University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 4

Date: November 29th, 2024

Group Number: 25

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Jeff Kim	70668132	m3v1i	Jeffkim7@hotmail.com
Jessica Patricia	81731218	l7j4y	jessicapatricia012@gmail.com
Hansel Poe	82673492	l7z7n	hpoe01@student.ubc.ca

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

GITHUB REPOSITORY LINK

https://github.students.cs.ubc.ca/CPSC304-2024W-T1/project_l7j4y_l7z7n_m3v1i.git

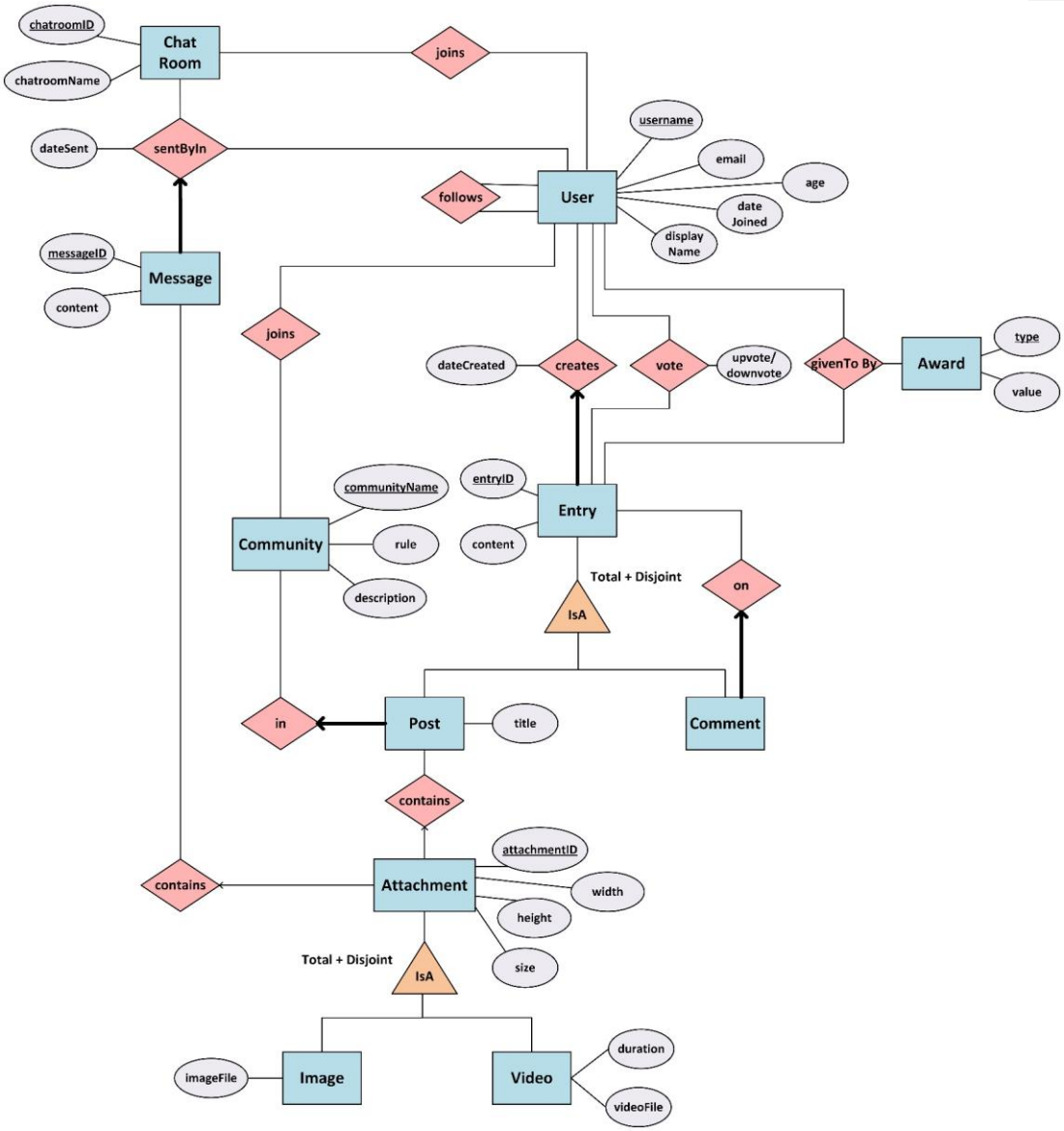
SUMMARY

We are making a database for a social media platform inspired by the popular application Reddit with similar features, but with our own tweaks. Our domain will model basic social networking concepts, such as users, communities, posts, comments, messages, attachments, chat rooms, and awards. The user interactions are modeled in the relationships between entities where users can join communities, follow other users, make posts and comments with attachments, give awards to and upvote/downvote posts and comments, and enter chat to message other users.

