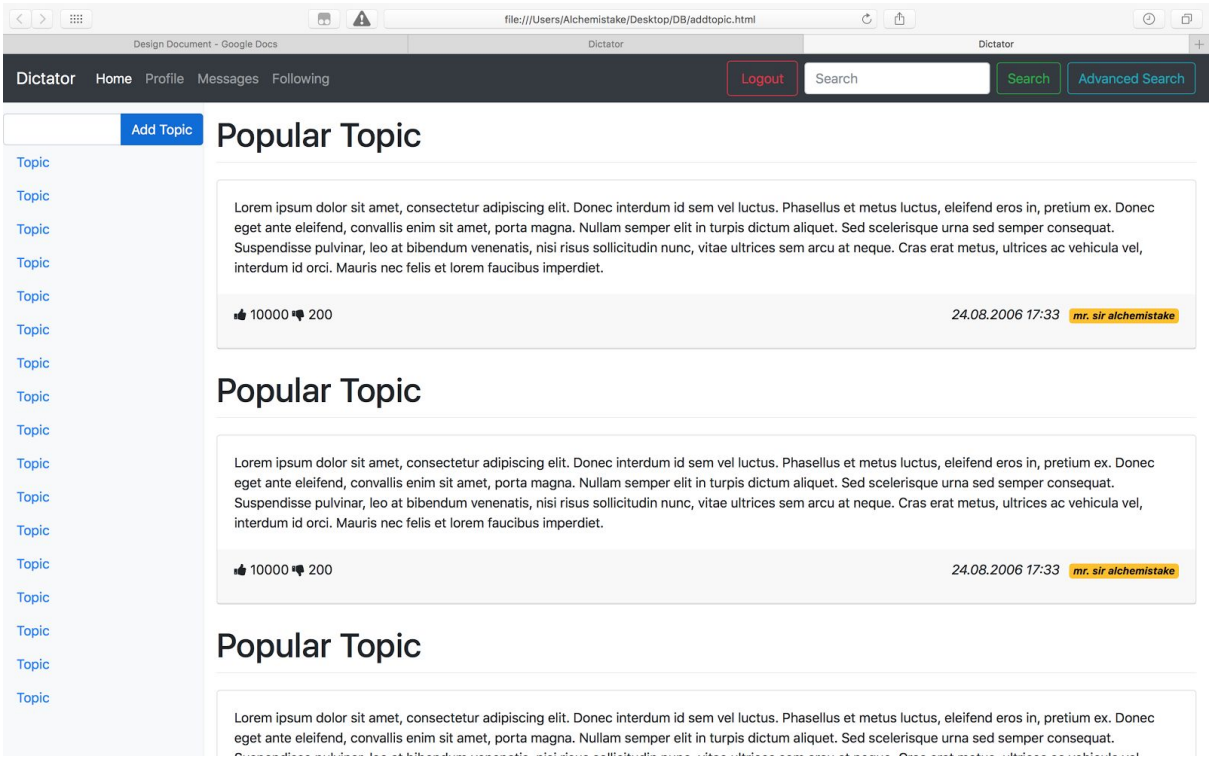
where topic_id == *cur_topic*;

cur_topic is passed from the Web Server component.

insert subtopic(topic_id, subtopic_name, adder_user) values (*cur_topic, requested_subtopic, cur_user*)

cur_topic, requested_subtopic, cur_user are passed from the Web Server component.

4.6. Adding Topic



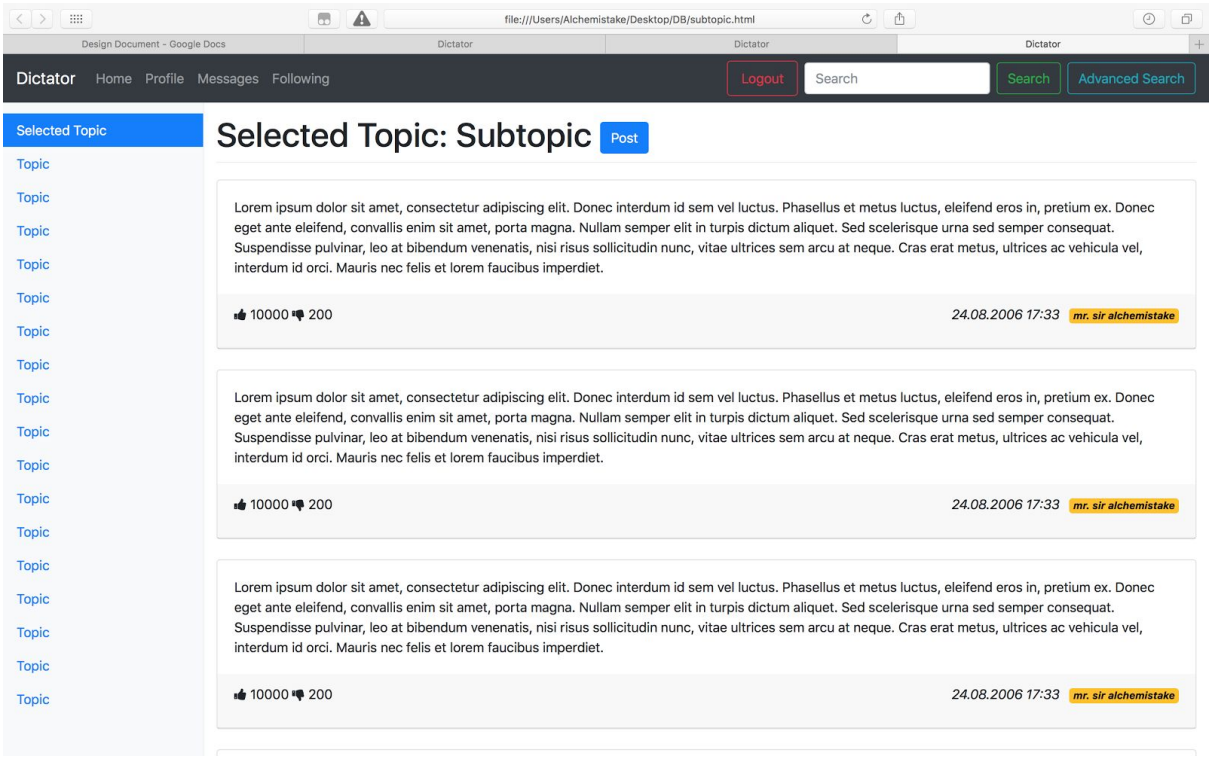
Like “Add Subtopic” part, if the user has enough privileges to add topics, there will be a textbox and button combo on the topics bar on the right.

