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

12th November 2023

Contents

1 Information and Contribution

1.1 Contribution

The basic information and contribution of our members are as follow

Name & SID	**Contribution**	**Rate**
Huimin LIANG 12210161	Report Composition	33%
	DDL review (Task 2)	
	Data analysis (Task 4)	
Zhuo WANG 12210532	ER Diagram Design (Task 1)	33%
	DDL design (Task 2)	
	FileIO script (Task 4)	
Ben CHEN 12212231	DDL review (Task 2)	33%
	Data import script and improve (Task 3)	
	SQL query design (Task 4)	

1.2 Language

We applied the following languages in our design

- Java 19.0.1 2022-10-18
  Java(TM) SE Runtime Environment (build 19.0.1+10-21)
  Java HotSpot(TM) 64-Bit Server VM (build 19.0.1+10-21, mixed mode, sharing)

1.3 Test Environment

We deployed two machines in our test

- Mac mini 2020(Processor: Apple M1 8 core)
- Lenovo Legion(Processor: Intel Core i7-12700H 20 core)

2 ER Diagram

2.1 Designing the E-R Diagram

This section outlines the methodology and tools used to design the ER diagram for our video website. Additionally, it provides a visual representation of the ER diagram.

2.1.1 Methodology

To design the ER diagram, we followed a systematic methodology:
1. Identify Entities and Attributes:
- We identified the main entities in our database, such as Users, Videos, and Danmu. For each entity, we determined the relevant attributes, like User ID, Video Title, and Danmu Content.
2. Define Relationships:
- we focus on the relationships between entities and attributes, and see how they connect.
3. Constraints:
- We assigned constraints to relationships to specify the nature of the connection between entities.
4. Normalization:
- We ensured that the ER diagram followed the principles of normalization to minimize redundancy and improve data integrity.

2.1.2 Tools Used

We used draw.io, a popular online diagramming tool, to create our ER diagram. Draw.io allows us to easily represent the entities, attributes, and relationships in our database model."

2.1.3 ER Diagram Snapshot

Below is a snapshot of the ER diagram we designed for our website:

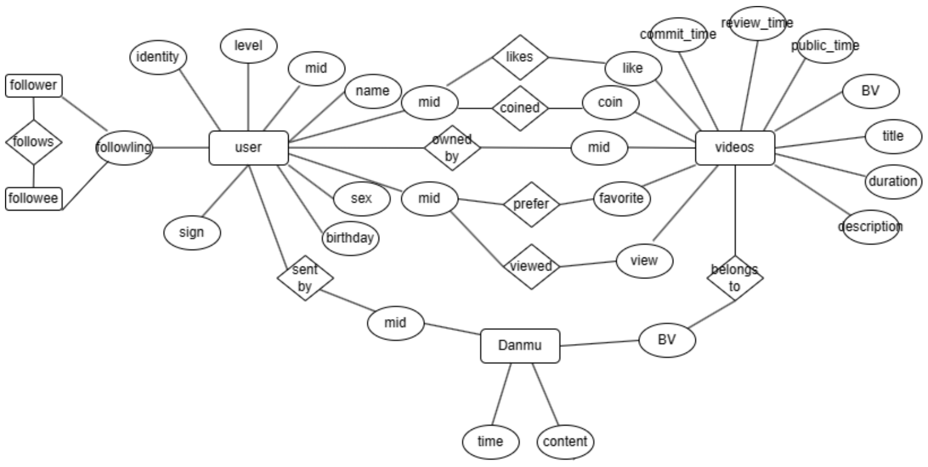


Figure 1: ER Diagram

The ER diagram visually represents the entities, their attributes, and the relationships between them, serving as a foundational step in our database design.

2.2 Entities and Attributes

In this ER diagram, we can visually identify all the entities, attributes, and relationships. The entities include "users," "videos," and "danmu." The attributes for each entity are as follows:

2.2.1 Users

- **Entity:** Users
- **Attributes:**
- mid (unique identifier)
- name
- sex
- birthday
- level
- sign
- following (sub-entity)
- identity (with values "user" or "superuser")

2.2.2 Videos

- **Entity:** Videos
- **Attributes:**

- BV (unique identifier string)
- title
- owner mid (unique identifier of the video owner)
- commit time
- review time
- public time
- duration
- description
- reviewer
- like
- coin
- favorite
- view

2.2.3 Danmu

- **Entity:** Danmu
- **Attributes:**
- BV
- mid
- time
- content

2.2.4 Relationships

In this diagram, we can clearly observe the relationships between different attributes. For example, the relationship between "danmu.mid" and "users.mid" is denoted as "sent by," indicating that the danmu is sent by a user with a specific user ID. Another example is the relationship between "danmu.BV" and "videos.BV," signifying that danmu exists within the time frame of a specific video BV.S

This ER diagram provides a comprehensive view of the structure of the database and how entities and attributes are interconnected.

3 Database Design

3.1 Raw Data

The data files are `users.csv`, `videos.csv` and `danmu.csv`.

3.1.1 users.csv

The columns are described below

- **mid**: the unique identification number for the user
- **name**: the nick name created by the user
- **sex**: include but not limited to biological sex
- **birthday**: the birthday set by user
- **level**: user engagement evaluated according to system decision criteria
- **sign**: the personal description created by the user
- **following**: a list holding all the users' *mid*s that this user follows
- **identity**: a value in to indicate the user's role

3.1.2 videos.csv

The file contains the following column

- **BV**: the unique identification string of a video
- **title**: the name of video created by video owner.
- **owner mid**: the mid of the video's owner
- **commit time**: the time when the owner committed this video
- **review time**: the time when the video was inspected by its reviewer
- **public time**: the time when the video was made public for all users
- **duration**: the video duration
- **description**: the brief text introduction given by the uploader
- **reviewer**: the mid of the video reviewer
- **like**: a list of the mids of the users who liked this video
- **coin**: a list of the mids of the users who have given this video their coins
- **favorite**: a list of the mids of the users who favorited this video
- **view**:

3.1.3 danmu.csv

The file describes the information shown below

- **BV**: the BV of the video where the Danmu was sent
- **mid**: the mid of the user who sent the Danmu
- **time**: the time of the video that Danmu appears
- **content**: the content of the Danmu

3.2 Design

3.2.1 Analysis

In this part, we delved into the provided data files and their structures, aiming to identify the primary entities, attributes, and define the relationships between them.

