Fall 2023 CS307 Project Part I

Contributors:

Topic Design: SUN Kebin

Data Preparation and Documentation: Wang Lishuang, Zhang Haoming, Sun Kebin

Other Contributors: Leng Ziyang, Tang Yulei

Review: WANG Weiyu

This project description is extended from the one of Fall 2022 CS307.

General Requirement

- A group project with only **2 ~ 3** teammates who are in the **same lab session**. Each group should finish the project independently and submit **only one report**. When a group submits several reports by different members, we will choose one randomly. When a group submits several reports by only one member, we will judge the latest one.
- The teammate(s) you select for Project I will also be your teammate(s) for Project II. It is not allowed to change teammates once paired.
- You should submit the report before the deadline. **All late submissions will receive a score of zero**.
- **DO NOT copy ANY sentences and figures** from the Internet and your classmates. Plagiarism is strictly prohibited.
- The number of pages for your report should be **between 8 and 20**. Reports less than 8 pages will receive a penalty in score, however, ones with more than 20 pages will NOT earn you a higher score.

Database management systems (DBMS) are extremely useful when we are going to deal with enough data. It can help us manage them in a convenient manner and improve the efficiency of both retrieval and modification. Hence, your work on Project I is composed of the following parts:

1. Find the inherent relationships of the provided data and then design an E-R diagram based on the relationships you found.
2. Design a relational database using PostgreSQL according to the provided data file and your E-R diagram.

3. Import all data into the database.

4. Compare the performances of data retrieval and modification between database and raw file I/O in a programming language. **The ONLY allowed programming language is *Java*.**[1]

Section 1 Background

This project is about the structure of a fictional Danmu video website – Synchronized User-generated Subtitle Technology Company (SUSTC). After decades of development, it has built an ecosystem around users and creators who continuously produce high-quality content including videos and Danmu comments.

Usually, a video is committed by users known as "Uploaders." Then, the videos will be sent for review to check if it conforms to standards by users known as "Reviewers." After a successful review, the Uploader can publish this video at their designated time. The users can then watch the video, during which they can interact with each other through Danmu beyond time and space. Of course, except for sending Danmu messages, the users can also do a series of operations, such as giving coins, liking videos, making videos their favorites.

1.1 Data Description

There are three provided data files: *user.csv*, *videos.csv*, *danmu.csv*.

1.1.1 *user.csv*

- *mid*: the unique identification number for the user.
- *name*: the name created by the user.
- *sex*: include but not limited to biological sex.
- *birthday*: the birthday of the 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.

[1] It is not unacceptable to finish THIS project in other common programming language. Yet, since the project II of this course will involve automatic benchmarking, you will have to translate your code to *Java* by then.

- *identity*: a value in {*"user", "superuser"*} indicating the user's role.

1.1.2 *videos.csv*

- *BV*: the unique identification string of a video.
- *title*: the name of video created by video owner.
- *owner mid*: the *mid* of the video 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 holding the *mids* of the users who liked this video.
- *coin*: a list holding the *mids* of the users who have given this video their coins.
- *favorite*: a list holding the *mids* of the users who favorited this video.
- *view*: a list holding the users who watched this video and their last watch time duration（最后一次观看所停留在的视频时间）.

1.1.3 *danmu.csv*

- *BV*: the *BV* of the video that 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.

Section 2 Report Structure and Task Requirements

2.1 Basic Information of Group and Workloads

You need to write down the **membership information**, specific **contribution content** and the **percentage** of each team member.

2.2 Task 1: E-R Diagram (15%)

Make an E-R Diagram of your database design with any diagram software. Hand-drawn results will NOT be accepted. Please follow the standards of E-R diagrams. In the report, you must provide a snapshot or an embedded vector graphics of the E-R diagram. Also, please specify the name of the software/online service you used for drawing the diagram.[2]

2.3 Task 2: Database Design (25%)

Design the tables and columns based on the background supplied above. First, you shall generate the database diagram via the "Show Visualization" feature of *DataGrip* and embed a snapshot or a vector graphics into your report. Then, briefly describe the design of the tables and columns including (but not limited to) the meanings of tables and columns.