SQL Queries:

insert topic(topic_name, topic_date, creator_user) values (*cur_topic_name, cur_date, cur_user*)

cur_topic_name, cur_date, cur_user are passed from the Web Server component.

4.7. Subtopic Page



After a user select any subtopic from 4.5 a page like following will be shown. If a user is privileged enough to post there will be a post button.

SQL Queries:

select *

from definition

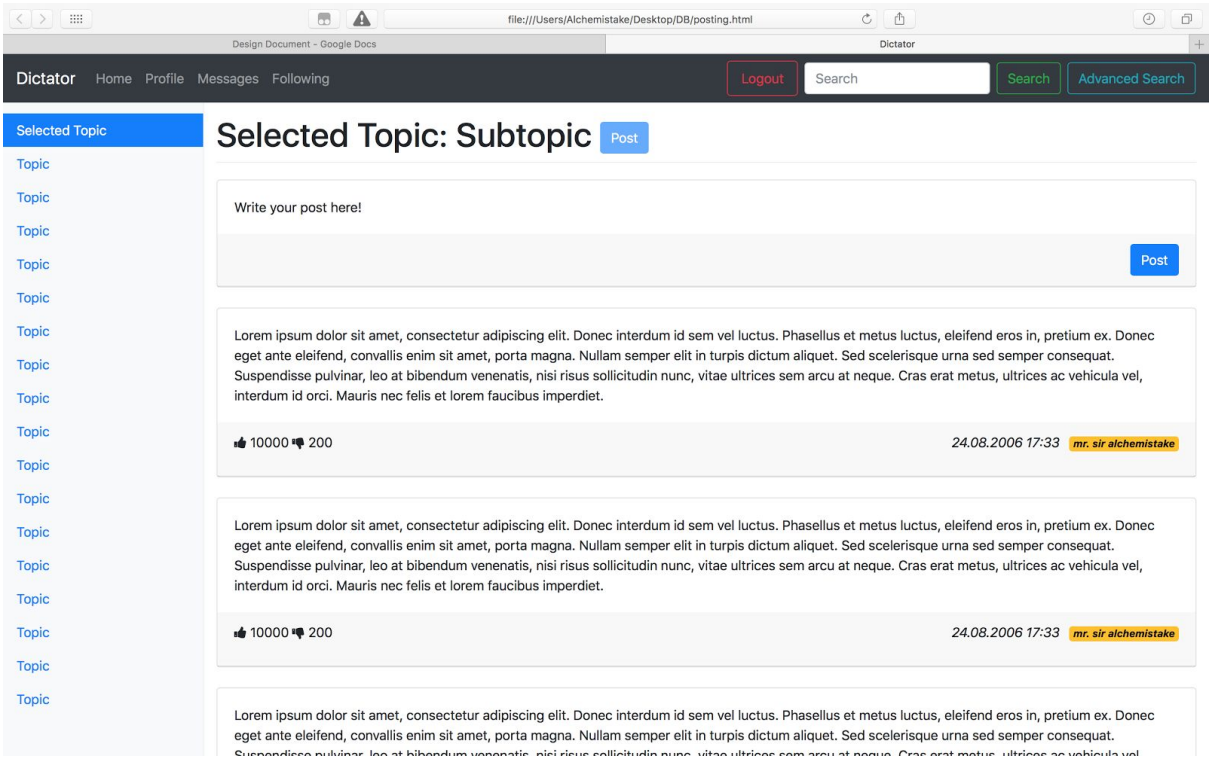
where definition_id in (select post_id

from Enter

where topic_id == *cur_topic* and subtopic_name == *cur_subtopic*)

cur_topic, cur_subtopic are passed from the Web Server component.

4.8. Posting



When user desires to post, they can click Post button from 4.7 to open a textbox to write. They can write their post and send by clicking the second post button. When this component is open first post button is disabled (both visually and functionally) to prevent confusion.

SQL Queries:

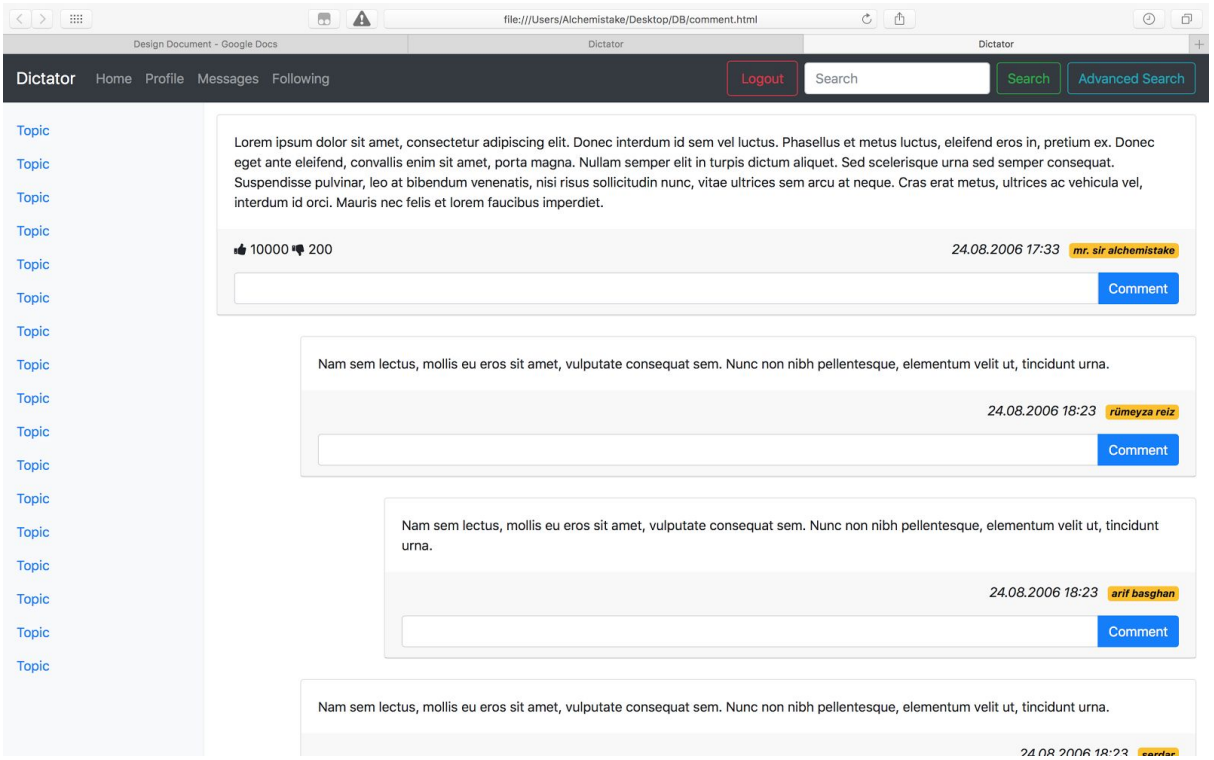
insert definition(definition_date, definition, definer_user) OUTPUT Definition.definition_id values (*cur_date, post_text, cur_user*)

insert post(post_id, rate_value) values (SCOPE_IDENTITY(), 0)

insert Enter(topic_id, subtopic_name, post_id) values (*cur_topic, cur_subtopic*, SCOPE_IDENTITY())

cur_date, post_text, cur_user, cur_topic, cur_subtopic are passed from the Web Server component.

4.9. Comments Page



If a user clicks on any post, there will be a page showing comments on it. User can comment on any of the comments on the page. There will be at most 3 levels of comments but if user desires they can click on any comment to make them main focus and show 3 more levels.

SQL Queries:

First Level Comment

```
select *
from Definition
where definition_id in (select comment_id
                        from Comment
                        where post_id == cur_post)
```

cur_post is passed from the Web Server component.

Second Level Comment

```
select *
from Definition
```

where definition_id in (select comment_id
from Reply
where replied_comment in(select comment_id
from Comment
where post_id == *cur_post*))

cur_post is passed from the Web Server component.

Third Level Comment

select *
from Definition
where definition_id in (select comment_id
from Reply
where replied_comment in(select comment_id
from Reply
where replied_comment in(
select comment_id
from Comment
where post_id == *cur_post*)))

cur_post is passed from the Web Server component.

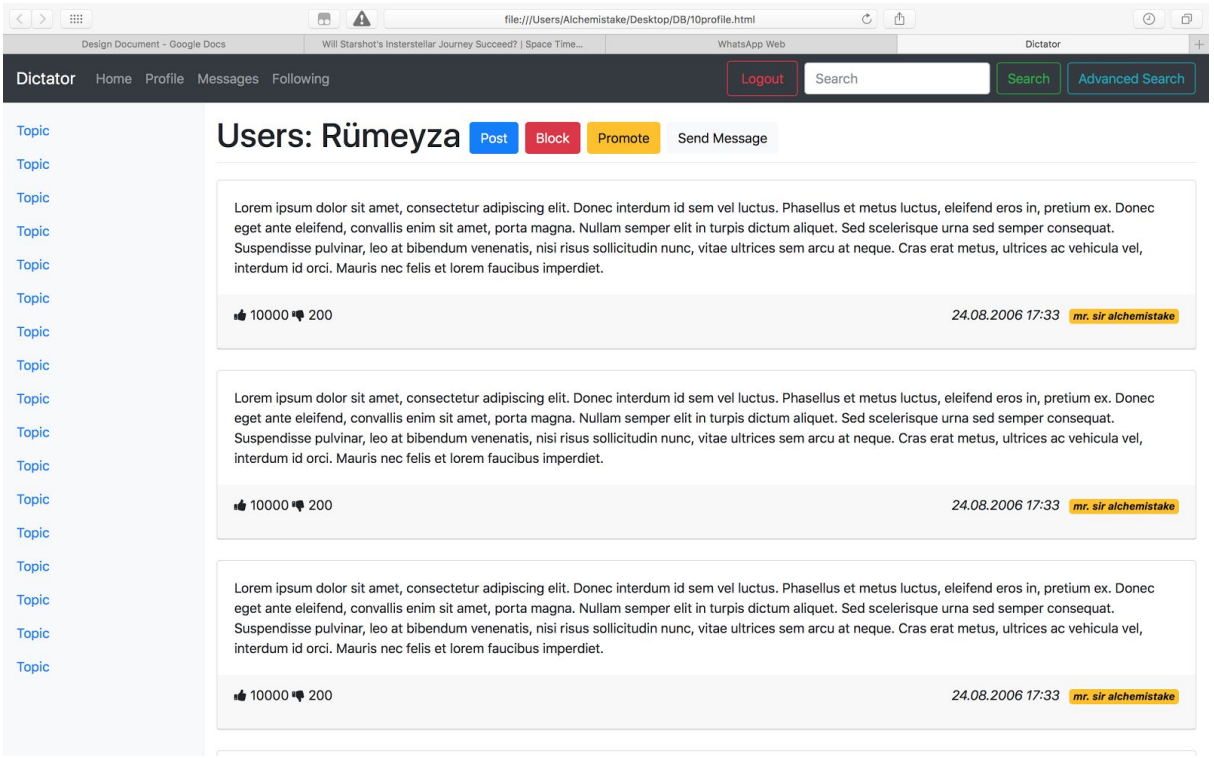
insert Definition(definition_date, definition, definer_user) values (*cur_date, comment, cur_user*)
insert Comment(comment_id, post_id) values (SCOPE_IDENTITY(), cur_post)