Entity Identification: we identified the main entities that would form the basis of our database. These entities include "users," "videos," and "danmu."

Attribute Identification: For each entity, we recognized the associated attributes. For example, in the "users" entity, attributes such as "mid," "name," "sex," and others were identified.

Relationship Definition: We explored the relationships between entities. For example:

- **videos - owned by - user:** Describes the relationship where users own videos.
- **videos.like - likes - user.mid:** Represents the "likes" relationship between videos and users, where user.mid likes videos.
- **videos.coin - coins - user.mid:** Indicates the "coins" relationship between videos and users, where user.mid gives coins to videos.
- **videos.favorite - prefers - user.mid:** Defines the "prefers" relationship between videos and users, where user.mid marks videos as favorites.
- **videos.view - viewed - user.mid:** Depicts the "viewed" relationship between videos and users, where user.mid views videos.
- **Danmu.BV - belongs to - videos:** Specifies that danmu belongs to videos based on the BV (unique identifier).
- **Danmu.mid - sent by - user:** Illustrates the "sent by" relationship between danmu and users, where danmu is sent by user.
- **user.following.follower - follows - followee:** Represents the "follows" relationship between users, where a user with follower follows another user as a followee.

Data Normalization: We ensured that our database design adhered to the principles of normalization. In the subsequent sections, we will delve into this aspect in greater detail.

Through this analysis phase, we laid the foundation for the database design by identifying data entities, attributes, and relationships, ensuring that the structure of the database aligns with the project requirements. The subsequent sections will delve deeper into the specific structure and constraints of the database.

3.2.2 Structure

In this section, we established three schemas: ”usr,” ”video,” and ”danmu.” Within this section, we created three main tables and five auxiliary tables. The main tables are ”users,” ”videos,” and ”danmu,” while the auxiliary tables include ”followings,” ”favorite,” ”likes,” ”view,” and ”coin” tables. Let me introduce each of the main and auxiliary tables individually.

main tables

- **Users Main Table:**
  - **mid (bigint Primary Key):** mid, of data type bigint. We chose bigint as the data type for the primary key because during our numeric import tests, we found that int could not meet our requirements. Bigint is the most suitable data type for our needs. The primary key allows us to instantly locate each user and connect them with other auxiliary tables.
  - **name (varchar(255)):** We used the varchar(255) data type for ”name” because we considered that some users might have longer names.
  - **sex (varchar(50)):** We used the varchar(50) data type for ”sex” as we allow users to define their own gender. This decision eliminates the restriction of only retrieving male or female genders.
  - **level (int):** We employed the int data type for ”level,” considering that the ”level” has a range from 1 to 6 and is numeric.
  - **sign (text):** For ”sign,” we used the text data type because ”sign” serves as a self-introduction. We chose text as our data type as it can handle larger content.
  - **identity (varchar(50)):** We utilized the varchar data type for ”identity” as there are only two possible identities.

- **Videos Main Table:**
  - **BV ((archar(255))Primary Key):** BV, which is of data type varchar(255). We selected varchar(255) as the data type for the primary key because BV numbers consist of both digits and other characters. BV serves as a unique identifier for each video, allowing us to quickly locate and connect videos with other tables.
  - **title (varchar(255)):** We used the varchar(255) data type for ”title” to ensure it can accommodate sufficiently long video titles.

- **owner-mid (bigint):** This attribute is linked to the "mid" attribute in the "users" table. It represents the ID of the video's creator.
- **commit-time (timestamp):** It records the video's upload time and is of data type timestamp.
- **review-time (timestamp):** This attribute records the time at which the video was reviewed and is of data type timestamp.
- **public-time (timestamp):** It indicates the time when the video is made public to all users and is of data type timestamp.
- **duration (interval):** Video duration is stored using the interval data type. The interval data type is suitable for representing time durations.
- **description (text):** Similar to the "sign" attribute in the "users" table, "description" uses the text data type as it may contain longer text descriptions.
- **reviewer (bigint):** It is of data type bigint and stores the reviewer's ID, connecting to the "mid" attribute in the "users" table.

- **danmu.danmu Table:**
  - **id (bigserial PRIMARY KEY):** We used bigserial, which is a big integer with an auto-incrementing feature, as the data type of id. It is used to create a unique primary key. The choice of the 'bigserial' data type ensures each danmu has a unique identifier, and the identifier increments automatically, making it easy to manage.
  - **BV (varchar(255) not null):** THis attribute indicates which video contains this danmu. it connects with the video.BV. Therefore they have same data type. This attribute is assigned with constraint not null, which means this danmu must belong to some video.
  - **mid (bigint):** THis attribute indicates this danmu is senet by the user with this mid. it connects with the user.mid. Therefore they have same data type.
  - **time (interval):** The 'time' attribute stores the time at which the danmu appeared in the video. The 'interval' data type is chosen because it accurately represents time intervals, making it suitable for storing time information related to danmus.
  - **content (text):** The 'content' attribute stores the content of the danmu, typically in the form of text. The choice of the 'text' data type is appropriate because danmu content may contain large amounts of text, and using 'text' allows for the storage of text of varying lengths.

auxiliary table

- **usr.followings Table:** this table is the auciliary table of users, indicating user's following list.
  - **id (bigserial PRIMARY KEY):** We chose bigserial as its data type, which is a big integer with an auto-incrementing feature. It

is used to create a unique primary key. It ensures each relationship has a unique identifier, and the identifier increments automatically, making it easy to manage.

