Milestone 1: Project Proposal

CSC 261 DATABASE SYSTEMS, Spring 2019
University of Rochester

Project Details

Project Name: Stack Overflow Website Database
Team ID: 7
Team Member 1: Zechen Wang
Team Member 1 ID: zwang88
Team Member 2: Ziyu Song
Team Member 2 ID: zsong10

Problem Statement

This project will be mainly addressing an archive of Stack Overflow content, including users, tags, votes, comments, and posts.

In this project, we will design a database system that will be used to address the need in solving the problem described above. There are several reasons why we choose to use a database instead of an excel file: 1.With a database, multiple access is allowed. Database also allows different levels of access to users. 2. In the database, all data are sorted and a database could save time when copying and pasting data entries. 3. Database can help with improving the data quality and consistency. 4. Effective version control. Spreadsheet usually doesn't have version control and people who use that may get confused by various versions. 5. More robust security. In certain situations, there is a need to keep certain data restricted, whilst other data can be readily shared. These sorts of privileges can be tricky to maintain – especially when there are several hundred documents. In a database, every user has their own login credentials meaning that it is far easier to manage access rights to information.

Target User

The target user of our database is Stack Overflow website users and website administrator. The database is built for website development purposes.

Users of the website can enter their user_id, post_id, tag…and get detailed information about their posts. The administrator can record votes, comments… Users of Stack Overflow will be able to see the popularity of questions and their activities on this website. Administrators can see what types of questions are most popular and they can administer and operate the website/forum accordingly.

List of Relations

answer
answer_id (int)
answer_body (char)
date (char)
owner_id (int)
parent_id (int)

post
post_id (int)
title (char)
answer_count (int)
accepted_answer_id (int)
creation_date (char)
comment count (int)
owner_id (int)
owner_name (char)
view_count (int)

vote
vote_id (int)
date (char)
post_id (int)
vote_type_id (int)

user
user_id (int)
display_name (char)
creating_date (char)
last_access (char)
up_votes (int)
down_votes (int)

comment
coment_id (int)
date (char)
post_id (int)
user_id (int)

Web-Interface

The web interface, which will be implemented with PHP, will be designed to take input from users for both updates and queries. The interface will allow these behaviors:

1. Accept new tuple input(s) if it is valid;
2. Update tuples in a specified relation;
3. Reject invalid input(s);
4. Query for tuples with a specified relation;
5. Query for tuples that satisfy certain attribute values;
6. Delete tuples upon request;
7. Register user privilege as “read_only” or “read_and_write”.