cur_date, comment, cur_user, cur_post are passed from the Web Server component.

insert Definition(definition_date, definition, definer_user) values (*cur_date, comment, cur_user*)
insert Reply(comment_id, replied_comment) values (SCOPE_IDENTITY(), cur_comment)

cur_date, comment, cur_user, cur_comment are passed from the Web Server component.

4.10. Profile Page



Profile page is like a subtopic page which is specialised by some other options. Users can post about another user here just like subtopics. Furthermore they can block, propose promotion, report or send message using this buttons. When “Send Message” is pressed it opens 4.14.

SQL Queries: Queries from subtopics are inherited here, I did not include them here.

insert block(blocker, blocked) values (*blocker_id*, *blocked_id*)

blocker_id and blocked_id are passed from URL arguments and passed by the Webserver element.

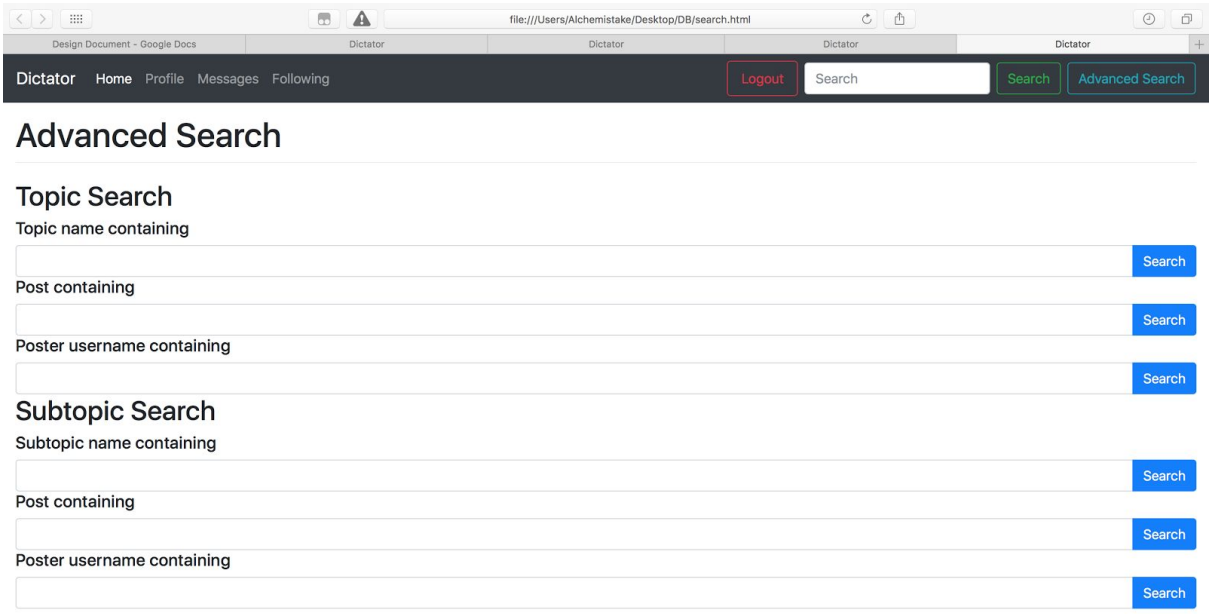
insert follow(follower, followed) values (*follower_id*, *followed_id*)

follower_id and followed_id are passed from URL arguments and passed by the Webserver element.

insert Propose(proposed, proposer) values (*proposed_id, proposer_id*)

proposed_id and proposer_id are passed from URL arguments and passed by the Webserver element.

4.11. Advanced Search Page



User can search using navigation bar or using Advanced search. The search bar acts as "topic name containing" option here. User can select any option and search according to that.

SQL Queries:

select *

from topic

where topic_name like '%*search*%'

search is passed from web server component.

```
select *
from topic
where topic_id in (select topic_id
                   from enter
                   where post_id in (select definition_id
                                     from definition
                                     where definition like '%search%'))
```

search is passed from web server component.

```
select *
from topic
where topic_id in (select topic_id
                    from enter
                    where post_id in (select definition_id
                                      from definition
                                      where definer_user like '%search%'))
```

search is passed from web server component.

```
select *
from subtopic
where subtopic_name like %search%
```

search is passed from web server component.

