CS307 Project Part1 Report

Name: Huimin LIANG
StudentID: 12210161
Class#: Lab Tue 1-2

Name: Zhuo WANG
StudentID: 12210532
Class#: Lab Tue 1-2

Name: Ben CHEN
StudentID: 12212231
Class#: Lab Tue 1-2

CS307 Principles of Database System
(Fall, 2023)

Southern University of Science and Technology(SUSTech), China
Students of Department of Computer Science

28th December 2023

Contents

1 Contribution

The basic information and contribution of our members are as follow

Name & SID	**Contribution**	**Rate**
Huimin LIANG 12210161	ER Diagram	30%
	Basic API Design	
	Report Composition	
Zhuo Wang 12210532	Database Design	35%
	Basic API Design	
	Advanced API Test	
Ben CHEN 12212231	Report Composition	35%
	Basic API Test	
	Advanced API Design	

2 Database Design

In this section, the description and visualization of our database would be given.

2.1 Designing the E-R Diagram

According to our design, the entity relationship is shown below.

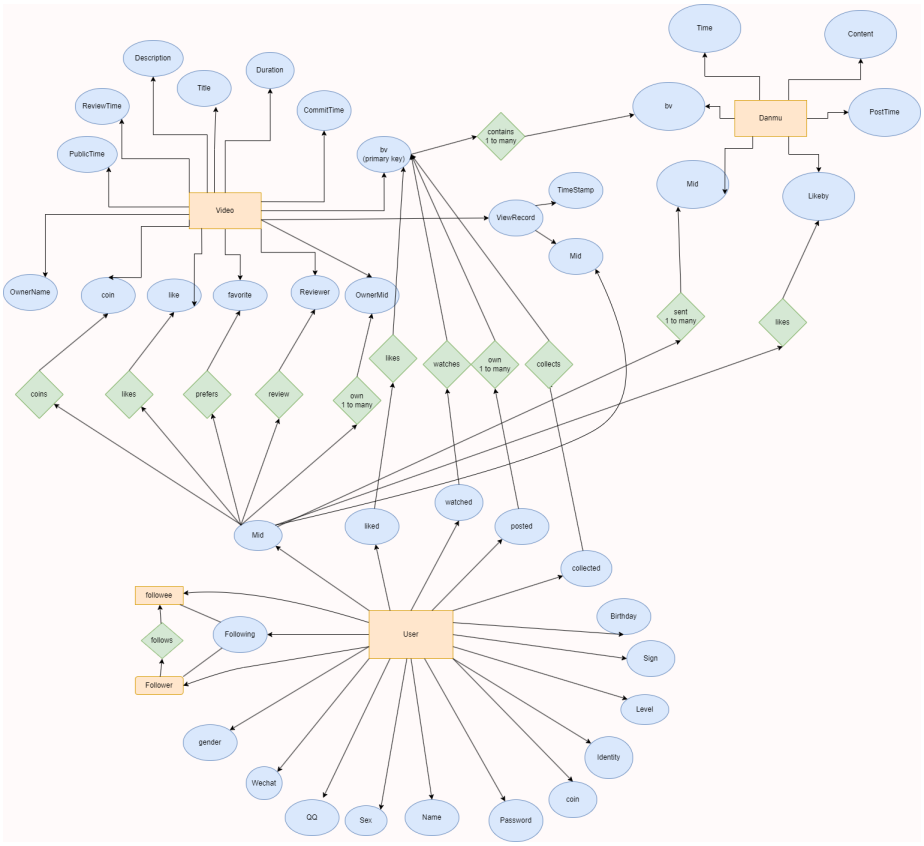

Figure 1: ER Diagram

2.2 Database Description

In our design, the database implements the functions with three major tables, **users**, **videos** and **danmus** and five subtables which would be described in detail.

Main tables

Users The table records the information of users related to themselves, excluding the ones that might imply others' information such as mid of other users

that they follow and videos they liked or coined. The primary key of the table is **mid** `BIGINT`.