In addition, please submit an SQL file as an attachment that contains the DDLs (create table statements) for all the tables you created. Please make it into a separate file but do not copy and paste the statements into the report.

Some notices:

1. Design a database by **PostgreSQL** allowing us to manage all the information mentioned above.
2. Your design needs to follow the requirements of the three normal forms.
3. Use primary key and foreign keys to indicate important attributes and relationships about your data.
4. Every row on each table should be uniquely identified by its primary key. (You may use simple or composite primary key).
5. Every table should be included in a link. No isolated tables included. (每个表要有外键，或者有 其他表的外键指向)
6. Your design should contain no circular links (对于表之间的外键方向，不能有环)
7. Each table should always have at least one mandatory ("Not Null") column (including the primary key but not the id column).
8. Use appropriate types for different fields of data.
9. Your design is as easy to expand as possible. (Especially for a three-person group)

2.4 Task 3: Data Import (28% + 3%)

In this task, you should **write a script** to import the Data into the database you designed. After importing the data, you should also make sure

[2] It does not matter what E-R diagram method you choose, label the essential elements clearly and completely.

all data is successfully imported.

In the report, you are required to accomplish the **basic requirements** (18% out of 28%):

1. Write a script to import the data file.
2. Write a description in your report of how your script import data. You should clearly state the steps, necessary prerequisites, and cautions to run it and import data correctly. Show the number of records in each Entity table. (Especially for a three-person group)

You may also finish the following **advanced requirements** to get the remaining points (10% out of 28%) (Three-person group should do better):

1. Find more than one way to import data and provide a comparative analysis of the computational efficiencies between these ways.
2. Try to optimize your script. Describe how you optimized it and analyze how fast it is compared with your original script.

For the advanced tasks, please make sure to describe your test environment, procedures, and actual time costs. You are required to write a paragraph or two to analyze the experiment results. You may refer to the requirements for reporting experimental results in Task 4 for details.

When you do in-depth exploration outside of the Lab in data import, and get exceptionally good efficiency, you can earn the **bonus** (1%~3%).

2.5 Task 4: Compare DBMS with File I/O (30% + 7%)

In the report, you are required to finish the following **basic requirements** (20% out of 30%):

1. A description of your test environment, including (but not limited to):
   a) Hardware specification, including the CPU model, size of memory.
   b) Software specification, including the version of your DBMS and operating system, the programming language you choose, and the development environment (the version of the language, the specific version of the compilers and libraries, etc.).
   c) When reporting on the environment, you can think about this question: if someone else is going to replicate your experiment, what necessary information should be provided for him/her?
2. A specification of how you organize the test data in the DBMS and the data file, including how do you generate the test SQL statements and what data format / structure of the files are.
3. A description of your test SQL script and the source code of your program. DO NOT copy and paste the entire script and the program in the report. Instead, please submit source codes as attachments.
4. A comparative study of the running time for the corresponding statements / operations. You are encouraged to use data visualization to

present the results. Be sure to use consistent style for graphics, tables, etc. Aside from a list / figure of the running times, you are required to describe the major differences with respect to running performance, what you find interesting in the results, what insights you may show to other people in the experiments, etc.

In addition to the basic requirements, you can also think about some of the following **advanced tasks** (but not limited to the following ones) to challenge yourself and get the remaining points. (10% out of 30%) (Three-person group should do better in this part)

1. Can your database deal with high concurrency? You may try to achieve the above benchmarks in a higher order of magnitude, such as hundreds of thousands of selections.
2. Can you compare the performance with different database software (e.g., MySQL, MariaDB, SQLite), file systems, disk performance and type, programming languages, libraries, or operating systems?

When you do in-depth exploration outside of the Lab in manipulate database, and get exceptionally good efficiency, you can earn the **bonus** (1%~7%).