- **follower-id (bigint not null):** The 'follower-id' attribute stores the ID of the user who is following another user. This attribute is linked to Users.mid, therefor they share same data type. 'not null' ensures that a follower must have a following list even though it's empty.
- **followed-id (bigint):** The 'follower-id' attribute stores the ID of the user who is following another user. This attribute is linked to Users.mid, therefor they share same data type. It doesnt have 'not null' constraints, indicating that a follower's following list can be empty.

- **video.likes Table, video.coin Table, video.favorite Table, video.view:** This four auxiliary tables of video sharing similar structure.
  - **id (bigserial PRIMARY KEY):** The attribute id for these four table are set as bigserial datatype. This is the primary key of these table, so bigserial enables each table has a unique identifier, and the identifier increments automatically, making it easy to manage.
  - **who-likes/coins/favorites (bigint not null):** this attribute indicates the user with this mid likes/coins/favorites/views this video. This attribute is linked to users.mid, therefor they share same data type. These attributes are assigned not null, which means if these actions are done, there must be a user perform it.
  - **BV (varchar(225)):** this attribute indicates the video with this BV number got liked/coined/favorite/viewed. This BV is linked to video.BV, therefor they share same data type.
  - **last-time (interval)):** This attribute is for table view only.This attribute stores the duration of the user's last viewing of this video. As we selected interval datatype to store the video's duration, and interval can satisfy our needs, we chose interval as the datatype of last-time.

3.2.3 Scalability

1. Introduction to Scalability: scalability refers to the system's ability to accommodate growing user numbers, increasing data volumes, and additional features without sacrificing performance or user experience. In the context of our project, we only consider the part of adding additional features, as the data volume isn't as big.

2. Scalability Strategies:

In our database design, we have implemented several key scalability strategies to ensure that our system can efficiently adapt to evolving requirements:

- **Schema Separation:** We organized our data into separate schemas (usr, video, danmu) to simplify data management and maintenance. This separation of schemas allows for a logical organization of data and is instrumental in future expansions.

- **Table Separation:** Different types of data, such as user data, video data, and danmu data, are stored in separate tables. This approach reduces the complexity of individual tables, making them more manageable and facilitating future expansions.

- **Primary Table and Auxiliary Table:** Our database design comprises three primary tables and five auxiliary tables, ensuring the scalability of our database. The three primary tables store the fundamental attributes of entities, while the five auxiliary tables extend these attributes. We have carefully considered future requirements, allowing us to seamlessly add new features. For instance, if we decide to introduce a user logging feature, we can simply add a new log auxiliary table and link it to the "mid" attribute of the user, without the need to modify the existing tables. This approach ensures convenience and scalability during subsequent data processing and feature additions. Another example is the introduction of an account deactivation feature. When we deactivate a user account by removing the 'mid,' all associated data, such as likes, favorites, and followings, is also deleted. This ensures the maintainability and scalability of our database. This flexible design approach facilitates the enhancement of our database as new functionalities are integrated. It ensures that we can adapt and expand the database without significant disruptions or alterations to the existing structure.

- **Data Type Selection:** We made thoughtful choices for data types, such as using 'bigint' to store large integers, 'varchar' for strings, and 'interval' for time durations. These selections ensure that the database can accommodate various data types.

3.Conclusion: The implemented scalability strategies are crucial for maintaining our database's responsiveness and reliability as the user base and data volume expand. These strategies ensure that our system can easily accommodate new features, maintain data integrity, and efficiently scale with the evolving demands of the application.

3.2.4 Normalization

I. Introduction to Normalization:
Normalization is a fundamental database design technique that aims to eliminate data redundancy and enhance data integrity. It revolves around adhering to specific criteria known as the three normal forms: 1NF, 2NF, and 3NF.

- **First Normal Form (1NF):**

  1NF ensures that each table possesses atomic (indivisible) values, devoid of repeating groups, and contains only unique values.

- **Second Normal Form (2NF):**

  2NF builds upon 1NF by ensuring that all non-key attributes are functionally dependent on the entire primary key, removing partial dependencies.

- **Third Normal Form (3NF):**

3NF goes a step further by eliminating transitive dependencies, ensuring that non-key attributes do not depend on other non-key attributes within the same table.

- **Normalization in My Design:**

  This analysis shows that your database design adheres to the three normal forms, which helps eliminate data redundancy and improve data integrity, aligning with the principles of normalization.

II. Normalization in Our Design

- **1NF (First Normal Form) Users, Videos, Danmu Table:**

  The "usr.users", "video.videos" and "danmu.danmu" table complies with 1NF. Each row holds unique and atomic values, so they satisfied with 1NF.

  Followings, Likes, Coin, Favorite, View Tables:

  Auxiliary tables (e.g., followings, likes, coin, favorite, view) also fulfill 1NF criteria, featuring unique records with atomic attributes.

- **2NF (Second Normal Form)**

  Users Table:

  The primary key "mid" in the "usr.users" table is the candidate key, ensures that all non-key attributes (name, sex, birthday, level, sign, identity) are functionally dependent on it, conforming to 2NF.

  Videos Table:

  The "BV" primary key in the "video.videos" table assures that all other attributes (owner-id, title, commit-time, review-time, public-time, duration, description, reviewer) are functionally dependent on it, satisfying the second normal form.

  Danmu Table:

  The primary key "id" in the "danmu.danmu" table and the other attributes (BV, mid, time, content) are functionally dependent on it, aligning with 2NF.

