12 May 2022

Online Forum Database System

Pengxiang Huang, Han Linghu, Zihan Li, Boyi Chen

School of Data Science, The Chinese University of Hong Kong, Shenzhen

Abstract--These remain the abstract

Index Terms—Remain the index term

—————————— ◆ ——————————

1. Background & Introduction

It is common for many student-programmers from CUHKSZ encounter many familiar programming questions in their projects or assignments. To solve those questions, their solutions mainly include: searching online, emailing TAs or professors for help, uploading questions in class WeChat group, or making appointments with TAs or professors during their office hours. Searching online sometimes may not be efficient because the blogs or some guidance information may not directly answer the assignment question. Even worse, programmers need to spend a lot of time filtering a vast amount of information. It becomes hard for them to get answers directly when the homework question is not relevant to the results on the website. Uploading questions on the class WeChat group could get detailed guidance and response. But the new WeChat group will be created every semester for other students who may encounter the same problems. The connection between students who have already taken this course with the students taking this course right now is broken in this way. Raising questions during office hours is not convenient for programmers to solve their questions immediately since they need to make an appointment and wait until that day comes. Therefore, our group would like to take the first step to change the current situation by providing an online Q&A platform with the support of an online forum database.

The target users of our database are programmers in a specific community who have similar knowledge backgrounds, for instance, CUHKSZ students who major in CS. By limiting the range of users to a particular group, we can ensure that the users will have similar problems related to their projects or assignments, saving them from filtering vast amounts of information. It would be easier for them to get help. The critical component of our project in the frontend is several blogs. The main body of the blog is the question, or problem users encounter during their programming procedure, which is stored in the database. To help users better explain their needs, we allow users to attach files and images to a certain blog. And the files and images are stored in the remote cloud server, while the relationships between images and files are stored in the database. Additionally, we allow users to directly answer others’ questions by leaving a comment under a certain blog, matching the answers and questions. If the user who raises the question is still confused after reading others’ comments, he/she can also leave a comment under others' answers to get a further reply. To increase the performance of searching through the database, we made great efforts. We divide blogs into different groups, and each group also has many subgroups. So the user can reduce the search range and get the most relevant blogs. Besides, we allow users to like and follow blogs or groups to find the information they want more easily next time.

To make our database more efficient and space-saving, we made efforts to do normalization on tables and introduce indexes in our project. Additionally, we built a fantastic UI in the frontend and robust backend server to hide the detailed implementation and manipulation of the database. So users can focus on the Q&A procedure, and they don't need to worry about how to get the desired information from the database, as the functions carry out all the queries in the frontend and backend.

1. DataBase Structure Design

*A. Requirement & Specification*

The main component of our project is the online forum database. There are three main entities in the database, namely “user”, “blog\_questions” and “blog\_answers”. The “user” entity stores the account information about the user, including their name, email address, password and etc., which is used for identity authentication when the user enters the online forum. The core of our project is that users can post a blog to raise a question and get relevant answers from other users, which brings about the “blog\_questions” entity and the “blog\_answers” entity to become the key features. Those two entities store the questions and answers information about the blog, including content, author\_id, etc. We also implement several relationship schemas to support the interaction between the user and the blog. For instance, the user can like a blog or follow a blog. With the purpose of making our database more efficient and space-saving, we spare efforts on doing normalization and applying indexes to our project.

*B. Entity-Relation Diagram*

The ER-diagram of our system contains 12 entities. The entities are user, group, sub\_group, blog\_questions, blog\_answers, attached\_file, attached\_picture, user\_like\_answer, user\_like\_question, user\_follow\_question, user\_follow\_group and user\_view\_question.

User Anyone who wants to have access to our system needs an account, so the user entity has the related information of the user’s account, such as username, password, email address, and the URL of the profile photo. In the process of identity authentication, the system will throw an exception if it detects that the identity information does not match the records stored in the user entity.