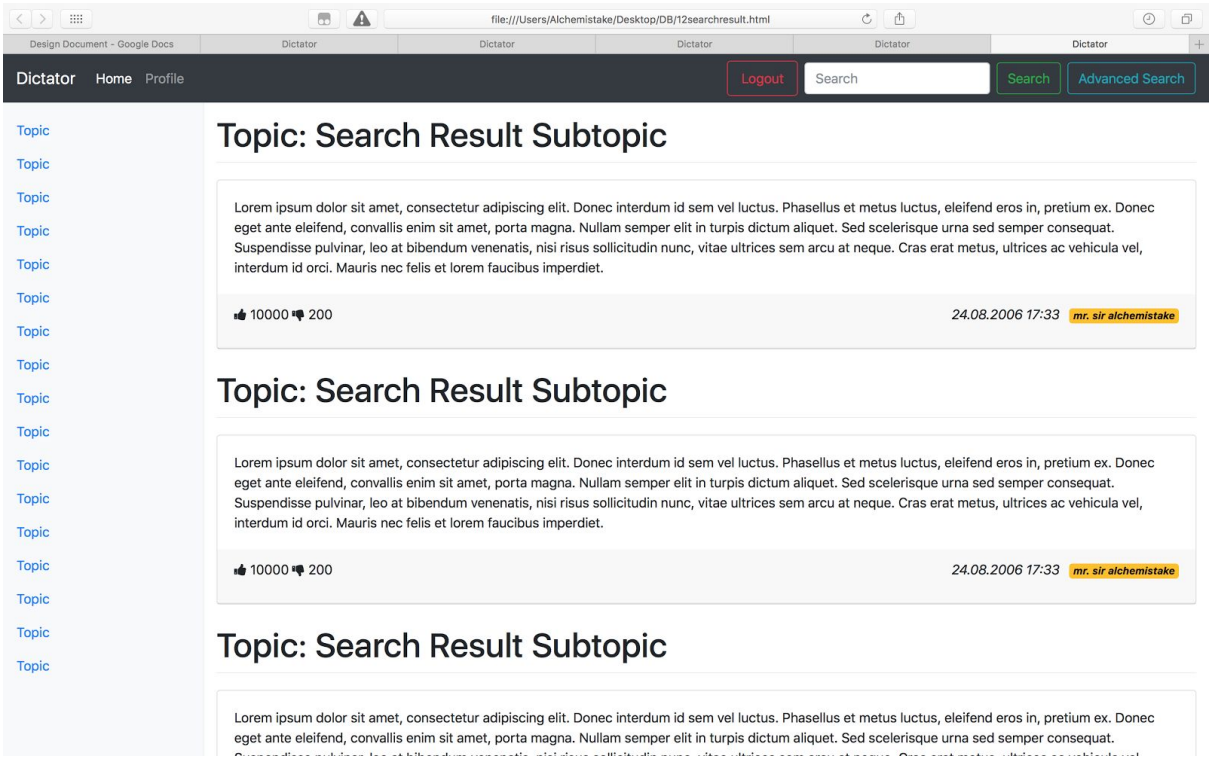
```
select *
from subtopic
where subtopic_name in (select subtopic_name
                        from enter
                        where post_id in (select definition_id
                                          from definition
                                          where definition like '%search%'))
```

search is passed from web server component.

```
select *
from subtopic
where subtopic_name in (select subtopic_name
                        from enter
                        where post_id in (select definition_id
                                          from definition
                                          where definer_user like '%search%'))
```

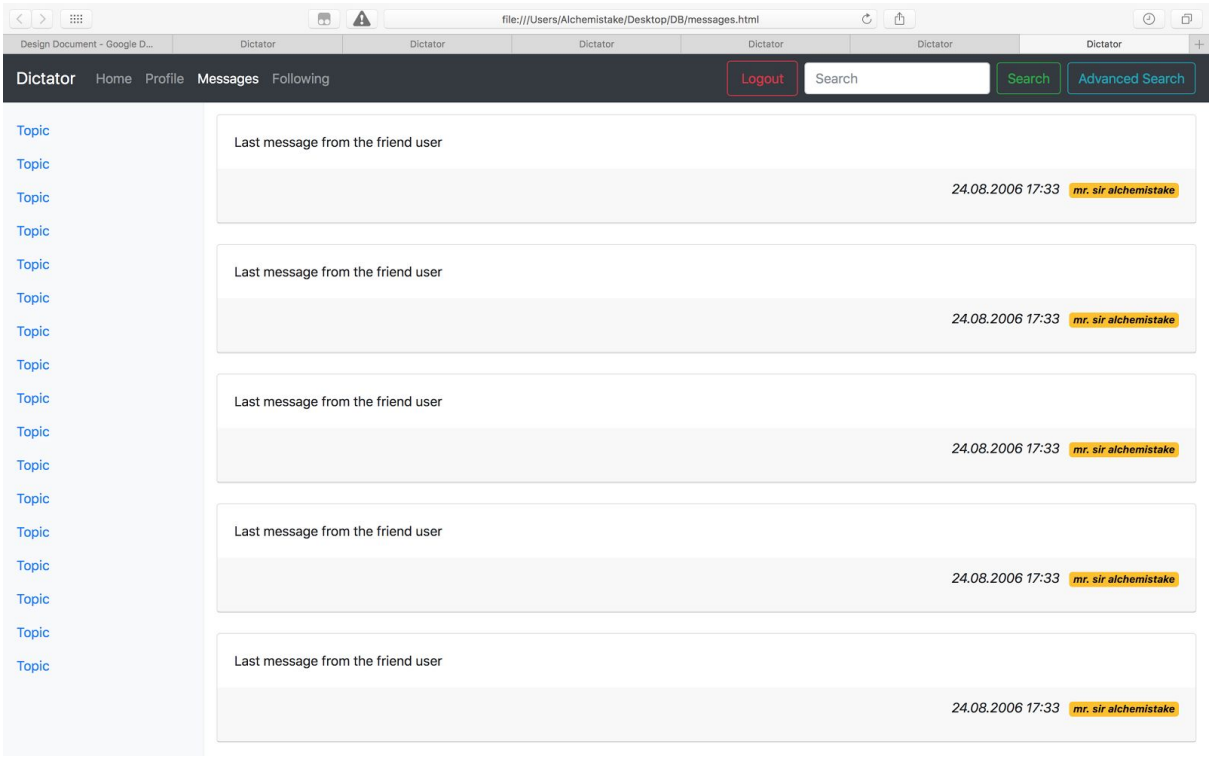
search is passed from web server component.