  Followings, Likes, Coin, Favorite, View Tables: We set a specific ids for each table to identify then following list/likes/coin/favorite and view. Therefor auxiliary tables exhibit unique identifiers as primary keys, and the other attributes(e,g follower-id, followee-id, who-likes/coins...) are functionally dependent on these keys, complying with 2NF.

- **3NF (Third Normal Form)**

  Users Table:

  The "usr.users" table complied with 2NF, additionally, it has no transitive dependencies among non-key attributes,for example, name, sex, birth, level, sign, identity have no dependencies among each other, and each attribute directly depends on the primary key "mid."

  Videos Table:

In the ”video.videos” table, the non-primary attributes are independent from each other, title, owner-mid, commit-time, review-time, public-time, duration, description, reviewer have no dependencies with each other, but they all strong connect to primary key mid. Therefor, it conforms 3NF.

Danmu Table:

The ”danmu.danmu” table adheres to 3NF. Non-primary attributes (BV, mid, time, content) are independent from each other and they are directly dependent on the primary key ”id.”

Followings, Likes, Coin, Favorite, View Tables:

Auxiliary tables follow 3NF criteria, as non-primary key attributes directly depend on their respective primary keys, but don’t connect to each other.

III. Minimizing Data Redundancy

This database design minimizes data redundancy by segregating data into distinct tables (Schema Separation and Table Separation). Each table contains only relevant attributes, avoiding data duplication, which bolsters data integrity and streamlines maintenance.

IV. Conclusion of Normalization

In conclusion, this analysis demonstrates that our database design meticulously adheres to the three normal forms, significantly reducing data redundancy and enhancing data integrity, in alignment with the principles of normalization.

3.3 Conclusion

Here is the database diagram for a clearer understanding of our database design.

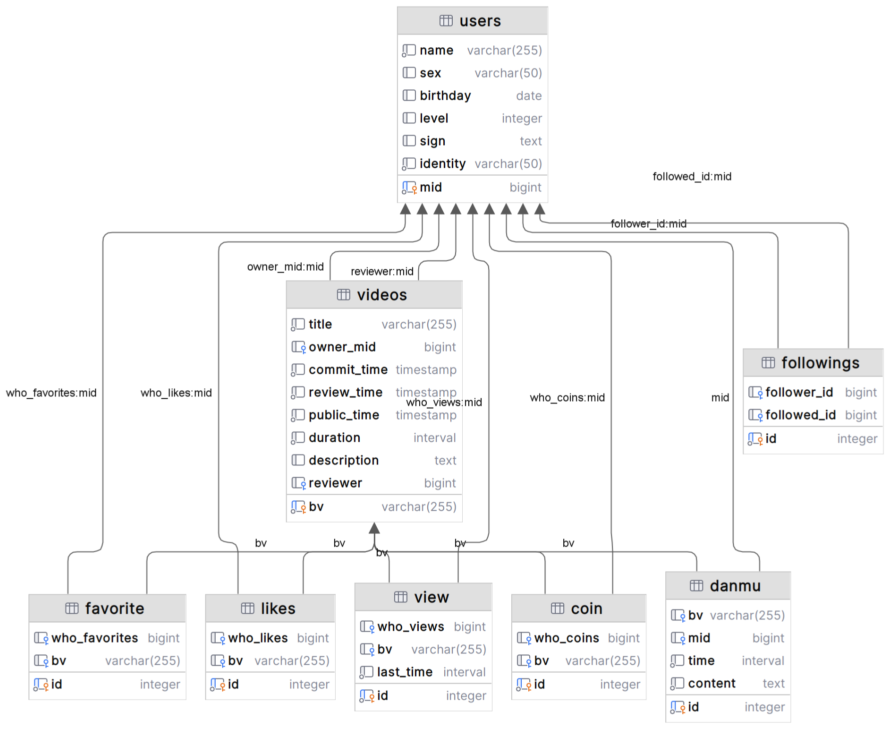


Figure 2: Diagram exported from Datagrip

In this project, we managed to design a database with PostgreSQL, with every information provided included. Our database design aligns with the three normal forms , helping minimize data redundancy and enhance data integrity. As the database diagram showed above, the proper utilization of primary keys and foreign keys ensures meaningful and well-defined relationships among the data. Additionally, Every table has its own primary key to be uniquely identified. Each table are part of a link, and there is no isolated tables, and no links formed a cycle.We made sure that every table has at least one mandatory not-null columns. And we used appropriate and reasonable datatype for every attribute.

Our database design is also highly structured and exhibits scalability. Employing schema separation, table separation, and the careful selection of data types ensures the flexibility of the database when addressing future expansion requirements.

We believe that this design will effectively meet future demands and provide a reliable basis for the successful operation of the application.

4 Data Import

In this part, we'll offer a guidance to run our import scripts. Next, an explanation of the principles of the scripts will be given. However, since the average method has a relatively low efficiency, we'll introduce a way to enhance the process. The strategies include

- Optimize the database design
- Modify the configuration of PostgreSQL
- Apply multiple threading in JDBC
- Use the transaction of PostgreSQL

4.1 Guidance

4.1.1 Prerequisite

To run our scripts, you should have these installed at first

- Java 19
- PostgreSQL 15
- CSVReader (Maven)
- JDBC PostgreSQL (Maven)

We also provide a user-friendly option that uses IntelliJ IDEA to run the program, which can be accessed through `https://github.com/chanbengz/SUSTech_CS307_2023F_Project1`(private for now).

4.1.2 Create Database

Then follow these steps to configure your database

1. execute "`initdb` /path/to/database & `createdb`" in your terminal
2. execute "`chmod` 700 /path/to/database"
3. execute "`pg_ctl` -D /path/to/database start" to start database
4. execute "`psql`" to enter CLI
5. type in "create database project1;" and press return key (enter key)
6. quit the CLI by pressing Ctrl-D
7. run the command "`psql` -d project1 -f /path/to/DDL.sql"

By now, you can run the import scripts. If you choose the option of IDEA, the only step is to open the project and run the code by clicking the button. Simple enough! If you'd like to do it manually, you can simply run the code by `javac` DBMS.java and `java` DBMS command.