Group In our system, each blog has a related group, such as CSC3170, which is convenient for users to find the blogs in the corresponding group. Moreover, it serves as the filter conditions in the process of searching, which improve the efficiency of searching. The group entity has group\_name, description, create\_time, update\_time, amount\_of\_follows. “Group\_name” is designed to store the course number of the computer science courses offered in CUHKSZ, such as CSC3170 and CSC4001. “Description” is designed to store the detailed information of the course, such as Database System and Software Engineering. “Create\_time” is designed to store the time of the creation of this group. “Update\_time” is designed to store the time period of the update of the blogs belonging to this group. “Amount\_of\_follows” is designed to store the amounts of users who follow the group.

Sub\_group In our project, each blog not only has a related group but also has a related subgroup, such as Assignment1. This can be served as the filter conditions in the process of searching, which improve the efficiency of searching. The sub\_group entity has group\_name and sub\_group\_name. “Group\_name” is designed to store the course number of the computer science courses offered in CUHKSZ, such as CSC3170 and CSC4001. “Sub\_group\_name” is designed to store the assignments in each course, such as Assignment1 and Project.

Blog\_questions The entity is of great significance in our online forum system. It has title, author\_id, group\_type, sub\_group\_type, content, code, like, follow, hot, create\_time, update\_time, views. “Title” is designed to store the title of the blog. “Author\_id” is designed to store the author of the blog, which is used to determine the authority of the deletion of the blog. “Group\_type” is designed to store the course number to which the blog belongs, such as CSC3170 and CSC4001. “Sub\_group\_type” is designed to store the corresponding assignment to which the blog belongs, such as Assignment1 and Project. “Content” is designed to store the content of the blog. “Code” is designed to store the code of the blog since our system provides an online code editor and compiler. “Like” is designed to store the amounts of users who like the blog. “Follow” is designed to store the amounts of users who follow the blog. “Hot” is designed to store the popularity value of the blog, which is used to determine the order of the blogs displayed in the “Hot Blogs” part. The value is highly related to likes, favors, views, and the creation time. The newer the questions posted, the larger the likes, favors, and views, and the higher the popularity value. “Create\_time” is designed to store the time period of the creation of this blog. “Update\_time” is designed to store the time period of the update of the answers belonging to this blog. “Views” is designed to store how many times users have viewed the blog.

Blog\_answers The entity is of great significance in our online forum system. It has question\_id, father\_answer\_id, content, like, author\_id, and create\_time. “Question\_id” is designed to store the blog question to which the answer belongs. “Father\_answer\_id” is designed to store the blog answers to which the solution belongs. Only one of the values between “Question\_id” and “Father\_answer\_id” is not null since the answer can only reply to the question or the solution. “Content” is designed to store the content of the answer. “like” is designed to store the amounts of users who like the answer. “Author\_id” is designed to store the author of the answer. “Create\_time” is designed to store the time period of the creation of this answer.

Attached\_file Some blogs or answers may have attached files, which will be stored in this entity. The entity has URL, corresponding\_question, corresponding\_answer, and create\_time. “URL” is designed to store the URL of the file. “Corresponding\_question” is designed to store the blog question to which the file belongs. “Corresponding\_answer” is designed to store the blog answer to which the file belongs. Only one of the values between “Corresponding\_question” and “Corresponding\_answer” is not null since the file can only belong to the question or the answer. “Create\_time” is designed to store the time period of the creation of this file.

Attached\_picture Some blogs or answers may have attached pictures, which will be stored in this entity. The entity has URL, question, answer, group\_name, and create\_time. “URL” is designed to store the URL of the picture. “Question” is designed to store the blog question to which the picture belongs. “Answer” is designed to store the blog answer to which the picture belongs. “Group\_name” is designed to store the group to which the picture belongs. Only one of the values among “Question”, “Answer” and “Group\_name” is not null since the picture can only belong to the question, the answer, or the group.

User\_like\_answer In our online forum system, users can click the button to like the answer so that those information will be stored in this entity. The entity has id and answer\_id. “Id” is designed to store the id of the user. “Answer\_id” is designed to store the id of the answer. Therefore, each record means the user like the answer.