4.12. Search Result Page



When a user uses search there will a page like this shown containing posts that satisfies search criteria. This page technically does not contain any SQL queries since all the queries are captured at 4.11.

4.13. Messages



Messages page contains latest messages from different users. User can click to open the communication between them to send message or read older messages (4.14).

SQL Queries:

select *

from DirectMessage

where message_id in(select max(message_id), sender_user

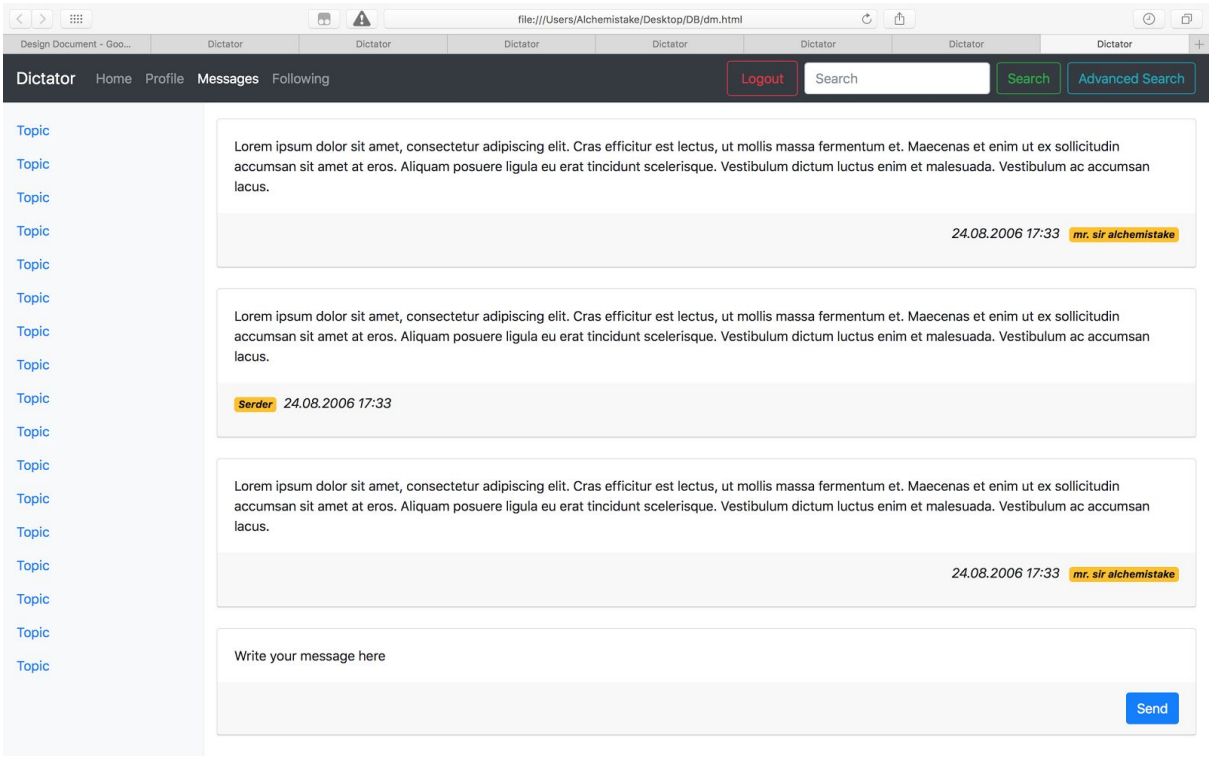
from DirectMessage

where receiver_user == *cur_user*

group by sender_user

cur_user is passed from web server component.

4.14. User to User Message Page



The messages between users are sorted in chronological order (using message_id). In the end of older messages there will be message send prompt.

SQL Queries:

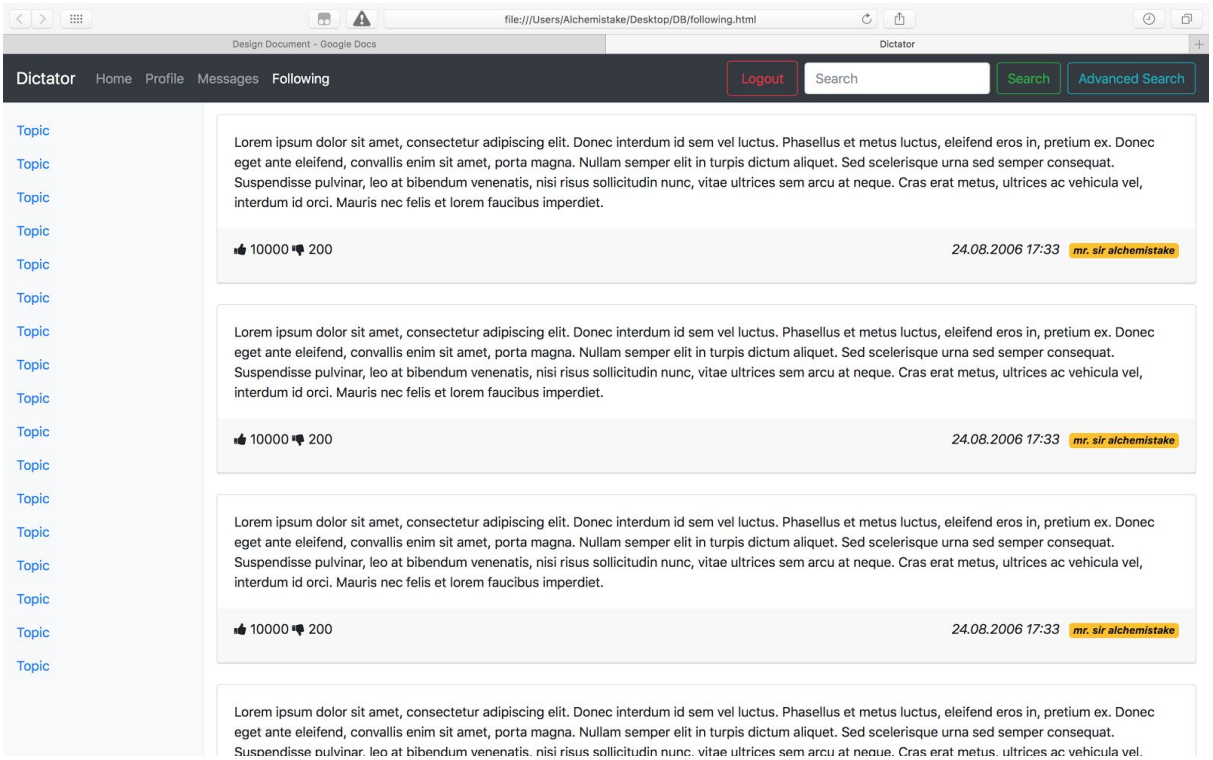
select *

from directMessage

where sender_user == *sender_user_id* and receiver_user == *receiver_user_id*

sender_user_id and receiver_user_id are passed from URL arguments and passed by the Webserver element

4.15. Following Page



Following page contains the posts from the followed users.

SQL Queries:

select *

from definition

where definer_user in (select followed

from follow

where follower == *cur_user*)

order by definition_id desc

cur_user is passed from web server component.

5. ADVANCED DATABASE COMPONENTS

5.1. Reports

5.1.1. Total Number of Users

Calculates the total number of users in Dictator Dictionary system.

```
select count(*) as total_user
from user
```

5.1.2. Total Number of Topics

Calculates the total number of topics created.

```
select count(*) total_topics
from topic
```

5.1.3. Total Number of Subtopics in each Topic

Calculates the total number of subtopics in each topic.

```
select  count(*)
from subtopic
group by topic_id
```

5.1.4. Total Number of Posts in each Subtopic

Calculates the total number of posts in each subtopic.

```
select  count(*)
from post natural join subtopic
group by subtopic_name, topic_id
```

5.1.5. Total Number of Comments for each Post

Calculates the total number of comments for each post.

```
select count(*) number_comment
from comment
group by post_id
```

5.1.6. Total Number of Replies for each Comment

Calculates the total number of replies for each comment.

```
select count(*) as number_reply
from reply
group by replied_comment
```

5.1.7. Total Number of Definitions for each User

Calculates the total number of definitions for each user.

```
select count(*) as number_def
from definition
group by definer_id
```

5.1.8. Total Number of Topics for each User

Calculates the total number of topics for each user.

```
select count(*) as number_topics
from topic
group by creator_user
```

5.1.9. Total Number of Messages for each User

Calculates the total number of messages for each user.

```
select count(*) as message_count
from directMessage
group by sender_user
```

5.2. Views

5.2.1. Profile View

Users will not be able to see other users' user_id and user_password. They can only see user_nickname, user_email and user_role.

```
create view profile (user_nickname, user_email, user_role) as
    select user_nickname, user_email, user_role
    from user
```

5.2.2. Topic View

Users will not be able to see topic_id. They can see topic_name, topic_date and creator_user.

```
create view topic_view (topic_name, topic_date, creator_user) as
    select topic_name, topic_date, creator_user
    from topic
```

5.2.3. Message View

User's can not see message_id while they are messaging. They can see the sender_user and the message itself.

create message_view(message, sender_user) as

select message, sender_user

from directMessage

5.2.4. Definiton View

Users can see definition_date, definition and definer_user. They can not see definiton_id.

create definition_view(definition, definition_date, definer_user) as

select definition, definition_date, definer_user

from definition

5.2.5. Search View

Users can see their search results but they can not see topic_id and definition_id

create search_view(topic_name, topic_date, definition, definition_date, definer_user) as

select topic_name, topic_date, definition, definition_date, definer_user

from topic natural join definition

where topic_name like '%top_*search*%' or definition like '%def_*search*% or

definer_user like '%definer_*search*%

top_search, def_search, definer_search are passed from web server component.

5.3. Triggers

5.3.1. Proposing a User

Users have a role value from 1 to 4:

1 -> Freshman

2-> Sophomore

3-> Junior

4-> Senior

When 100 users promote another user who has a role value less than themselves, promoted user's "role_value" increases.

```
create trigger user_role_update after insert on propose
referencing new row as new_row
when ( select count(proposed)                    =>100  and
        from proposed
        where proposed = new_row.proposed )
for each row
        begin
                update user
                set user_role = user_role +1
                where user_id = new_row.proposed

                delete from propose
                where proposed = new_row.proposed
        end
```

5.3.2. Blocking a User

Every time when 1000 users block a user, "role_value" of blocked user decreases "1" point until he/she becomes a freshman.

```
create trigger user_role_update after insert on block
referencing new row as new_row
when ( select count(blocked)              =>1000  and
        from block
        where blocked = new_row.blocked )
for each row
        begin
                update user
                set user_role = user_role -1
                where user_id = new_row.blocked

                delete from block
                where blocked = new_row.blocked
        end
```