Videos The table is designed to store information related to videos, including details about the video content, ownership, and reviewing activities. The primary key of the table is **bv** (`VARCHAR(255)`), which serves as a unique identifier for each video.

Danmus The table is designed to store information about user-generated comments or danmus associated with specific videos. Each record in the table represents a single danmu, and the primary key is **id** (`SERIAL`), which serves as a unique identifier for each danmu.

subtables

followings The `followings` table is designed to track the relationship between users. Each record in the table represents a user following another user. The primary key of the table is composed of two fields: **followedMid** and **followerMid**, ensuring the uniqueness of each follow relationship.

coin, videolike, favorite This three tables share similar structure, therefore they will be introduced at the same time.
The `videolike` table is designed to record instances where users express liking for specific videos. The `coin` table is designed to track instances where users contribute coins to specific videos. The `favorite` table is designed to capture instances where users mark videos as favorites. The primary key of these table is a composite key consisting of **bv** and **mid**, ensuring the uniqueness of each liking relationship.

danmulike The `view` table is designed to record instances of users viewing videos. Each record represents a user viewing a specific video. The primary key is a composite key consisting of **bv** and **mid**, ensuring the uniqueness of each view relationship.Also, there is a attribute viewTime FLOAT to record how the user has watched this video;

Primary key and Foreign key
all the subtables above have a 2-attributes primary key and they are also the foreign key. This implies that the user/video/danmu must exist in the `Users/Videos/Danmus` table, establishing referential integrity. For example, in table followings, the primary key, consisting of **followedMid** and **followerMid**. Additionally, two foreign keys are defined, both referencing the `mid` field in the `Users` table.

Cascade Deletion
all the subtables have cascade deletion, which means when either atributes' value is deleted, the relationships will be automatically deleted.This ensures consistency and prevents invalid follow relationships in the database. For example, in the Followings table, the foreign key constraints include `ON DELETE CASCADE`, indicating that if a user referenced by either **followedMid** or **followerMid** is deleted from the `Users` table, the corresponding follow relationships in the

`followings` table will be automatically deleted.