User\_like\_question In our online forum system, users can click the button to like the question so that that information will be stored in this entity. The entity has id and question\_id. “Id” is designed to store the id of the user. “Question\_id” is designed to store the id of the question. Therefore, each record means the user like the question.

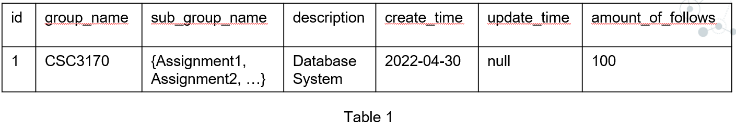
User\_follow\_question In our online forum system, users can click the button to follow the question so that that information will be stored in this entity. The entity has id and question\_id. “Id” is designed to store the id of the user. “Question\_id” is designed to store the id of the question. Therefore, each record means the user follows the question.

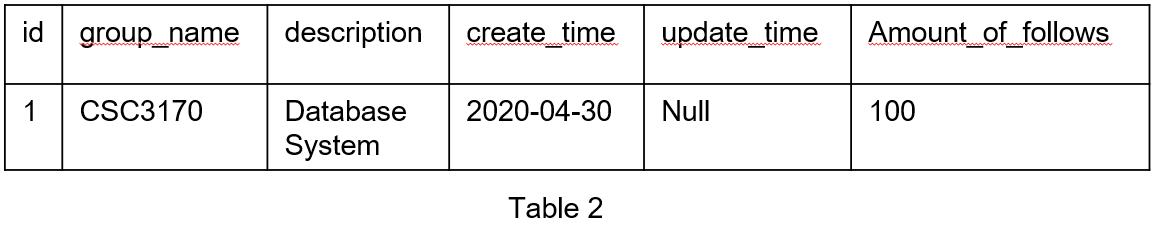
User\_follow\_group In our online forum system, users can click the button to follow the group so that those information will be stored in this entity. The entity has an id and group\_name. “Id” is designed to store the id of the user. “Group\_name” is designed to store the name of the corresponding group. Therefore, each record means the user follows the group.

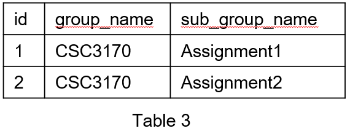
User\_view\_question In our online forum system, users can click the blog to view the detailed question so that those information will be stored in this entity. The entity has id, question\_id, and time. “Id” is designed to store the id of the user. “Question\_id” is designed to store the id of the question. “Time” is designed to store how many times the user views the question.

*C. Schema & Normalization*

In our project, we spare a lot of effort on normalization. At first, we intend to reach the first normal form, so we reconstruct our database. For example, "Table 1" shows our initial group table design with a tuple of subgroup names. Therefore, we split the group table into group table (Table 2) and sub group table (Table 3). In this case, the first normal form is achieved.







Based on the first normal form, we also try to reach the second and third normal form. As a result, we create a unique id for each table to enable all nonprime attributes to be fully functionally dependent on the primary key (id). Apparently, there do not exist nonprime attributes in our tables transitively dependent on the primary key (id). Therefore, the second normal form and third normal form are also implemented. As a result, the relational schema is shown below:

|  |
| --- |
| user(id, email, username, password, photo, major, grade, create\_time, update\_time) |
| group(id, group\_name, description, create\_time, update\_time, amount\_of\_follows) |
| sub\_group(id, group\_name, sub\_group\_name) |
| blog\_questions(id, title, author\_id, group\_type, sub\_group\_type, content, code, lang, content\_format, like, follow, hot, create\_time, update\_time, views) |
| blog\_answers(id, question\_id, father\_answer\_id, content, code, lang, content\_format, like, author\_id, create\_time) |
| attached\_file(id, url, corresponding\_question, corresponding\_answer, create\_time) |
| attached\_picture(id, url, question, answer, group\_name, create\_time) |
| user\_like\_answer(id, answer\_id) |
| user\_like\_quesition(id, question\_id) |
| user\_follow\_question(id, question\_id) |
| user\_follow\_group(id, group\_name) |
| user\_view\_question(id, question\_id, time) |