4.1.3 Notice

Remember to change the username and password in the `connect` method of the script. Please neatly follow the steps. We highly recommend using IntelliJ IDEA to run the code.

4.2 Design

4.2.1 Code

The source code can be reached by `DBMS.java` in the archive.

4.2.2 Principles

The script mainly consist of the following five steps

1. Connect database
2. Import *user.csv* into tables in schema *usr*
3. Import *video.csv* into tables in schema *video*
4. Import *danmu.csv* into tables in schema *danmu*
5. Close database

In the script, the steps are executed one after one by the order in single thread.

In each import method, we firstly instantiate a CsvReader with arguments of the path of csv files and the encoding method, after which we create the **prepared statements** of insert in SQL. The other options remain unchanged, such as the autocommit. Next, we iterate the lines in the files and parse the fields into types that our database accepts, inserting the data into the prepared statements. For each line, we execute the update of the statements, meaning that after one statement is filled with enough data, it will be executed at once. In the main method, we record the time needed for every import step and the total time equals the sum of them.

4.3 Analysis

The result of successful import is

Tablename	users	followings
Records	37881	5958770

Table 1: Result of user.csv

Table	videos	view	likes	favorite	coin
Records	7865	163997974	86757948	79181895	80571520

Table 2: Result of video.csv

Table	danmu
Records	12478996

Table 3: Result of danmu.csv

CPU Avg Load	Memory Pressure	Disk Load	Workers	Memory Used
2.1	50%	12.3 MB/s	4	44 MB

Table 4: Performance Information

4.3.1 Performance

The whole process takes over 7 hours to import, which is unacceptable. In order to address the bottleneck, we look up for the activity monitor to see the performance of the test computer, and we got the following information

From the information, it should be noticed that the import didn't make full use of the performance our computer can provide. Beyond that, after searching the Internet for how to improve the insert performance, we were told that the uses of foreign keys and constraints will greatly reduce the performance for the reason that since the foreign keys refer to a existing table, the time to build the table and search items of it will be required, and that constraints will add up the time cost for each insert operation. Meanwhile, with the limitation of foreign keys we can never apply multi-threading to it.

We also noticed that the disk had a relatively low pressure. While we wondering why the disk was not used properly, we dug further into the principles of database. PostgreSQL has a feature named "Transaction", which is used to commit and execute a bunch of SQL operations at a time. However, the function of auto commit is enabled by default, which will automatically commit after one SQL is executed. That's a lot of waste, since the disk is waiting for DBMS to write on at the most of time. Besides, PostgreSQL offers custom configuration and it's not the best for every computer, so we came up with thoughts to optimize these arguments.

4.4 Enhancement

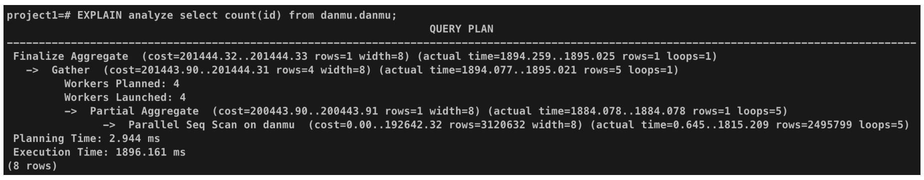
Based on the performance analysis, we propose four major strategies to improve our script. The first two steps are the optimization of settings of database and the last two steps are the optimization of operations of our script.

Configuration The configuration file locates at `postgresql.conf`. The file offers a clear description of what each argument means. So with our understanding about the arguments, we modify these following configuration

- `max_worker_processes` = 8
- `max_parallel_workers` = 8
- `max_parallel_workers_per_gather` = 4
- `max_parallel_maintenance_workers` = 4
- `work_mem` = 1024MB
- `max_wal_size` = 2GB
- `min_wal_size` = 100MB
- `wal_buffers` = 16MB

The configuration was obtained from *PGTune*(`https://pgtune.leopard.in.ua/`). These configuration enables the database to operate asynchronously. Also, due to the reason that PostgreSQl forces to log WAL when inserting, we expand the size of WAL buffer and increase the frequency of checkpoints, which will improve the disk usage.

Redesign Database After providing enough performance to the database, there are surely some necessary optimization to be done in the efficiency of it. In other word, the database should do as less as possible. According to the requirements, we attach several constraints to the database, which will be checked during each insertion. From our analysis, the time complexity will degenerate from $O(n)$ to $O(n \log n)$ for the foreign keys, and there will be extra constant added for the not null constraints. What's more, we also notice that the estimated cost by PostgreSQL is quite inaccurate, for which every check will take $O(n)$ time to perform sequence search and the whole time will degenerate to $O(n^2)$. We expect the database to perform index search and this can be done by changing the option `enable_seqscan` to false, but it will cost fatal errors someday.



```
project1=# EXPLAIN analyze select count(id) from danmu.danmu;
                                                         QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=201444.32..201444.33 rows=1 width=8) (actual time=1894.259..1895.025 rows=1 loops=1)
   ->  Gather  (cost=201443.90..201444.31 rows=4 width=8) (actual time=1894.077..1895.021 rows=5 loops=1)
         Workers Planned: 4
         Workers Launched: 4
         ->  Partial Aggregate  (cost=200443.90..200443.91 rows=1 width=8) (actual time=1884.078..1884.078 rows=1 loops=5)
               ->  Parallel Seq Scan on danmu  (cost=0.00..192642.32 rows=3120632 width=8) (actual time=0.645..1815.209 rows=2495799 loops=5)
 Planning Time: 2.944 ms
 Execution Time: 1896.161 ms
(8 rows)
```

Figure 3: Time analysis

Therefore, we redesign our database under these principles and the detailed modification can be accessed by `DDL.sql`

- Remove not null constraint. Since we have primary key for the two of the main tables, not null constraint is redundant. And for the rest tables who just have a id column, we add barely one not null constraint on the column that we think useful in real situation and in our test of task 4.