2.3 Datagrip

The following figure is generated from Datagrip

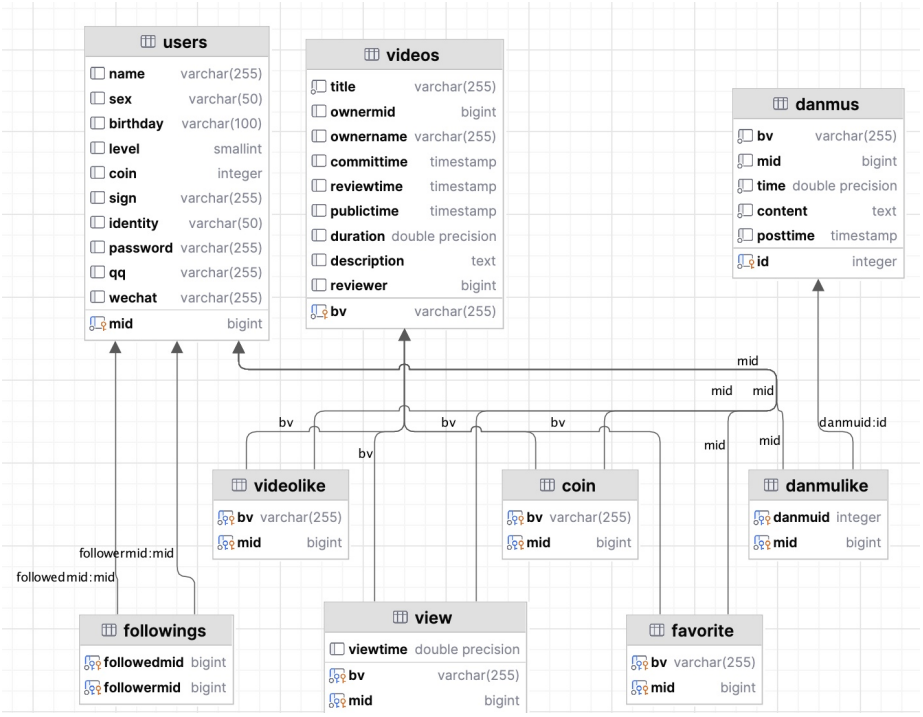

Figure 2: Datagrip

3 Basic API

Introduction: The implementation of the video danmu website involves the creation and utilization of various services to manage the database, user interactions, video operations, danmu functionalities, and recommendations. In this section, we provide an in-depth overview and documentation of the basic APIs that form the backbone of the entire system.

3.1 DatabaseService

3.1.1 Functionality Overview

The `DatabaseService` interface provides methods for managing the underlying database. It includes functionality for retrieving group members, importing data into the database, truncating tables, and performing basic arithmetic operations through the database.

3.1.2 Methods

`getGroupMembers()` `Description:` Acknowledges the authors of this project by returning a list of group members' student IDs.

`importData()` `Description:` Imports data into an empty database. Invalid data will not be provided.
`Implementation:In this interface method, we divide the insertion into three methods, corresponding to users, videos, and danmus. In our database design, we have not only constructed three tables, but based on three main tables, we have also created six auxiliary tables. Therefore, the method for inserting users also completes the insertion for users and followings. Similarly, the method for inserting videos handles the insertion for five tables related to videos. This approach makes the implementation clearer.`

`truncate()` `Description:` Truncates all tables in the database. This method is intended for local benchmarking to clean the database without dropping it, and it does not affect the score.

3.2 UserService

3.2.1 Functionality Overview

The `UserService` interface provides methods for user management, including user registration, account deletion, following/unfollowing other users, and retrieving user information.

3.2.2 Methods

`register(RegisterUserReq req)` `Description:` Registers a new user. `password` is a mandatory field, while `qq` and `wechat` are optional OIDC fields.

`Implementation:`
This interface method inserts the provided user information into all tables related to the user and generates a unique mid for the user.

`deleteAccount(AuthInfo auth, long mid)` `Description:` Deletes a user. The method removes relationships with other users, watch records, coin records, etc.
`Implementation:`
In this interface, we will delete all information related to the user. Since we have implemented cascade deletion, simply deleting the user's mid and videos will consequently remove all related information from all tables.

`follow(AuthInfo auth, long followeeMid)` `Description:` Follows the user with `followeeMid`. If already followed, unfollows the user.
`Implementation:`
After verifying the user's valid information, we need to add the mids of the follower and the followed to the 'followings' table.

`getUserInfo(long mid)` `Description:` Gets the required information (in DTO) of a user.
`Implementation:`
Since much of the given data is stored in arrays for rational design, we have extracted the data from these arrays to create auxiliary tables. Now, we need to retrieve the information from these auxiliary tables, which requires querying multiple auxiliary tables and integrating the information.

3.3 VideoService

3.3.1 Functionality Overview

The `VideoService` interface provides methods for managing videos, including posting, deleting, updating video information, searching for videos, calculating average view rate, identifying hotspots, reviewing videos, donating coins, liking videos, and collecting videos.

3.3.2 Methods

`postVideo(AuthInfo auth, PostVideoReq req)` `Description:` Posts a video. The commit time is set to the current time.
`Implementation:`
After validating that all the information is legitimate, we need to insert the video information into the 'videos' table and return a newly created BV number. How this BV number is generated will be discussed in the subsequent advanced API.

`deleteVideo(AuthInfo auth, String bv)` `Description:` Deletes a video. Can be performed by the video owner or a superuser.
`Implementation:`
We need to verify whether the user has the authority to delete this user's video, which involves a multifaceted check. After passing the check, we need to delete

all information related to the video. Since we have implemented cascade deletion, simply deleting the video's BV number will consequently remove all other related information.