Some interpretations about foreign keys referencing:

* “group\_name” in sub\_group refers to “group\_name” in group: each sub group belongs to a group.
* “author\_id” in blog\_questions refers to “id” in user: each blog belongs to a user.
* “group\_type” in blog\_questions refers to “group\_name” in group: each blog belongs to a group.
* “sub\_group\_type” in blog\_questions refers to “sub\_group\_name” in sub\_group: each blog belongs to a sub group.
* “question\_id” in blog\_answers refers to “id” in blog\_questions: each answer belongs to a blog.
* “author\_id” in blog\_answers refers to “id” in user: each answer belongs to a user.
* “corresponding\_question” in attached\_file refers to “id” in blog\_questions, and “corresponding\_answer” in attached\_file refers to “id” in blog\_answers: each file belongs to a question or an answer.
* “question” in attached\_picture refers to “id” in blog\_questions, “answer” in attached\_picture refers to “id” in blog\_answers, and “group\_name” in attached\_picture refers to “group\_name” in group: each picture belongs to a question, an answer or a group.

*D. Index & Hashing*

1. Brief Introduction

To introduce index in our project to speed up some frequently used queries. We first search on the internet to know some main kinds of indexes:

* T-Tree Index
* B-Tree Index
* Full-Text Index
* Spatial Index
* Hash Index

After taking the structure of our database into consideration, we decided to focus on the hash index and B-Tree index, which can fulfill our requirements.

1. Hash Index

The first kind of index we thought of is the hash index. The main advantage of hash index is the fact that hashes can be much smaller than the indexed value itself, which can be space-saving. And the query speed is relatively fast for exact lookups if there are not many conflicts.

But after searching online and reading some formal documentation about MySQL, we found that hash index also has disadvantages. The main problem is that this index type can’t be used to do anything other than a simple lookup. So its usage is limited. What's worse, the hash index is only supported by the "Memory" engine, and the hashed index tables are all stored in the main memory, so the data is temporary and volatile. When the database restarts or encounters some system failure, the data are all gone. But the requirement of our database is that it stores all the questions and answers so users can quickly solve their problems by simply searching the database or posting a blog to get some help. So the hash index does not satisfy our project’s requirement and was not adopted.

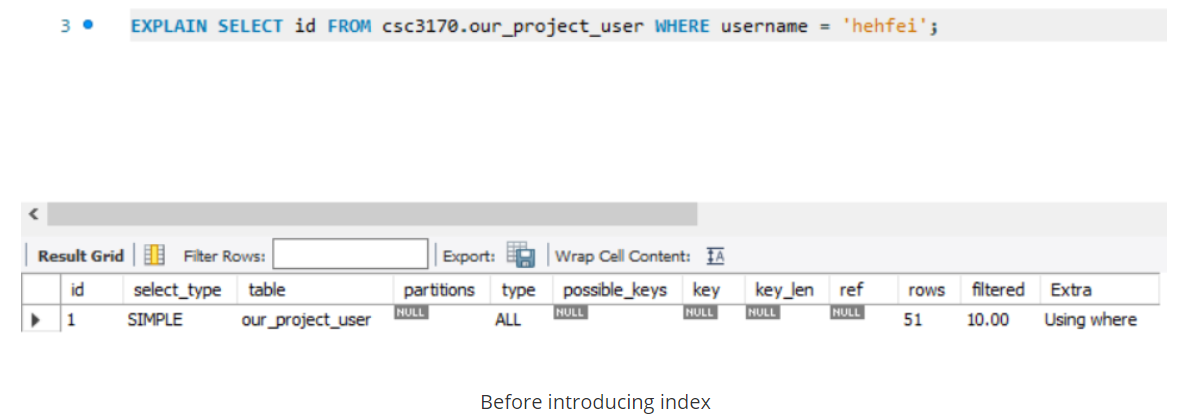
c）B-Tree Index

The B-Tree index is the most commonly used index, and it can speed up queries that match a full value. It uses the B-Tree structure to store the index, so the average searching time is reduced to log(h) (h is the height of the tree). Compared to the all-table scanning of a simple query without an index, the improvement is vast.