- Remove foreign keys. In our design, we will ensure that the input data is valid in the script and that when deleting records, the related records won't be deleted at once, and instead, we will remove them regularly. In this way, we can also improve delete performance.
- Optimize data type. Primitive types have relatively higher performance than the other, e.g., integer and varchar.

Transaction In our depth study of the PostgreSQL, we've learned that it offers a feature to execute and commit a bunch of statements at the same time. To take full advantage of it, we firstly set the autocommit option to false, and in the following steps, we will commit manually at the end of each import method. Also, we apply Batch to the execute of SQL statements. The batch size is set to 1000, considering the speed of parsing csv files and executing statements. In general, in one single method of our import script, we can perform three thread of operations theoretically, that is, the thread of parsing csv files, the thread of executing statements and the thread of writing disk.

Multi-Threading Since the tables have got rid of the limitation of constraints, we can apply multiple threads to the insertion of the tables. In previous step, we have three thread of program and furthermore, we can expand the thread to eight thread, one for each table. The multi-threading feature is well supported by Java and there's something to be noticed. First, a csv file will create more than one table, which means different threads will accessing the same file at the same time. This can be solved by setting the shared lock of file, and since we don't need to modify the file, we can make a copy of the file for every thread. Second, multi-threading will make the record of time hard to do because we hardly know when the threads end. To solve it, Java provides a tool called "CountDownLatch". In the main thread, we just simply create a CountDownLatch and the eight threads, each thread having its own connection of database and able to access the CountDownLatch. When one thread ends, it will call CountDownLatch to decrease by one and the main thread will keep waiting until the CountDownLatch counts to zero. So the main thread is doing the record job. Now, we have eight threads for parsing csv files, eight threads for executing statements and eight threads to write disk, and that's 24 threads in total. The total time cost is determined by the slowest one.

The source code of the enhanced import script can be viewed at `DBMS_Enhanced.java`. The performance information after optimization is shown below and the import result will be given in next section.

CPU Avg Load	Memory Pressure	Disk Load	Workers	Memory Used
4.8	70%	223.6 MB/s	8	128 MB

Table 5: Performance Information

4.5 Comparison

4.5.1 Basic Test Information

- Hardware:
  - **CPU**: Apple M1 (4 Core 3.2 GHz + 4 Core 2.064 GHz)
  - **Memory**: 8GB 4266 MHz LPDDR4X SDRAM
  - **Hard Drive**: APPLE SSD AP0256Q 256GB
- Software:
  - **DBMS**: PostgreSQL 15.4 (Homebrew) on aarch64-apple-darwin23.0.0, compiled by Apple clang version 15.0.0 (clang-1500.0.40.1), 64-bit
  - **Operating System**: macOS 14.0 Sonoma (23A344)
  - **Programming Language**: Java 19.0.1
  - **Development Environment**:
    - JDBC: PostgreSQL 42.2.5
    - CSV: CsvReader 2.0
- Steps:
  1. Follow the steps of the Guidance section
  2. Run the original and enhanced script
  3. Wait until the program finish
  4. Record the data that programs output

4.5.2 Result

Script	**user.csv**	**video.csv**	**danmu.csv**
DBMS.java	5 min 20.835 s	7 hours 2 min 33.578 s	12 min 45.375 s
DBMS_Enhanced.java	3 min 37.093 s	43 min 53.481 s	6 min 4.948 s

Table 6: Comparison of the runtime

Script	**user.csv**	**video.csv**	**danmu.csv**
DBMS.java	18691 records/s	16192 records/s	16304 records/s
DBMS_Enhanced.java	27622 records/s	155890 records/s	34193 records/s

Table 7: Comparison of the speed

We can tell from the table 6 & 7 that for *user.csv* and *danmu.csv*, the cost of time is nearly half of the previous one's and the speeds are doubled. And for *video.csv*, since the original speed is limited by the performance of disk, the enhanced one has a speed of 10 times the original one.

The total time the enhanced version takes is 43 min 53.481 s, compared with 7 hours 20 min of the original one. With the help of multiple threading, the total optimization rate rises amazingly 904 percents which means that the enhanced one is 9 times faster.

5 Performance Comparison

5.1 Introduction

In this section, our aim is to compare the performance differences between DBMS and Java file I/O when handling the same tasks on the same device. The tasks include comparing the performance differences in "adding data," "deleting data," "modifying data," and "searching for data" between the two data management methods.

5.2 Test Environment

In the spirit of reproducible research, we are providing all necessary information for others to replicate our experiment. Here is a list of key details that would be required:

- Hardware:
  - **CPU**:12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
  - **Memory**:16.0 GB (15.7 GB)
  - **Hard Drive**:SAMSUNG MZVL2512HCJQ-00BL7
- Software:
  - **DBMS**:IntelliJ IDEA
  - **Operating System**:Windows11
  - **Programming Language**:Java
  - **Development Environment**:
    - *
    - *
- Test Data:
  - You can download the test data files from [BB URL / github].
  - Data Format/Structure: [data format and structure]The data of newly added users is automatically generated by the random class in Java, and its format is the same as user.csv (except for the header).
- Test Scripts and Programs:
  - The SQL scripts and file I/O programs can be found in the project's repository at [Repository URL].
  - Instructions for running the scripts and programs are available in the repository's README file.

