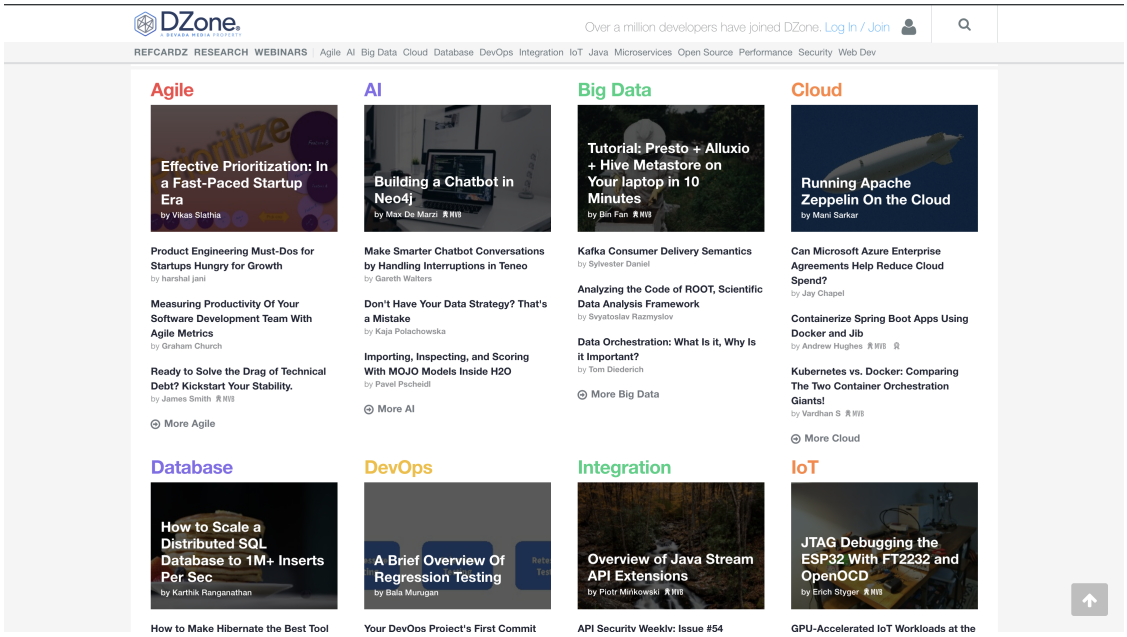
CSCE-608 Project #1

Minreng Wu ◇ 529005234

October 29, 2019

1. Application Description

Technical blog is a very popular way for software engineers to share personal development experience with the other developers and learn new techniques. Below is the blog site called DZone which I visited most:



My application is about a blog system like DZone. It's a web application. It will provide interfaces for some simple operations, such as querying a article, counting the number of articles for each category, etc. As this project is aimed to practise the skills for designing and developing a database but not building a mature product, my application will only completes very few features of a real blog system and will skip implementing the querying interfaces for some tables.

For a blog system, we should have some basic entities like User, Article, Comment, Role and Category. User represents the users of this blog site; Article represents the articles of this blog site; Commtent represents the comments that users give to some articles; Role represents the role users have, such as administer, common users; Category represents the article categories, for example, there're categories like Agile, Cloud, DevOps, etc.

Based on these 5 entities, I designed my applications with these 5 interfaces:

1. Query users with conditions like username and role. (Select)

2. Query articles with conditions like username and category (Join)
3. Add/Update an article (Insert/Update)
4. Count how many comments each article has (group by & aggregate)
5. Delete the articles with the least comments (delete & subquery)
6. calculate the average comment amount for articles in each category (group by & aggregate)

2. E/R Diagram

As we discussed in the first section that we have 5 entities, then we can list the attributes for each entity:

For User entity, it will have some attributes: username, nickname, email, description, avatar. Username is the unique id for this user and it will be used to login; nickname is the name for this user to display on the site and different users can use the same nickname; email is the registered email for the user and it is unique; description is a brief self-description and it could be in 300 letters; avatar is the full URL path of the avatar picture in the file system. Username is the primary key.

For Role entity, it will have two attributes: id and roleName. Id is the auto-increment primary key and it has no specific real meaning; roleName represents the name of this role and it's unique. Id is the primary key.

For Article entity, it will have four attributes: id, title, description and body. Id is the auto-increment primary key and it has no specific real meaning; title represents the article title; description represents the brief introduction to the article; body represents the article content. Id is the primary key.

For Comment entity, it will have two attributes: id and content. Id is the auto-increment primary key and it has no specific real meaning; content represents the comment content. Id is the primary key.