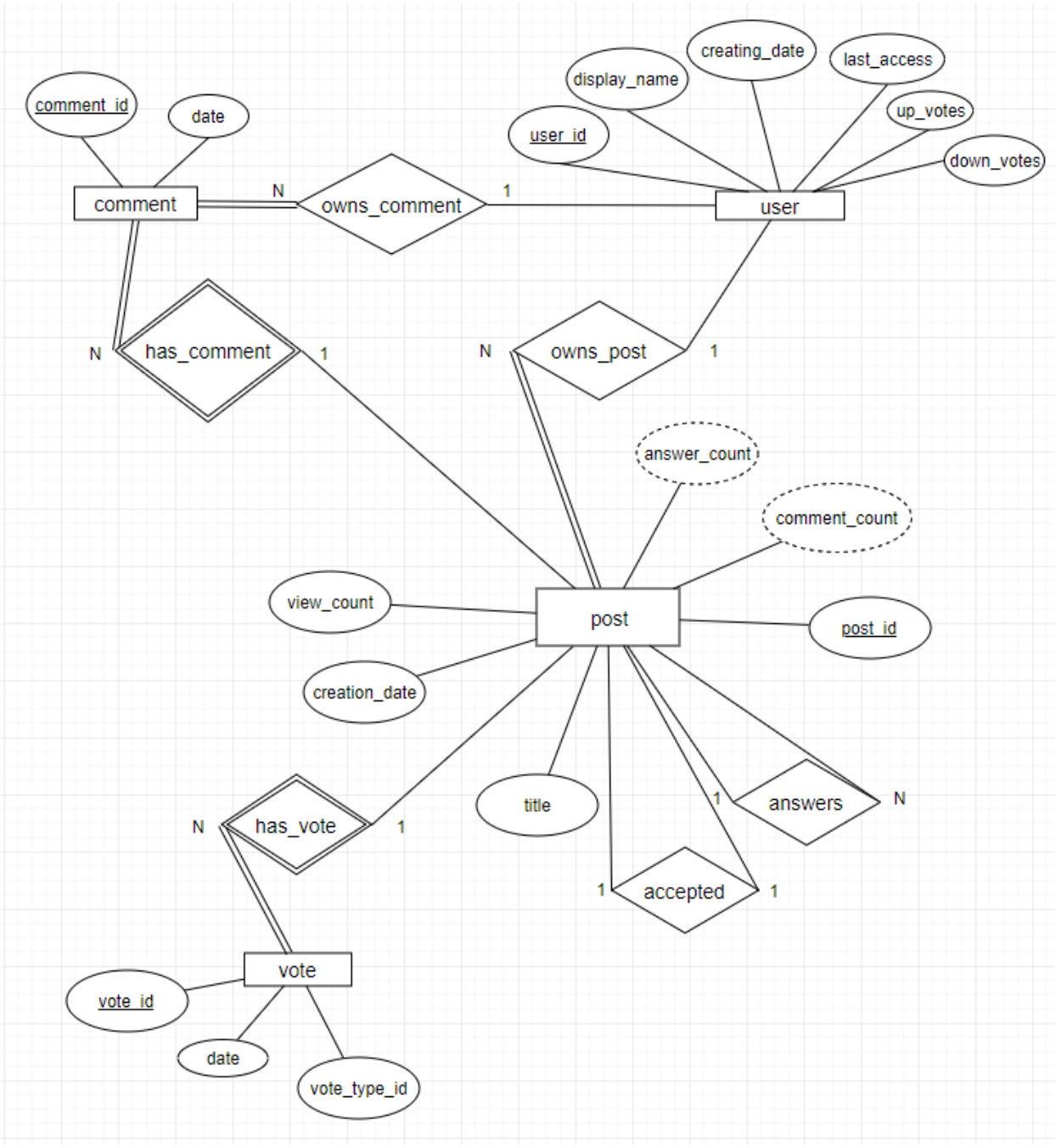
Data

BigQuery Dataset on Stack Overflow Data, Kaggle.com. [Accessed on 2019/02/07].
Url: https://www.kaggle.com/stackoverflow/stackoverflow

Milestone 2
CSC 261 Database Systems, Spring 2019

Team ID: 7
Zechen Wang (zwang88)
Ziyu Song (zsong10)

Task A: Draw an ER diagram



Assumptions:
Every user can own one or more posts. Each post is posted by only one user.
Every user can own one or more comments. Each comment is posted by only one user.
Each post can have one or more comments. Each comment is only aimed at one post.
An answer is a post. Each answer can have one more vote. Many votes can be made for one answer.
Answers can be accepted by a post.
A post can have no owner.

Task B: Relational Database Design Using ER-to-Relational Mapping

1. ER-to-Relational Mapping Algorithm
Step 1: Mapping of Regular Entity Types
Since there is no weak entity in our ER-schema, for each entity E, create a relation R.

Entity E	Relation R
post	post
vote	vote
comment	comment
user	user

Step 2: Mapping of Weak Entity Types
This step is not applicable in this case. There is no weak entity in the schema.

Step 3: Mapping of Binary 1:1 Relationship Types
For this step, we use the foreign key approach for each 1:1 relationship. In this schema, the only binary 1:1 relationship is ACCEPTED.

Step 4: Mapping of Binary 1:N Relationship Types
The foreign key approach is applied to the schema in this step. Add foreign key PARENT_ID which references POST_ID to POST.