5.3 Test Scripts and Programs

In the design of Java file I O, the functions in the Apache package are used to store every line of information in the data file (excluding the header) into an arraylist. This means that all data files are read into the virtual machine memory, and then modified and rewritten back into the data file. This is also the normal working principle of Java file I O. Calling Apache functions does make the process more concise. But there is a drawback here that when the size of the data file exceeds the size of the virtual machine's memory, it is not possible to read and write data normally.
In the design of Java file I\O, the functions in the Apache package are used to read all the data files into the virtual machine memory, and then modify and rewrite them back into the data files. This is also the normal working principle of Java file I O. Calling Apache functions does make the process more concise. But there is a drawback here that when the size of the data file exceeds the size of the virtual machine's memory, it is not possible to read and write data normally.

5.4 Performance Comparison

We used Java's random classes to generate 1w, 5w, and 20w, respectively. The data files and generated code can be searched on GitHub, and these three sets are used to compare the performance of the two methods.

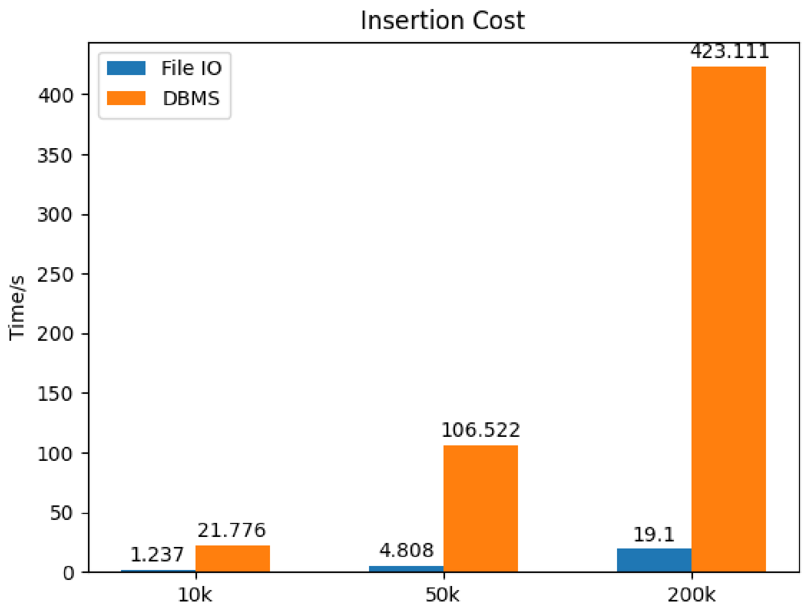


Figure 4: Insertion time comparison

Similarly, when comparing the deleted parts, we also chose to delete 1w, 5w, and 20w pieces of data respectively. Since the time consumption measured by the Java file I\O is almost the same under different data volumes, the average value is taken.

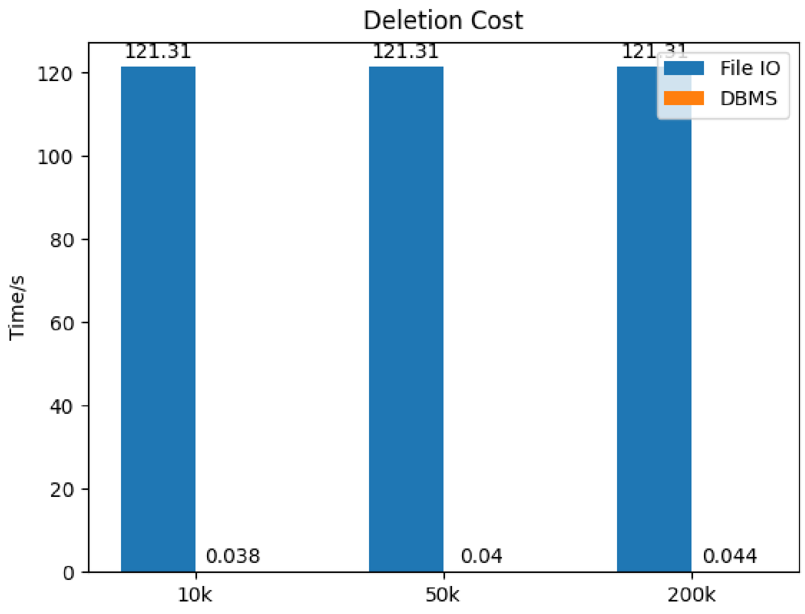


Figure 5: Deletion time comparison

In the update section, we chose to randomly change the user's sign to between 5 and 20 characters in three different data volume states. Similarly, we found that under the method of Java file I\O, the time loss of changing sign is almost similar for different data volumes, so the average value is taken.

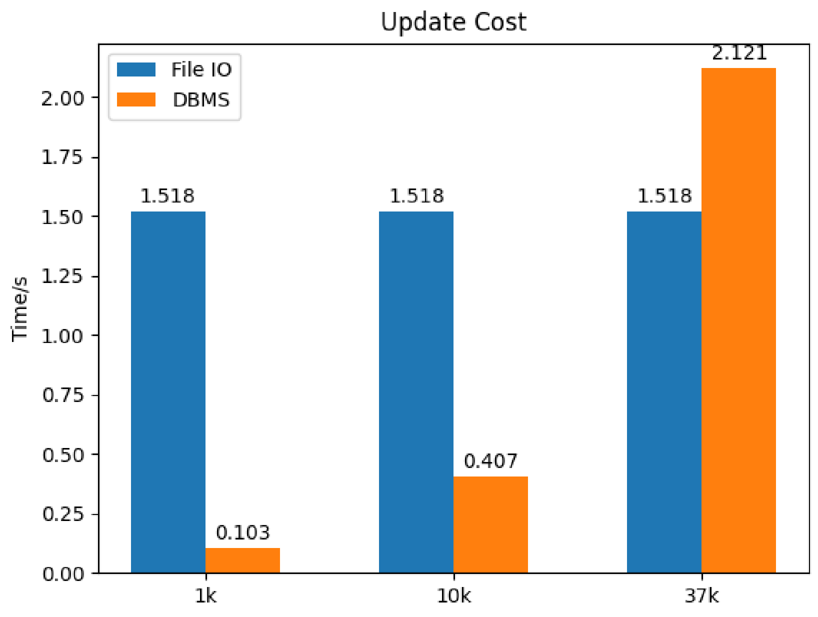


Figure 6: Update time comparison

In the query section,Due to the lack of query function in the Java file I\O method, we chose to directly output the results.