PROJECT DESCRIPTION

Our project is a simple social media app that mimics the user experience of a social media platform. These are the list of the features from our app:

- Create an account and post (INSERT)
- Delete an account(DELETE)
- Update User account (UPDATE)
- Search for awards (SELECTION)
- Filter information of an account (PROJECTION)
- See Posts and Comments from a user (JOIN)
- See Number of posts from a user (AGGREGRATION WITH GROUPBY)
- Search for users who has given at least 5 awards (AGGREGRATION WITH HAVING)
- See the most popular communitites (NESTED AGGREGRATION WITH GROUPBY)
- Search users who have joines all communities (DIVISION)


chatroomID
Chat Room
chatroomName
joins
dateSent
sentByIn
username
email
age
follows
User
date Joined
display Name
messageID
Message
content
joins
dateCreated
creates
vote
upvote/ downvote
givenTo By
Award
type
value
communityName
entryID
Entry
Community
rule
content
description
Total + Disjoint
IsA
on
in
Post
title
Comment
contains
attachmentID
contains
Attachment
width
height
size
Total + Disjoint
IsA
imageFile
Image
Video
duration
videoFile

SCHEMA CHANGES

- Added attribute age to User that indicates the age of the account (not user), just like how Reddit has cake day (dateJoined) and Reddit age. This change was made so that dateJoined->age and we could normalize User. Previously, we had ImageContainedBy and VideoContainedBy normalized, but there are some concerns regarding it (https://piazza.com/class/m01ini8hsm72gv/post/569)
- Removed the total participation constraint on ChatRoom to joins, Award to givenToBy, and Community to joins for simplification. Our diagram still adhere to the requirements and make sense (in some cases, more sense): we still keep the data on a chatroom even if no user are in it, an Award entitiy should still exist even if no user has given/received it (we still keep the data on the type and value associated with it), an empty community can exist and we should retain its data (a user can join that community anytime).

UPDATED SCHEMA

- User (username: varchar(20), email: varchar(20), dateJoined: date, displayName: varchar(20), age: integer)
  Primary Key: username
  Candidate Key: email (one email can only be associated with one account)
  Unique: email
  Not NULL: dateJoined, email

UPDATED FUNTIONAL DEPENDENCIES

From User:

- username → email, dateJoined, displayName, age
- email → username, dateJoined, displayName, age
- dateJoined → age

From ImageContainedBy:

- ~~imageFile → width, height, size~~
- attachmentID → imageFile, width, height, size, entryID, messageID

From VideoContainedBy:

- ~~videoFile → width, height, size, duration~~
- attachmentID → videoFile, width, height, size, duration, entryID, messageID

Commented [JP1]: I made 1 schema per entity/relationship but we could maybeee merge some though I don't understand how. There are prob more constraints we need to add too (unique, not null, foreign key) but idk I could only figure out some

Commented [JP2R1]: And for schemas involving relationships with total participation, idk how we're supposed to show that constraint but I made it according to this post https://piazza.com/class/m01ini8hsm72gv/post/187

Commented [JP3]: Added this (?)

Commented [HP4R3]: Idk FDs should kinda be between attributes in one table, So I'd say this is wrong

Commented [JP5R3]: Wdym image has attachmentID and imageFile

Commented [JP6R3]: The one that is wrong if you said so is imageFile->resolution, size bcs res and size is not on image

UPDATED NORMALIZATION (to 3NF)

Instead of normalizing ImageContainedBy and VideoContainedBy, we normalize Users:

Closures:

- username += email, dateJoined, displayName, age
- email += username, dateJoined, displayName, age
- dateJoined += age

username is key

Normalize to 3NF by Loseless Join Method:

dateJoined → age is not in 3NF because dateJoined is not a superkey and age is not part of superkey.