5.3.3. Deleted User Trigger

5.3.3.1. Topic Trigger after Deleted User

When a tuple is deleted in the "User" table, the "created_user int(8) " attribute in the topics that she/he created becomes "00000000" which stands for *deleted user*.

```
create trigger topic_update before delete from user
referencing old row as old_row
for each row
        begin
                update topic
                set creator_user = 00000000
                where creator_user =old_row.user_id
        end
```

5.3.3.2. Subtopic Trigger after Deleted User

When a tuple is deleted in the "User" table, the "adder_user int(8) " attribute in the subtopics that she/he created becomes "00000000" which stands for *deleted user*.

```
create trigger subtopic_update after delete from user
referencing old row as old_row
for each row
        begin
                update subtopic
                set adder_user = 00000000
                where adder_user =old_row.user_id
        end
```

5.3.3.3. Definition Trigger after Deleted User

When a tuple is deleted in the "User" table, the "definer_user int(8) " attribute in the definitions that she/he created becomes "00000000" which stands for *deleted user*.

```
create trigger def_update after delete from user
referencing old row as old_row
for each row
        begin
                update topic
                set definer_user = 00000000
                where definer_user =old_row.user_id
        end
```

5.3.4. Reply Trigger after Deleted Comment

When a comment is deleted from "Comment" table, all replies of that comment are deleted from "Reply" table.

```
create trigger reply_update after delete from comment
referencing old row as old_row
for each row
        begin
                delete from reply
                where replied_comment =old_row.comment_id
        end
```

5.3.5. Rate Trigger

5.3.5.1. Rating a Post

After a "Post" entity is rated either +1 or -1 ("vote" attribute) by a "User" entity, "vote" will be added to "Post" entity's sum of rates and new "rate_value" will be generated.

```
create trigger rate_update after insert on rate
referencing new row as new_row
referencing old row as old_row
for each row
        begin
                update post
                set rate_value =old_row.rate_value + new_row.vote
                where post_id =old_row.post_id
        end
```

5.3.5.2. Rating after Deleted User

When a tuple is deleted in the "User" table, the votes that the user had will be deleted and "rate_value" for each post will be re-generated.

```
create trigger rate_delete after delete from delete
referencing new row as new_row
referencing old row as old_row
for each row
        begin
                update post
                set rate_value =old_row.rate_value - new_row.vote
                where post_id =old_row.post_id
        end
```

5.3.5.3. Example of Rate Trigger

Post A is rated by 10 users and "rate_value" is 5.

Case 1:

User A rates Post A with "vote" 1.

Post A: rate_value becomes -> (10*5 + 1)/11 = 4.6

Case 2:

User B was one of the 11 users who rated Post A with "vote" 3.

User B deleted.

Post A: rate_value becomes -> (11*4.6-3)/10 = 4.7

5.4. Constraints

5.4.1. Role Constraint

- User can not have a user role lower than Freshman(1) and greater than Senior(4).
- Users become Freshman when they first registered to the system.
- A Freshman can not be promoted down.
- A Senior can not be promoted up.

5.4.2. Topic Constraint

- A topic can not have more than 1 creator user.
- A topic's creator_user can not be null but 00000000 which stands for deleted user.
- A subtopic can not have more than 1 creator user.
- A subtopic's creator_user can not be null but 00000000 which stands for deleted user.
- A topic can not be sent without a topic_name.

5.4.3. Definition Constraint

- A definition can not have more than 1 creator user.
- A definition's definer_user can not be null but 00000000 which stands for deleted user.
- A definition can not be defined without a context.

5.4.4. Rate Constraint

- A post's rate_value can not be anything other than +1 or -1.

5.4.5. Message Constraint

- A user can not send message herself/himself.
- A message can not be sent empty.
- A user can not send a message to a user which blocks herself/himself.

5.4.6. Subtopic Constraint

- A user can not add a subtopic without assigning it to a particular topic.

5.4.7. Post Constraint

- A user can not enter a post without assigning it to a particular subtopic.

5.4.8. Follow Constraint

- A user can not follow herself/himself.
- A user can not followed by a user who blocked herself/himself.

5.4.9. Propose Constraint

- A user can not propose herself/himself to promote.
- If user A blocks user B, user B can not be proposed by user A unless user A unlocks user B.
- If user A blocks user B, user A can not propose user B.
- A user can not be proposed by a user who has a lower role than herself/himself.

5.4.10. Block Constraint

- A user can not block herself/himself.
- If a user A is blocked by user B, user A can not block user B.

5.5. Stored Procedures

Stored procedures will be used when creating a new topic, post, comment or reply. A new tuple will be inserted into the Dictator dictionary system with the given values when a new instance is created.

5.5.1. Topic Created Stored Procedure

Whenever a topic is created, after the topic name is indicated. Then according to that subtopics page is created. This procedure is repeated whenever a topic is created.

5.5.2. Subtopic Created Stored Procedure

Whenever a subtopic is created, after the topic id and subtopic name are indicated. Then according to that subtopic page is created. This procedure is repeated whenever a subtopic is created.

5.5.3. Post Created Stored Procedure

Whenever a post is created, after the topic id, subtopic name, post contents are indicated. Then according to that subtopic page is updated and comments page is created . This procedure is repeated whenever a post is created.

5.5.4. Comment Created Stored Procedure

Whenever a comment is created, after the topic id, subtopic name, post id or reply id are indicated. Then according to that comments page is updated. This procedure is repeated whenever a comment is created.

6. Implementation Plan

We are planning to use Python and Flask framework for basic web server implementation. If needed, Apache2 Virtual Hosting would be added for stability. We are going to use PostgreSQL for our database. Classical HTML, CSS, JS will used as well. But they will be compiled from Jinja 2 Templates, HMTL generalization standard used by Flask framework. All will be run on a remote server.