Step 5: Mapping of Binary M:N Relationship Types
This step is not applicable in this schema. There is no M:N relationship.

Step 6: Mapping of Multivalued Attributes
This step is not applicable in this schema. There is no multivalued attribute.

Step 7: Mapping of N-ary Relationship Types
This step is not applicable in this schema. All relationships are binary.

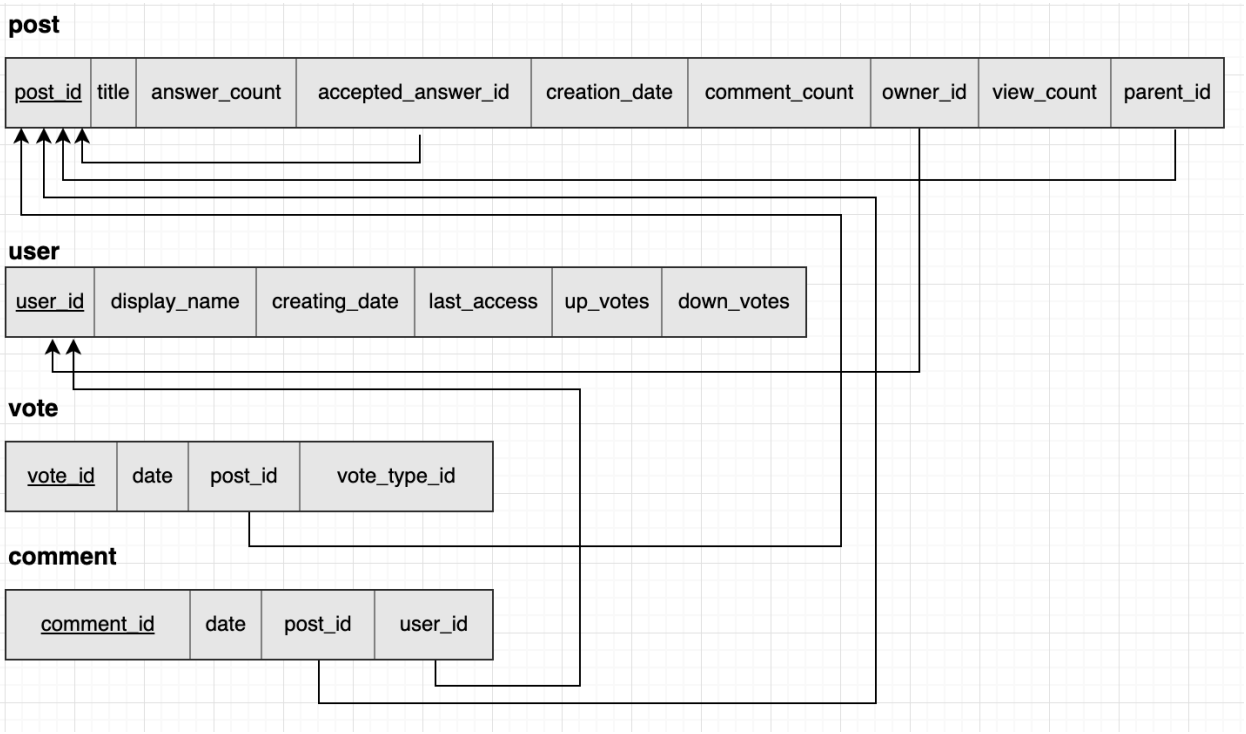
Step 8: Options for Mapping Specialization or Generalization

There is one originally proposed entity ANSWER which is a subclass of POST. When we construct the ER schema, this entity is generalized into a single POST entity. So this step is skipped here.

Step 9: Mapping of Union Types (Categories)

This step is not applicable in this schema. There is no surrogate key needed for this schema. The attributes in most relationships are POST_ID and USER_ID. However, they are already keys that we can use directly in these relationships.