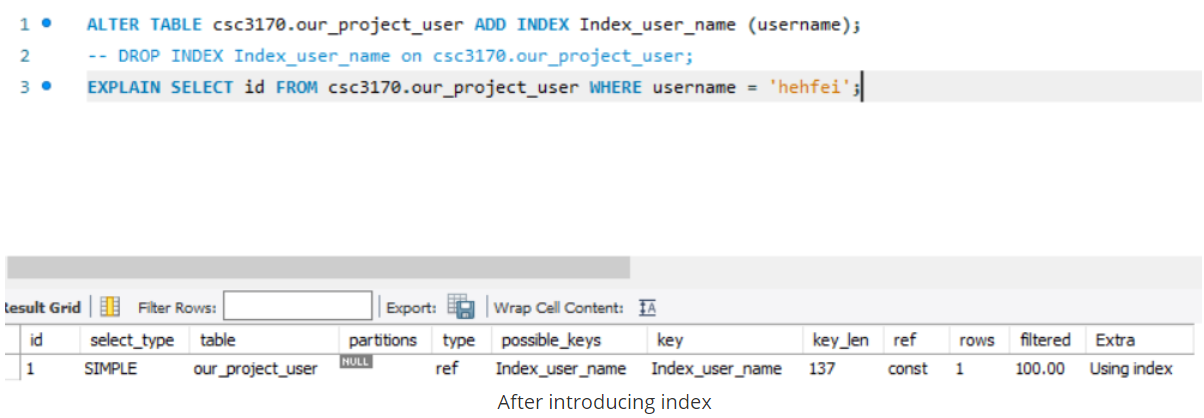
Considering the trade-off between search performance and system cost, we add indexes to some frequently queried columns in certain tables, e.g. the "username" column in the "User" table:

(To better understand the importance of index, we use the "explain" syntax, the meaning of some keywords are listed below:

* type: information about join type:
* ref: multiple rows can be accessed for a given value, so we are using a standard, non-unique index to retrieve them.
* ALL: full table scan.
* possible\_keys & key: indexes that could be used for the particular query and the index chosen by the optimizer as the most efficient one.
* rows: an estimate of how many rows the query will scan.
* Extra: prints additional information relevant to how the query will be executed.)



Before we add index to the table, when we want to get the id of a user with a specific name, the whole table is scanned, and around 51 rows are scanned. (if the table grows larger, this number will also grow), so the speed is very slow.

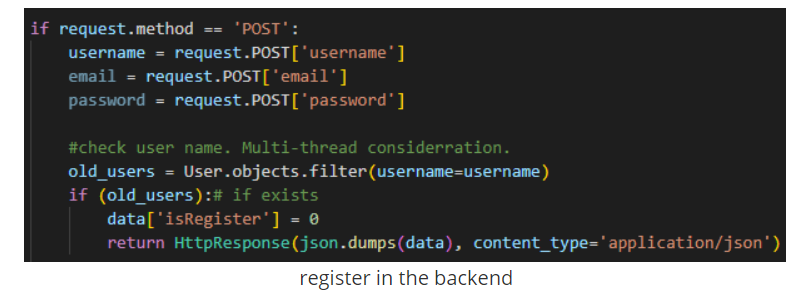


But after we introduce the B-Tree index, the query will take advantage of the index, and the number of rows to be scanned is significantly reduced.

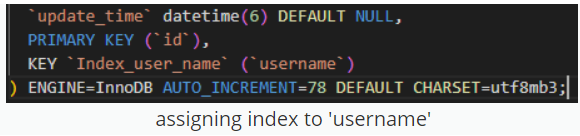
The following part is where we introduce indexes with some explanation.