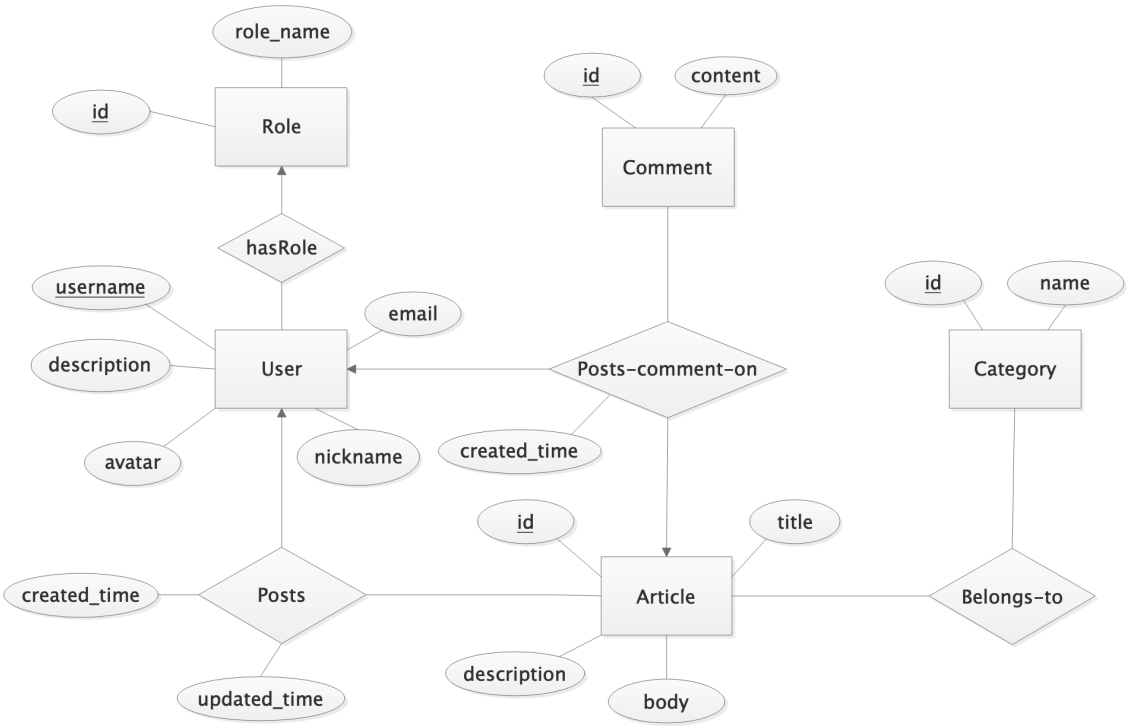
For Category entity, it will have two attributes: id and name. Id is the auto-increment primary key and it has no specific real meaning; name represents the name of the category. Id is the primary key.

For this blog system, it will also have some relationships:

1. Users can post new articles. One article will only have a single creater user, but one user can create more thab one articles so it's a many-one relationship.
2. Users can post comments on the article they want to comment as well. It's a multiple relationship. One comment can only be posted on one article by one user, but one article can have more than one comments from different users.
3. Each user has a role. One user can only have one role, but one role can be assigned to more than one users. so it's a many-one relationship.

4. Each article belongs to a category. One article can belong to more than one categories and one categories can have more than one articles. so it's a many-many relationship.

Based on the above discussion, we can draw a E/R diagram like the following:



3. Database Schema and Normalization

Transfer the E/R diagram to relations as below:

1. User(<u>username</u>, nickname, email, description, avatar)
2. Role(<u>id</u>, roleName)
3. hasRole(<u>username</u>, <u>roleId</u>)
4. Article(<u>articleId</u>, title, description, articleBody)
5. Posts(<u>username</u>, <u>articleId</u>, createdTime, updatedTime)
6. Comment(<u>commentId</u>, commentContent)
7. PostsCommentOn(<u>username</u>, <u>commentId</u>, <u>articleId</u>, createdTime)
8. Category(<u>categoryId</u>, categoryName)

9. BelongsTo(articleId, categoryId)

The nontrivial FDs in each relation:

1. User's FDs: {username → {nickname, email, description, avatar}, email → username}
2. Role's FDs: {roleId → roleName}
3. hasRole's FDs: ∅
4. Article's FDs: {articleId → {title, description, articleBody}}
5. Posts's FDs: {{username, articleId} → {createdTime, updatedTime}}
6. Comment's FDs: {commentId → commentContent}
7. PostsCommentOn's FDs: {{username, commentId, articleId} → createdTime}
8. Category's FDs: {categoryId → categoryName}
9. BelongsTo's FDs: ∅

Then we check whether each relation is in BCNF:

1. For User: username and email are two candidate keys. The left sides are all superkeys. Therefore, User is in BCNF.
2. For Role: roleId is the only key. The left sides are all superkeys. Therefore, Role is in BCNF.
3. For hasRole: As the FDs is an empty set, BelongsTo is in BCNF.
4. For Article: articleId is the only key. The left sides are all superkeys. Therefore, Article is in BCNF.
5. For Posts: {username, articleId} is the only key. The left sides are all superkeys. Therefore, Posts is in BCNF.
6. For Comment: commentId is the only key. The left sides are all superkeys. Therefore, Comment is in BCNF.
7. For PostsCommentOn: {username, commentId, articleId} is the only key. The left sides are all superkeys. Therefore, PostsCommentOn is in BCNF.
8. For Category: articleId is the only key. The left sides are all superkeys. Therefore, Category is in BCNF.
9. For BelongsTo: As the FDs is an empty set, BelongsTo is in BCNF.