Mapping Summary	
Relation Name	**ER Diagram Components**
post	E(post)+R(owns_post)+R(has_comment)+R(accepted)+R(has_vote)+R(answers)
user	E(user)+R(owns_post)+R(owns_comment)
vote	E(vote)+R(has_vote)
comment	E(comment)+R(owns_comment)+R(has_comment)



2. Schema of the Database

POST

Attribute	Data Type	Default Value	Accept Null	Purpose	Description
post_id (Primary Key)	int	current max + 1		Primary key that identifies each post.	A positive integer that identifies a post. A later post should have a larger ID.
title	char	NULL	TRUE	Record the title of the post	The title of the post.
answer_count	int	0		How many answers are posted under a post	A derived attribute that reports the number of answers to this post.
accepted_answer_id (Foreign Key)	int	NULL	TRUE	Record the answer accepted	The post id of the accepted answer.
creation_date	datetime	NULL	TRUE	Record the post date	timestamp, post date
comment_count	int	0		Record the comments posted for each post	A derived attribute that reports the number of comments to this post.
owner_id (Foreign Key)	int	NULL	TRUE	The user id	User ID of the post owner.
view_count	int	0		Record the views of a post	An integer that reports the number of views of this post.
parent_id (Foreign Key)	int			Link to the parent post	The ID of the parent post.
post_body	text		TRUE		

CUSTOMER

Attribute	Data Type	Default Value	Accept Null	Purpose	Description
user_id (Primary Key)	int	current max +1		Primary key that indicate each user	A positive integer that identifies a user. A later registered user should have a larger ID.
display_name	char	NULL	TRUE	Show the name of each user	The display name of the user.

creating_date	char	NULL	TRUE	Record the creation date of the account.	The registration date of this user.
last_access	datetime	creating_date	TRUE	Record the last access of the user	The last time the user logs on.
up_votes	int	0	TRUE	Record the up votes user makes	The number of up votes.
down_votes	int	0	TRUE	Record the down votes user makes	The number of downvotes.

VOTE

Attribute	Data Type	Default Value	Accept Null	Purpose	Description
vote_id (Primary Key)	int			Primary key that indicates each vote	A positive integer that identifies a vote. A later vote should have a larger ID.
vote_date	char	NULL	TRUE	Record the date when the vote is made. Timestamp	The time of the vote.
post_id (Foreign Key)	int			The corresponding post the vote is made	User ID of the post the vote associates to.
vote_type_id	int	NULL	TRUE	The ID of the vote type	A code that describes the vote type.

USER_COMMENT

Attribute	Data Type	Default Value	Accept Null	Purpose	Description
comment_id (Primary Key)	int			Primary key that indicates each comment	A positive integer that identifies a comment. A later post should have a larger ID.
comment_date	char	NULL	TRUE	Record the date when the comment is posted	Timestamp
post_id (Foreign Key)	int			The corresponding post of where the comment is posted	The ID of the parent post.

user_id (Foreign Key)	int			User who posts this comment	User ID of the owner of the comment.