1. "user" Table ---- 'username'

The 'username' attribute of the "user" table is frequently queried when the user register for an account or he/she wants to log in, so an index is assigned to it.

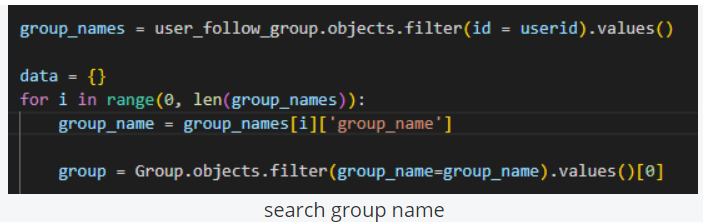


The above figure shows the register procedure; the server will check if the given 'username' is already in the database.

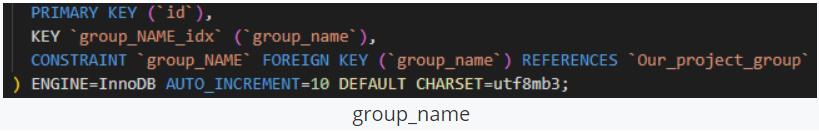


1. "sub\_group" Table ---- 'group\_name'

The 'group\_name' of the 'sub\_group' table is frequently queried when the user wants to get the groups he/she follows.

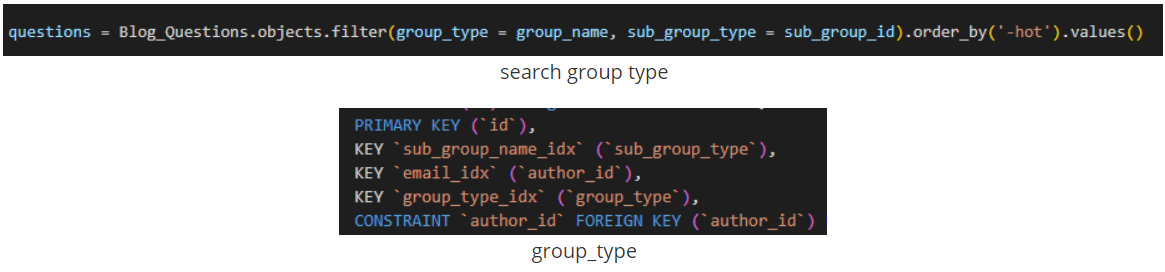


The above figure shows the procedure that the server returns the group\_name of sub\_groups the user follows.



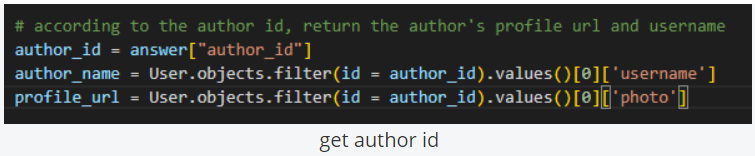
1. "blog\_questions" Table ---- 'sub\_group\_type'

The 'sub\_group\_type' attribute of the "blog\_questions" table is frequently queried during the range limited searching procedure when the user wants to search for blogs in the specific sub-group.

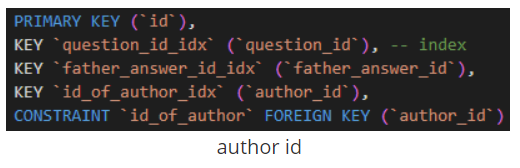


1. "blog\_answer" Table ---- 'author\_id'

The 'author\_id' attribute of the "blog\_answer" table is frequently queried when showing the answers to the blog.



The above figure shows the procedure of getting the information about the author of the answer.



1. SQL Function

This section is mainly about SQL functions. Many SQL requirements and their back-end SQL sentences and corresponding results will be demonstrated. They include operational queries as well as analytic queries. All the functions are implemented in our system and can be accessed in the GUI.