Section 3 Submit (2%)

Submit a report named "*Report_sid1_sid2_sid3.pdf*" in **PDF format** and **a .zip archive** of necessary attachments (such as SQL scripts and source code files) on the Blackboard website before **23:00 on November 5th 2023**, Beijing Time (UTC+8). For the report, replace the *sid1/sid2/sid3* with your members' student ID. For attachments, please put them into separate directories based on the task and compress them into a .zip archive.

Section 4 Disclaimer

The characters, businesses, and events in the background of this project are purely fictional. The items in the files are randomly generated fake data. Any resemblance to actual events, entities or persons is entirely coincidental and should not be interpreted as views or implications of the teaching group of CS307.

Appendix A

When comparing the performance of data retrieval and manipulation between database APIs and file APIs, we advise you to conduct the comparative analysis according to the following steps:

1. Benchmarking with database APIs: Based on the database you created, you are required to write a program in Java that accesses the database via database APIs and contains a series of *INSERT*, *DELETE*, *UPDATE*, and *SELECT* statements. You may specify the number of statements in each type on your own and decide which data to be modified and read. However, the operations of each statement type cannot be too small to fail illustrating the strength and weakness of database APIs over file I/O (depending on the programming language, tens of thousands of records each would usually be enough). Finally, you need to record the running time of each statement type or each statement. Here, we provide some typical test descriptions in the database you can refer to:
   a) *INSERT*: First, randomly drop out some rows of the csv file and import it into your database. Then evaluate the time cost of importing rest of the rows of the csv file.
   b) *DELETE*: First, import all data of the csv file into your database. Then, evaluate the time cost of deleting arbitrarily chosen rows of your database's delivery record table.
   c) *UPDATE*: First, import all data of the csv file into your database. Then, evaluate the time cost of updating all the *EMPTY* values of your database's delivery record table to arbitrary values.
   d) *SELECT*: First, import all data of the csv file into your database. Then, evaluate the time cost of finding all delivery records that have not finished or finding all the delivery records that had been packed by an arbitrary container, etc. You may pay more attention to *SELECT* statement tests since they are used more frequently than other ones in many real-world scenarios.
2. Benchmarking with file APIs: This step is designed to replicate all your operations in the first step, but via a generic programming language's standard file APIs. First, create file(s) that store(s) the same data as you have in the tables in the DBMS. Then, write a program to insert, delete, update, and find the data items as you did in the SQL statements and queries. Be sure that the file operations (and the number of operations)

are identical to the SQL operations (and the number of the statements). Finally, record the running time of each operation (type) as in database API benchmarks.

3. Comparative analysis: Compare the recorded running time of the same operation/statement from the DBMS and the file, respectively. You may conduct comparisons from multiple levels, such as comparing statements with corresponding operations (statement-level) or comparing the total time of all statements in a specific type with the corresponding operation type (type-level).

Appendix B

Some notes on how to finish this project in a better way:

1. You can perform the above benchmarks with different orders of magnitude of statements/operations (e.g., from hundreds to thousands to ten(s) of thousand(s)).
2. You can choose or design any format you want to store data in the file, such as plain text formats (CSV, JSON, XML, etc.) or a self-defined binary format.
3. Please only stick to standard file APIs, i.e., the java.io in Java. The only exception is that if you choose to use JSON and XML, you may utilize third-party JSON/XML libraries if the standard library does not provide it, e.g., *Gson*.[3]
4. We acknowledge that there are numerous libraries that can facilitate data manipulation works or even speed up the performance of insertions and selections significantly (e.g., pandas in Python). You are encouraged to also compare the performance of these libraries with DBMS. However, you should conduct the analysis of DBMS vs. standard file APIs beforehand.
5. Some useful resources:
   a) Advantages of Database Management System over file system
   b) Advantages of Database Management System
   c) Characteristics and benefits of a database

[3] If you insist on using another language, this rule still holds. For example, you shall stick to iostream and *fstream* in C/C++, the file object in Python, or the *System.IO* in C#. For JSON and XML, you may utilize *json* package for Python for instance. As for languages like C# whose standard library provides *System.Text.Json* and *System.Xml*, you shall not use replacements.