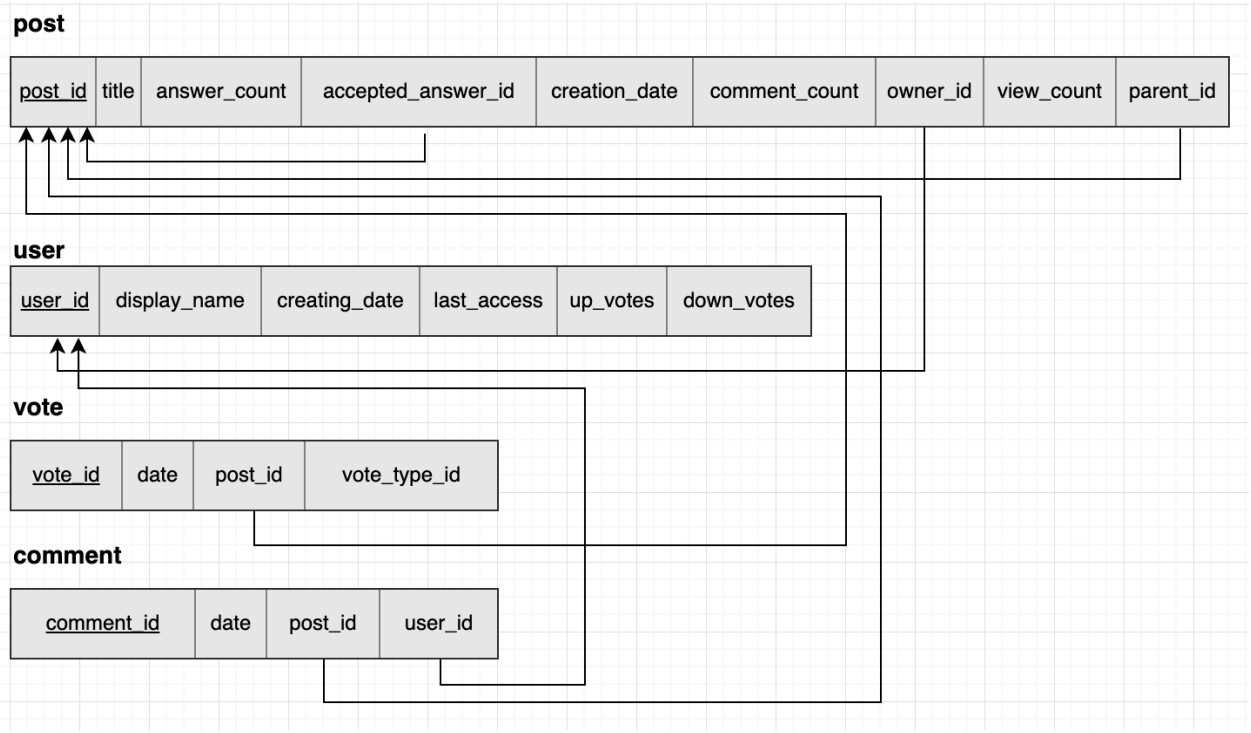
All the primary keys are unique keys.

For VOTE and COMMENT relation, when the corresponding tuple for the referred table is deleted, the action on the foreign key is DELETE CASCADE.

For POST relation, the action taken depends on the foreign key attribute. If the tuple which OWNER_ID refers to is deleted, use the SET NULL option. For all other foreign key attributes, use the DELETE CASCADE option.

Task A: BCNF Normalization

Initial Schema



The initial relation schema is already in Boyce-Codd Normal Form (BCNF). For each non-trivial dependency X→Y in relation R, X is a key, or a minimal superkey, for R. Therefore, all relations in the initial schema are in BCNF. Here are the lists of non-trivial functional dependencies (FD) for each relation.

post_id → accepted_answer_id
post_id → parent_id
post_id → title
post_id → answer_couunt
post_id → certain_data
post_id → comment_count
post_id → owner_id
post_id → view_count

user_id → display_name
user_id → creating_date
user_id → last_access
user_id → up_votes
user_id → down_votes
user_id → {user_id, display_name, creating_date, last_access, up_votes, down_votes}
user.user_id → comment.owner_id

user.user_id → comment.user_id

vote_id → date
vote_id → post_id
vote_id → vote_type_id
vote_id → {vote_id, date, post_id, vote_type_id}

comment_id → date
comment_id → post_id
comment_id → user_id
comment_id → {comment_id, date, post_id, user_id}

Foreign Key Relations
post.post_id → comment.post_id
post.post_id → vote.post_id

Project 1 Milestone 4 Report

CSC 261 DATABASE SYSTEMS, Spring 2019

University of Rochester

Team ID: 7

Zechen Wang (zwang88)

Ziyu Song (zsong10)