`updateVideoInfo(AuthInfo auth, String bv, PostVideoReq req)` `Description:` Updates the video's information. Only the owner of the video can perform this operation.
`Implementation:`
This is a relatively complex method involving numerous checks, particularly ensuring the validity of time-point checks related to 'reviewtime' and 'publictime', which requires very clear logic. Following this method, we will update the video information in the 'videos' table and reset both 'reviewtime' and 'reviewer' to null.

`searchVideo(AuthInfo auth, String keywords, int pageSize, int pageNum)`
`Description:` Searches for videos based on keywords.
`Implementation:`

This is a highly challenging API design where simply using the 'like' query in SQL language is insufficient for fulfilling the complete query requirements (passcnt: 13/20). Therefore, for completeness, we switched to regular expression matching. This approach correctly handles instances where a keyword appears in the string to be matched without being counted repeatedly. At the same time, it ensures that a keyword can be matched multiple times in a single string, thus meeting the final requirements. (PS: The learning curve is indeed steep.)

`getAverageViewRate(String bv)` `Description:` Calculates the average view rate of a video.
`Implementation:`
First, we need to validate the effectiveness of the video. After that, we find all the records of the video in the 'view' table using its BV number, sum up all the 'viewtime', and calculate the total number of records. Finally, we divide the total 'viewtime' by the number of viewing records.

`getHotspot(String bv)` `Description:` Gets the hotspot of a video.
`Implementation:`
We need to cut the video into segments every 10 seconds and store the results in a set. After obtaining all the segments, we then identify the most popular segment.

`reviewVideo(AuthInfo auth, String bv)` `Description:` Reviews a video by a superuser.
`Implementation:`
Similarly, we first need to verify the validity of the video. Then, using the BV number, we locate the record of the video in the 'videos' table and set the 'reviewtime' to the current real-time point.

`coinVideo(AuthInfo auth, String bv)` `Description:` Donates one coin to the video.
`Implementation:` Similarly, we need to check the validity of the video. Additionally, we need to verify whether the user has watched the video, whether there are coins remaining in the inventory, and whether the user has already donated coins, among other checks. After passing these validations, we will add the video's BV information and the user's mid information to the 'coin' table.

`likeVideo(AuthInfo auth, String bv)` `Description:` Likes a video.
`Implementation:`
Similar to 'coinVideo', after all validations are passed, we will add the video's BV information and the user's mid information to the 'videolike' table.

`collectVideo(AuthInfo auth, String bv)` `Description:` Collects a video.
`Implementation:`
Similar to 'coinVideo', after completing all the validations, we will add the video's BV information and the user's mid information to the 'favorite' table.

3.4 DanmuService

3.4.1 Functionality Overview

The `DanmuService` interface provides methods for managing danmus, including sending danmus to a video, displaying danmus in a time range, and liking danmus.

3.4.2 Methods

`sendDanmu(AuthInfo auth, String bv, String content, float time)` `Description:` Sends a danmu to a video.
`Implementation:`
Since the id of a danmu is designed as an auto-increment primary key in our database, when sending a new danmu and inserting data into the 'danmus' table, we need to immediately obtain the newly generated id using 'generateKey' and return it.

`displayDanmu(String bv, float timeStart, float timeEnd, boolean filter)`
`Description:` Displays the danmus in a time range.
`Implementation:`
The most crucial aspect of this API is to validate the timeliness, and then retrieve the ids of all danmus within a specific time frame.

`likeDanmu(AuthInfo auth, long id)` `Description:` Likes a danmu.
`Implementation:`
First, we need to verify the user's information, then check the validity of the corresponding video and danmu, and also ascertain whether the user has already watched the video, among other checks. Once all validations are passed, we then add information to the 'danmulike' table, including the danmu's id information and the user's mid information.

3.5 RecommenderService

3.5.1 Functionality Overview

The `RecommenderService` interface provides methods for recommending videos and friends based on user preferences and interactions.

3.5.2 Methods

`recommendNextVideo(String bv)` `Description:` Recommends a list of top 5 similar videos for a given video.
`Implementation:`
To recommend videos to a user, we need to gather information from the 'view' table, querying the entire database for all users' viewing information, and then recommend the five most relevant videos to the user.