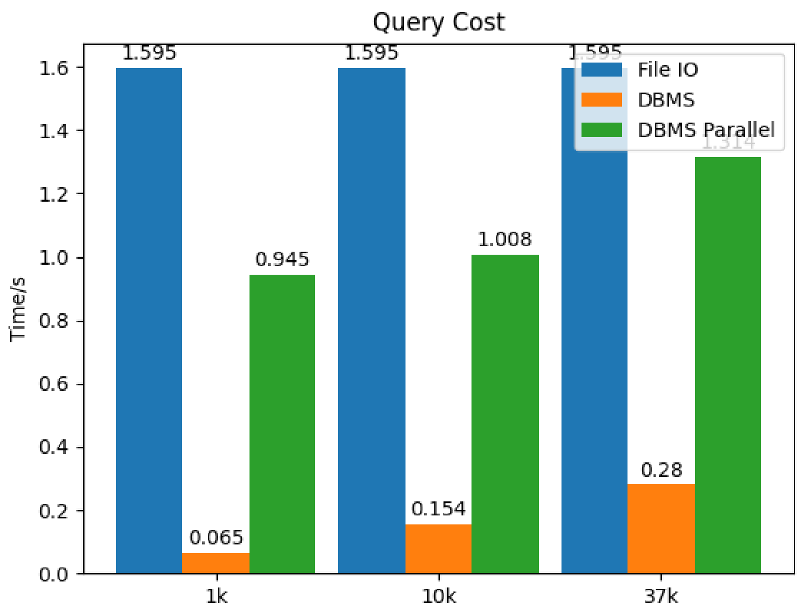


Figure 7: Query time comparison

5.5 Analyze

In the insert section, it can be observed that the time consumption of both methods increases linearly with the growing amount of data. Among these, the DBMS method exhibits a notably higher duration. This is attributed to the decision made during database construction, where we opted not to create three large primary tables. Instead, the data was distributed into several auxiliary tables connected through foreign keys, facilitating more manageable data handling. However, this necessitates the partitioning of data to be inserted into different tables, consequently leading to additional time expenditure.But in Java file I\O,We just need to write new data at the end of the data file.

In the deletion part, the DBMS method allows for the direct removal of all related data through the intermediary of 'mid'. However, under the file I/O method, for the same 'mid', it is necessary to search for and delete other related data associated with the same 'mid'. This search operation can be extremely time-consuming. Therefore, in this aspect, the DBMS method demonstrates a precipitous lead.

In the update section, it is evident that with the DBMS method, the time taken gradually increases as the volume of data grows. In contrast, with the file I/O method, the time remains almost constant under different data volumes. This is due to disk caching issues and the way file I/O operates. In scenarios with varying amounts of data, the file has already been stored once on the disk and, regardless of the data volume, the entire file is read into the virtual machine's cache. The time taken to modify just one attribute is actually very minimal. Therefore, the time spent on updates is quite similar across different data volumes.

Similarly, in the query section, under the file I/O method, it is necessary to read the entire data file into the virtual machine and then output the required data based on demand. Therefore, the time expenditure for the query part remains very close under different query requirements. However, for DBMS, queries involving small data volumes incur very little time loss. Yet, as the volume of queries increases, the time consumption still shows a linear increase. Concurrent queries, which require the opening of multiple(one hundred) threads, tend to take longer than regular DBMS queries.

6 Conclusion

This paper presents a comparative analysis of Java's File I/O and Database Management Systems (DBMS) in managing data, specifically focusing on Create, Read, Update, and Delete (CRUD) operations. We can draw the following conclusion.

6.1 Data Creation

- **File I/O:** Directly appends data at the end of CSV files, which is simple and time-efficient.
- **DBMS:** Requires insertion of data into three primary and five secondary tables, based on the database structure, thus consuming more time.

6.2 Data Deletion

- **File I/O:** Time taken to delete 100, 500, and 10,000 entries is almost identical, possibly due to the file I/O mechanism or disk caching issues.
- **DBMS:** Significantly outperforms File I/O by utilizing foreign key associations for rapid data deletion, unlike the global traversal needed in File I/O.

6.3 Data Update

- **File I/O:** Slower with small data sets, but faster with larger data sets (up to 200,000 records) compared to DBMS.
- **DBMS:** More efficient with smaller data sets but becomes less efficient as data size increases.

6.4 Data Retrieval

- **File I/O:** Lacks a dedicated query function, requires reading the entire file for output, which is not time-consuming.

- **DBMS:** Superior in query performance to File I/O. However, concurrent queries (e.g., 100 threads) take more time than single-threaded queries.

6.5 Time Efficiency

File I/O may be more efficient in handling large volumes of data, but DBMS shows significant advantages in query and deletion operations.

6.6 Data Security

DBMS generally offers higher data security and integrity, including transaction management, rollback capabilities, and concurrency control.

6.7 Multi-threading

DBMS supports multi-threading operations but does not always yield the highest efficiency; File I/O may encounter synchronization and data consistency issues in a multi-threaded environment.

While File I/O might be faster in certain scenarios, such as bulk data processing, DBMS provides a more comprehensive solution for data security, query efficiency, and data consistency. The choice between the two depends on the specific requirements of the application scenario.