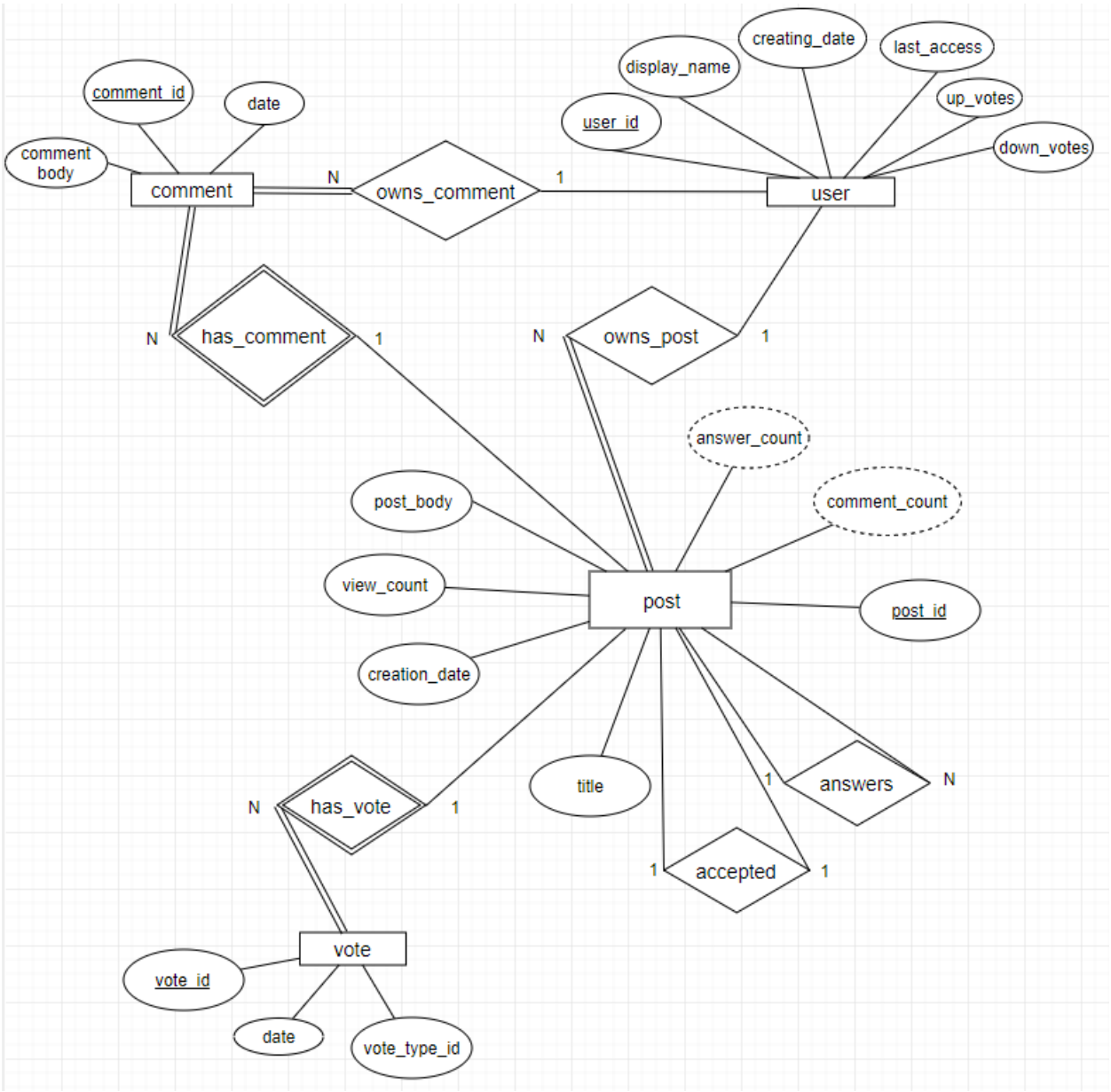
Project Description:

This project is mainly addressing an archive of Stack Overflow content, including users, votes, comments, and posts. Along with our designed database, we also designed a web interface for users to quickly access our database.