`generalRecommendations(int pageSize, int pageNum)` `Description:` Recommends videos for anonymous users based on popularity.
`Implementation:`
This is a highly challenging interface design, requiring us to combine five metrics: like rate, completion rate, collection rate, coin donation rate, and danmu interaction rate, to score all videos in the database. This query aims to assess each video's overall performance in user interactions (collections, coin donations, likes, danmus) and viewing behaviors. By this method, videos with high user engagement and popularity can be identified and recommended to users.

`recommendVideosForUser(AuthInfo auth, int pageSize, int pageNum)` `Description:` Recommends videos for a user, restricted to their interests.
`Implementation:`
In this interface, we need to complete a design called 'UserInterests'. First, a temporary table named 'UserInterests' is created in the query (using the WITH statement), which contains all the users (referred to as 'friendid') that the people followed by the current user ('followedmid') are following. We need to find all the videos watched by these 'friendid's but not by the user themselves, and recommend them to the user. If the user has no interest in these videos, then we simply revert to the interface that recommends popular videos.

`recommendFriends(AuthInfo auth, int pageSize, int pageNum)` `Description:` Recommends friends for a user based on common followings.
`Implementation:`
Defining Common Followings: - A temporary table named 'CommonFollowings' is created, including the list of followings for both the current user (u1.followedMid) and potential friends (u2.followedMid). This table selects those followed by the people the current user (u1) follows (u2), excluding the current user themselves.

Excluding Already Followed Users (ExcludedUsers): - A temporary table 'ExcludedUsers' is created, listing all users already followed by the current user, to avoid redundant recommendations.

Selecting Potential Friends: - Potential friends (i.e., users not yet followed by the current user) are selected from 'CommonFollowings' and are not listed in 'ExcludedUsers'. - The number of common followings (commonFollowings) for

each potential friend is calculated, along with recording each potential friend's mid (user ID) and level (user level).

Sorting and Limiting the Results: - Finally, the query sorts the results in descending order by the number of common followings (commonFollowings). If the number of common followings is the same, then it sorts by user level (level) in descending order and then by user ID (mid) in ascending order. - The results are then paginated using LIMIT and OFFSET.

4 Advanced API

On the basis of implementing the basic APIs, we refactor the implementation to optimize the performance by adapting **multi-threading** and to reinforce the reliability by generating unique BVs.

Multi-Threading To enable the multiple threading function, two steps need to be done that we shall add a configuration to the thread pool and remove foreign keys while inserting since the procedure of inserting might violate the foreign key constraint. The configuration is composed in the way that creating a class with annotation `configuration` and provides the necessary arguments.

```
@Configuration
public class AsyncConfig {
    1 usage
    private static final int MAX_POOL_SIZE = 50;
    1 usage
    private static final int CORE_POOL_SIZE = 20;

    @Bean("asyncTaskExecutor")
    public AsyncTaskExecutor asyncTaskExecutor() {
        ThreadPoolTaskExecutor asyncTaskExecutor = new ThreadPoolTaskExecutor();
        asyncTaskExecutor.setMaxPoolSize(MAX_POOL_SIZE);
        asyncTaskExecutor.setCorePoolSize(CORE_POOL_SIZE);
        asyncTaskExecutor.setThreadNamePrefix("async-task-thread-pool-");
        asyncTaskExecutor.initialize();
        return asyncTaskExecutor;
    }
}
```

Figure 3: Async Configuration

Then async mode is available, and by adding another annotation `EnableAsync` at the beginning of class we can enable asynchronous method. `AsyncTaskExecutor` is used for invoking the inserting methods and with a `CountDownLatch`, the main thread is able to create the foreign keys that we will use in our APIs. The latch at the main thread will keep blocking until the threads is executed and the latch counts down by one.

```java
CountDownLatch counter = new CountDownLatch(3);
asyncTaskExecutor.submit(() -> {
    importUserRecords(userRecords);
    counter.countDown();
});
asyncTaskExecutor.submit(() -> {
    importVideoRecords(videoRecords);
    counter.countDown();
});
asyncTaskExecutor.submit(() -> {
    importDanmuRecords(danmuRecords);
    counter.countDown();
});
```