Minimal cover:

- username → email
- ~~username → dateJoined~~
- ~~username → displayName~~
- ~~username → age~~
- email → username
- email → dateJoined
- email → displayName
- ~~email → age~~
- dateJoined → age

dateJoined → age violates 3NF because dateJoined is not a superkey and age is not part of superkey.

Decomper User on dateJoined → age:

- UsersAge(<u>dateJoined</u>, age)
- Users (<u>username</u>, email, dateJoined, displayName)

Result:

- UsersAge(<u>dateJoined</u>, age)
- User (<u>username</u>, email, **dateJoined**, displayName)

*Primary keys are <u>underlined</u>. Foreign keys are **bolded**. Candidate key for User: email.*

QUERIES

2.1.1 Insert

Create Post:

```
INSERT INTO ENTRYCREATEDBY(dateCreated, content, username) VALUES
(:myDate, :myContent, :myUser) RETURN entryID INTO :myId;
INSERT INTO POSTIN(entryID, title,communityName) VALUES
(:myId, :myTitle, :myCommunity);
```

appService.js: 179-188, 194-198

2.1.2 Update

Update Attributes of User (expect dateJoined because it refers to the date an account was created and it shouldn't be updatable: https://piazza.com/class/m01ini8hsm72gv/post/637_f1):

```
UPDATE USERS
SET email = :email,
    displayName = :displayName,
    dateJoined = TO_DATE(:dateJoined, 'YYYY-MM-DD')
WHERE username = :username
```

appService.js: 265-273

2.1.3 Delete

Delete User:

```
DELETE FROM USERS WHERE username = :username
```

appService.js: 213-217

2.1.4 Selection

Search for awards:

```
SELECT * FROM AWARD
WHERE UPPER(${clause.attribute}) = UPPER(:${clause.attribute}_${index}) ${clause.operator} ...
```

(where queryparams['${clause.attribute}_${index}'] = clause.value)

appService.js: 289-312

2.1.5 Projection

Shows selected attributes of entity

```
SELECT ${columns.join(', ')} FROM ${tableName}
```

scripts.js:

2.1.6 Join

See Posts and Comments from a user

```
SELECT ${selectedAttributes}
FROM Users
JOIN EntryCreatedBy
ON Users.username = EntryCreatedBy.username
WHERE Users.username = '${selectedUsername}
```

scripts.js: 520-535

2.1.7 Aggregation with GROUP BY

See number of posts from each user (if there is any):

```
SELECT u.username, COUNT(*)
FROM Users u, EntryCreatedBy e, PostIn p
WHERE e.username = u.username and e.entryid = p.entryid
GROUP BY u.username
```

appService.js: 230-233

This query fetches all posts from users by natural joining USERS, ENTRYCREATEDBY, and POSTIN, then groups the tuples by username and caculates the count for each group.

2.1.8 Aggregation with HAVING

See users who have given at least 5 awards:

```
SELECT username, COUNT(*)
FROM GivenToBy
GROUP BY username
HAVING COUNT(*) >= 5
```

appService.js: 324-331

Groups the tuples from GIVENTOBY by username and keep only groups with at least 5 awards

2.1.9 Nested aggregation with GROUP BY

See the most popular community (will show multiples if there's a tie):

```
SELECT communityName, COUNT(*)
FROM joinsCommunity
GROUP BY communityname
HAVING count (*) >= ALL(SELECT COUNT(*)
                        FROM joinsCommunity
                        GROUP BY communityName)
```

appService.js: 245-250

Groups tuples in joinsCommunity by communityName and keep only groups (communites) whose count (number of members) are greater than or equal to count of all other groups (communities).

2.1.10 Division

See Users who have joined all communities:

```
SELECT username, displayName
FROM Users u
WHERE NOT EXISTS
        ((SELECT C.communityname
        FROM Communities C)
        MINUS
        (SELECT j.communityName
        FROM JoinsCommunity j
        WHERE j.username = u.username))
```

appService: 341-353

The input to NOT EXISTS clause selects all communities that are not joined by user u. If no such community exists, then NOT EXISTS evaluates to true. The query then filters tuples from users where this clause is true.