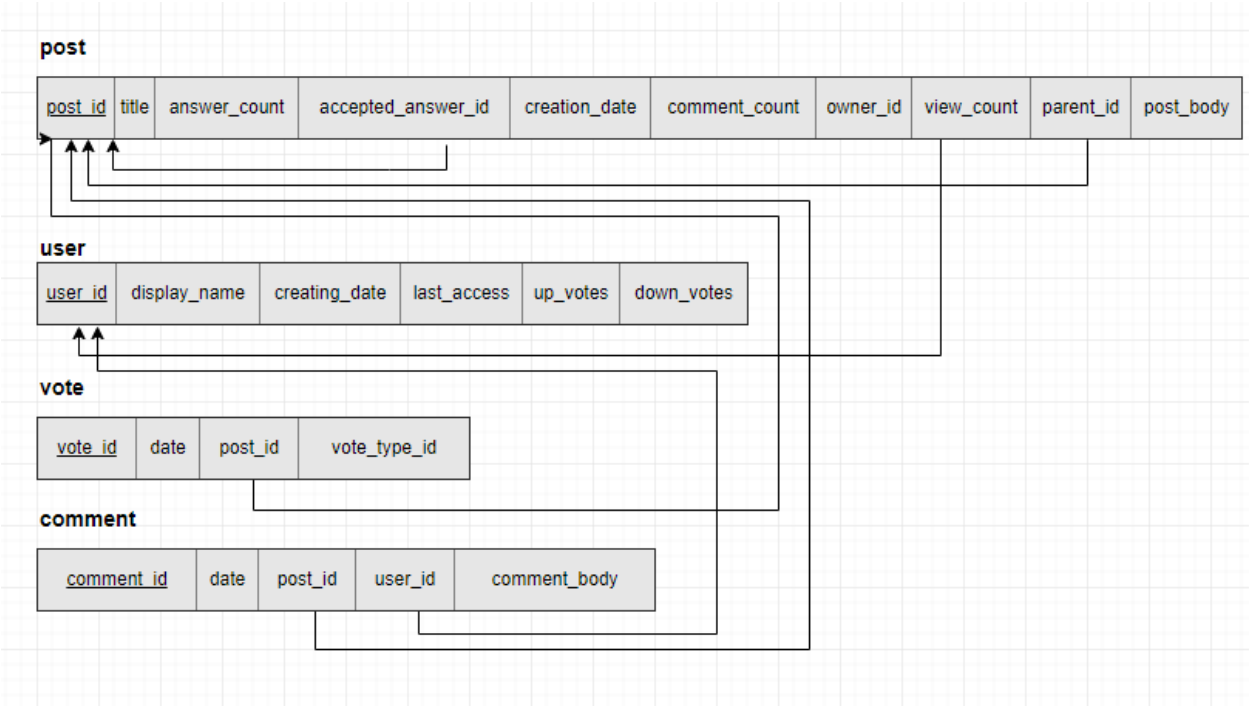
(*Homepage*: http://betaweb.csug.rochester.edu/~zwang88/group7_web_m4v6/homepage.html)

The target user of our database is Stack Overflow website users and website administrator. The database is built for website development purposes. Users of the website can enter their user_id, post_id, tag…and get detailed information about their posts. The administrator can record votes, comments… Users of Stack Overflow will be able to see the popularity of questions and their activities on this website. Administrators can see what types of questions are most popular and they can administer and operate the website/forum accordingly.

A detailed description of a selection of the features mentioned above is further discussed in the "Summary" session later in this report. The updated ER diagram and relational schema are given below. Milestone 3 and 4 share the same set of ER-diagram and relation schema.



ER-diagram, updated for milestone 3 and 4



Relation Schema, updated for milestone 3 and 4 (only foreign keys shown)

We also included a list of functional dependencies to show it is in BCNF. Since this is already listed in milestone 3 task A, the list will be shown at the end of this report.

Summary:
Compared to the previous milestone (milestone 3) of this project, we made several improvements for this final milestone (aka the final version of our project). We improved and redesigned the database functions, features, and web-interface of the database.