Therefore, we can conclude these relations are all in BCNF.

Since we have some many-one relationships, we can try to merge them to the "many" side of entity. After merging, we can get a set of new relations:

1. User(<u>username</u>, nickname, email, description, avatar, roleId)
2. Role(<u>roleId</u>, roleName)
3. Article(<u>articleId</u>, <u>username</u>, title, description, articleBody, createdTime, updatedTime)
4. Comment(<u>commentId</u>, <u>username</u>, <u>articleId</u>, commentContent, createdTime)
5. Category(<u>categoryId</u>, categoryName)
6. BelongsTo(<u>articleId</u>, <u>categoryId</u>)

The nontrivial FDs in this new relation set:

1. User's FDs: {username → {nickname, email, description, avatar, roleId}, email → username}
2. Role's FDs: {roleId → roleName}
3. Article's FDs: {articleId → {username, title, description, articleBody}, {username, articleId} → {createdTime, updatedTime}}
4. Comment's FDs: {commentId → username, articleId, commentContent, {username, commentId, articleId} → createdTime}
5. Category's FDs: {categoryId → categoryName}
6. BelongsTo's FDs: ∅

Definitely, there're some changes in these two relations: Article and Comment.

1. For User, the only key is username. So User is in BCNF.
2. For Article, the only key is articleId. So Article is in BCNF.
3. For Comment, the only key is commentId. So Comment is in BCNF.

Therefore, we can conclude these modified relations are still all in BCNF.

I think it's hard to improve my schemas by 3NF and 4NF, because the FDs of current schemas are mostly in the pattern that primary key determines the other attributes.

4. Creating Tables

Based the analysis in the above sections, We can design the SQL schemas like the following:

Table 1: User

Attribute Name	Datatype	Comment
id	INT	primary key; not null; auto increment
username	VARCHAR(45)	unique; not null
nickname	VARCHAR(45)	not null
email	VARCHAR(256)	unique; not null
description	VARCHAR(300)	a brief personal description; it could be null
avatar	VARCHAR(45)	a URL refered to the avatar picture; it could be null

Table 2: Role

Attribute Name	Datatype	Comment
id	INT	primary key; not null; auto increment
name	VARCHAR(45)	role name; not null; unique

Table 3: Article

Attribute Name	Datatype	Comment
id	BIGINT	primary key; not null; auto increment
username	VARCHAR(45)	foreign key to table User; not null
title	VARCHAR(100)	article title; not null
description	VARCHAR(500)	brief description; not null
body	BLOB	article body; it could be very large; not null
created_time	TIMESTAMP	using current timestamp as default value
updated_time	TIMESTAMP	it will be set to the current timestamp on update

Table 4: Comment

Attribute Name	Datatype	Comment
id	BIGINT	primary key; not null; auto increment
username	VARCHAR(45)	foreign key to table User; not null
article_id	BIGINT	foreign key to table Article; not null
content	VARCHAR(1024)	comment content; not null
created_time	TIMESTAMP	using current timestamp as default value

Table 5: Category

Attribute Name	Datatype	Comment
id	INT	primary key; not null; auto increment
name	VARCHAR(80)	category name; not null; unique

Table 6: Belongs To

Attribute Name	Datatype	Comment
id	INT	primary key; not null; auto increment
article_id	BIGINT	foreign key to table Article; not null
category_id	INT	foreign key to table Category; not null

The SQL DDL script is in the source file, please see more information in README.

5. Data generation

I choose to get my data by generating some fabricate data.

First, I use random strings with different length to fill the usename, nickname, email, description, avatar attributes for a user.

Then, for each user I simulated the process of creating 10 articles. I also use random strings with different length to fill the attributes for an article.

Also, for each user I simulated the process of creating 10 comments. But I will fill the attribute of articleId as 1 initially. Because the comments should be pointed to one article and we can't assigned the articleId before we generating all articles. After we generate all the articles, we can re-scan the comments csv and generate the articleId again.

For Category and Role, they are all small tables. So we can generate the data before we start to generate User, Article and Comment.

For more explanation, please see the README file.

6. User Interface and Functions

I use Vue.js to build the frontend pages and use Java and Spring Framework to build the backend service.

For more explanation, please see the README file.