Figure 4: AsyncTaskExecutor

The foreign keys are created at the end of inserting all the tables

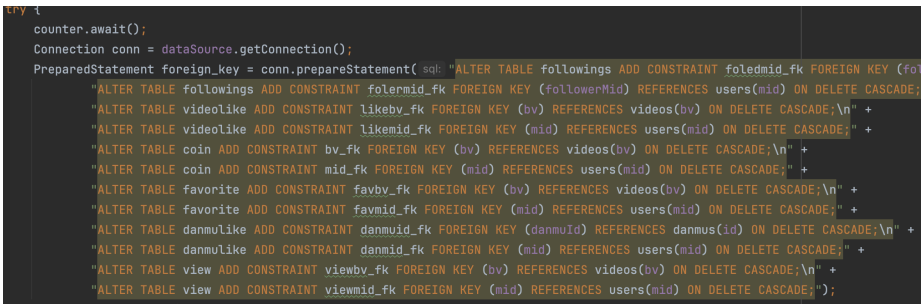

Figure 5: Foreign Keys

The comparison of the primitive and optimized is

Script	**small_data**	**big_data**
Primitive	2 min 32 s	13 min 54 s
Optimized	22 s	56 s

Table 1: Runtime result

BV Generation Given that a real BV number consists of a prefix followed by ten characters, of which five are digits and the other five are case-sensitive English letters, considering random permutations, the total number of possible combinations will reach 10^{21}. Relative to the actual number of videos, even with randomly generated BV numbers, the probability of hash collisions will be negligible.

```
1 usage
private String generateNewBv(Connection conn) throws SQLException {
    Random random = new Random();
    String newBv;
    do {
        newBv = "BV" + generateRandomDigits( length: 5) + generateRandomLetters( length: 5);
    } while (bvExists(conn, newBv));
    return newBv;
}

1 usage
private String generateRandomDigits(int length) {
    Random random = new Random();
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < length; i++) {
        sb.append(random.nextInt( bound: 10));
    }
    return sb.toString();
}

1 usage
private String generateRandomLetters(int length) {
    Random random = new Random();
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < length; i++) {
        char letter = (char) (random.nextInt( bound: 26) + (random.nextBoolean() ? 'A' : 'a'));
        sb.append(letter);
    }
    return sb.toString();
}
```

Figure 6: Generating

User Privilege Taking into account the real-world scenarios of video websites, which typically involve different identities operating on the platform, here we only consider three types of users: ordinary users, website administrators, and auditors. For different user identities, we will grant varying operation permissions to different users.

```sql
-- 创建普通用户
CREATE USER normal_user WITH PASSWORD 'password';
-- 授予表格的增删改查权限
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO normal_user;


-- 创建管理员用户
CREATE USER admin_user WITH PASSWORD 'password';
-- 授予所有权限
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin_user;


-- 创建审核员用户
CREATE USER auditor_user WITH PASSWORD 'password';
-- 授予查改权限
GRANT SELECT, UPDATE ON ALL TABLES IN SCHEMA public TO auditor_user;
```

Figure 7: Privilege

5 Conclusion

In the design of the entire video website, the basic APIs ensure the fundamental applications of the video site. On top of ensuring these basic applications, we strive to make the implementation of each API as close to a real-life danmu (bullet chat) website as possible. Beyond this, we aim to optimize the completion time for each API, which, when applied to a real video website, can enhance the user experience. Additionally, we can consider the security of data transmission, such as implementing further encryption measures.

Overall, the entire API system we designed using JDBC and database connection pooling demonstrates exceptional performance in terms of fast multi-threading, thread safety, high concurrency efficiency, and database data security.