In milestone 3, when a user views the database, they can only go back to the previous page by clicking the "go back" arrow on the webpage. In this milestone, we added return buttons to all the web pages. Users can choose to either go back to the previous page ("Add another one"), go back to each relation homepage ("Back to table home"), or go back to the homepage ("Back to home page").

When we were trying to improve this feature, we started with using history.go() on the click event to implement the "back to homepage" button. Unfortunately, this will cause the button not to respond, or go to the wrong page, when the user directly visits the sub-pages. We were considering this as a safe option for our website since we expect the users to exclusively access the sub-pages via the homepage, and visit the homepage first under all circumstances. However, after we added more features to the database, such

as adding the sorting feature to all the attributes of each relation, history.go() click event is no longer working properly. To solve this problem, we changed the button to a form format that links to the target webpage (.html).

We added the "update" function to users, posts, and comments. Users of the website can choose to update a user, a post, or a comment by its corresponding ID. We also added a more specific restriction to our functions to help our database achieve better performance: "All fields are required" for all "add" and "update" functions.

We added a "sorting" feature to our database to help users sort results by column. This sorting feature is currently applied to any view actions that may have a large table of query results. When viewing the result page, users of the website can sort the table by columns in ascending order by clicking the name of the column in the table head (for instance, order by user ID).

We improved the "add new data" function of our database. In milestone 3, the website we designed will only allow the user to enter one data entry each at a time; in this milestone, the user can choose to enter a single data entry or add bulk of data (from a CSV file). This part is also the most challenging part of this milestone. At the homepage of each relation (users, posts, comments, and votes), there are two options where users can choose to either add an individual data entry or directly read in a .csv file which consists of all the data entries they want to add. For testing purposes, we created five testing .csv files for each relation. Testing .csv files are named as "new_data_*.csv". The sample files we used for testing can be obtained following this link:
https://drive.google.com/drive/folders/18_uOjxJkFbnqGxaAnk1dOceO8M95Ahqs?usp=sharing

Examining the entire project from where we are when writing this report, the database we built grows bits by bits from a proposal in milestone 1 to a prototype database that can be remotely accessed from a web-based interface. In general, the website we developed in milestone 4 is a major improvement in its user-friendliness and its capability of remotely delivering MySQL database queries correctly and efficiently. It follows the database design we made in milestone 3 which is modified and improved from milestone 2, and fulfills most of the proposals we made in milestone 1. By following the algorithm in the textbook in the early stages of the entire project, we transformed our ER model and relation schema into BCNF without too much effort. Since milestone 4 is a continuation and a modification of milestone 3 mostly with the web interface part, the database inherits the properties of the database in milestone 3 and is kept in BCNF, as we show the schema in this report.

Future Works:

There are still a lot of things we wish to add to our website. One of them is to beautify our web page by adding more decorations, for example, the Stackoverflow logos on the website. Also, there are some alignment issues with the buttons, and the partition of our website is probably not the most user-friendly. Besides that, there are still a lot more functions we want to achieve on our websites, such as a function to help users quickly search for keywords so that they can choose to view specific content by their interests.

Furthermore, if we could collect more data, we hope we could also build a page when the admin and the general user enter their login information, the system will recognize their identity permission and give them a different level of access based on matched identity.

APPENDIX

List of Functional Dependencies:

post_id → accepted_answer_id
post_id → parent_id
post_id → title
post_id → answer_couunt
post_id → certain_data
post_id → comment_count
post_id → owner_id
post_id → view_count
post_id → post_body

user_id → display_name
user_id → creating_date
user_id → last_access
user_id → up_votes
user_id → down_votes
user_id → {user_id, display_name, creating_date, last_access, up_votes, down_votes}
user.user_id → comment.owner_id
user.user_id → comment.user_id

vote_id → date
vote_id → post_id
vote_id → vote_type_id
vote_id → {vote_id, date, post_id, vote_type_id}

comment_id → date
comment_id → post_id
comment_id → user_id
comment_id → comment_body
comment_id → {comment_id, date, post_id, user_id, comment_body}