Fall 2022 CS307 Project Part II

Contributors:

Leader and Overall Design: SUN Kebin

Data Preparation: WANG Lishuang, Li Boyan

Documentation: WANG Lishuang, Lu Diyun, SUN Kebin

Other Contributors: ZHANG Liyu, LENG Ziyang

Reviewer: WANG Weiyu

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

General Requirement

- This is a group project with same group members as Project I. Each group should finish the project independently and submit only one report. It is NOT allowed to change teammates grouped in Project I.
- You should submit the report and the source code 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 better score.

After finishing Project I, it is not difficult for you to get an overview about the advantages of database management system (DBMS) against ordinary file I/O, especially for their performances. In this project, you are required to continue the works done in Project I and finish the following tasks:

1. Design a new database with permission management based on the one you finished in Project I to meet the new requirements presented by the Shipments across Urbans and Seas Through Containers (SUSTC).
2. Correctly implement the APIs described below. They will be used to communicate with your database including operations requested by different roles in SUSTC. Since the APIs are defined in `Java`, no other programming language is allowed.
3. Other advanced tasks described below.

Section 1 Background

This project is based on further requirements for a fictional international shipping organization named SUSTC since the department managers are not exactly happy about Project I: the lack of permission management and unified accessing points makes it hard for all the staffs to interact with the database. Therefore, in this project, your main goal is to redesign a database for SUSTC with permission control and implement the required APIs for them to manage the whole organization and provide functionalities for all the staffs and registered entities of SUSTC.

As a reference in your design and implementation process, we provide the authorizations of the staffs and entities which will log in SUSTC, as well as data samples which can be used to test your database.

1.1 Entities of SUSTC

Courier: They can only query and update the pickup/delivery status of his/hers. They can also query and update personal information. (No mandatory API for personal information updating.)

Company manager: They can designate appropriate ship(s) for shipping items awaiting at the seaport. Of course, they can query and update the companies' information. (No mandatory API for company information updating.)

Seaport officer: They can finish the items' import/export checking and update their status (include reporting check failure). They can also query about the items currently at the seaports that they are working for.

Department manager of SUSTC: They can query all information. However, they have no permission to write to database directly.

1.2 Data Description

Like Project I, the given **`records.csv`** file includes 50,000 records for testing purpose. Furthermore, the given **`staffs.csv`** file includes the information of all couriers, company managers, seaport officers and SUSTC department managers.

Here is the explanation of the columns of the **`records.csv`** file:

- Item Name: the unique name of the shipped item.
- Item Class: the type (or class) name of the corresponding item.
- Item Price: the actual price of the corresponding item.
- Retrieval City: the delivery start city that the retrieval courier picks up the corresponding item. This is also the working city of "Retrieval Courier".
- Retrieval Courier: the retrieval courier's name who picks up the item.
- Delivery City: the target city's name of the delivery. This is also the working city of "Retrieval Courier".

- Delivery Courier: the name of the courier who delivers the item. Can be EMPTY if the item has not arrived at the target city.
- Export City: the export city's name of the item of this record.
- Import City: the import city's name of the item of this record.
- Export Tax: the export tax of the item of this record.
- Import Tax: the import tax of the item of this record.
- Export Officer: the checking officer of the export seaport. Will be EMPTY if it has not been exported.
- Import Officer: the checking officer of the import seaport. Will be EMPTY if it has not been imported.
- Container Code: the unique code to identify the container that is used to store the item. Will be EMPTY if the item is not being packed or had been packed.
- Container Type: the type of the container that is used to store the item. Will be EMPTY if "Container Code" is EMPTY.
- Ship Name: the unique name to identify the ship that is used to ship the item. Will be EMPTY if "Item State" has not reached "Shipping".
- Company Name: the name of the company that manages this shipment. The retrieval and delivery couriers and the ship are all managed by it.
- Item State: the current state of this record. See Figure 1 for more information.

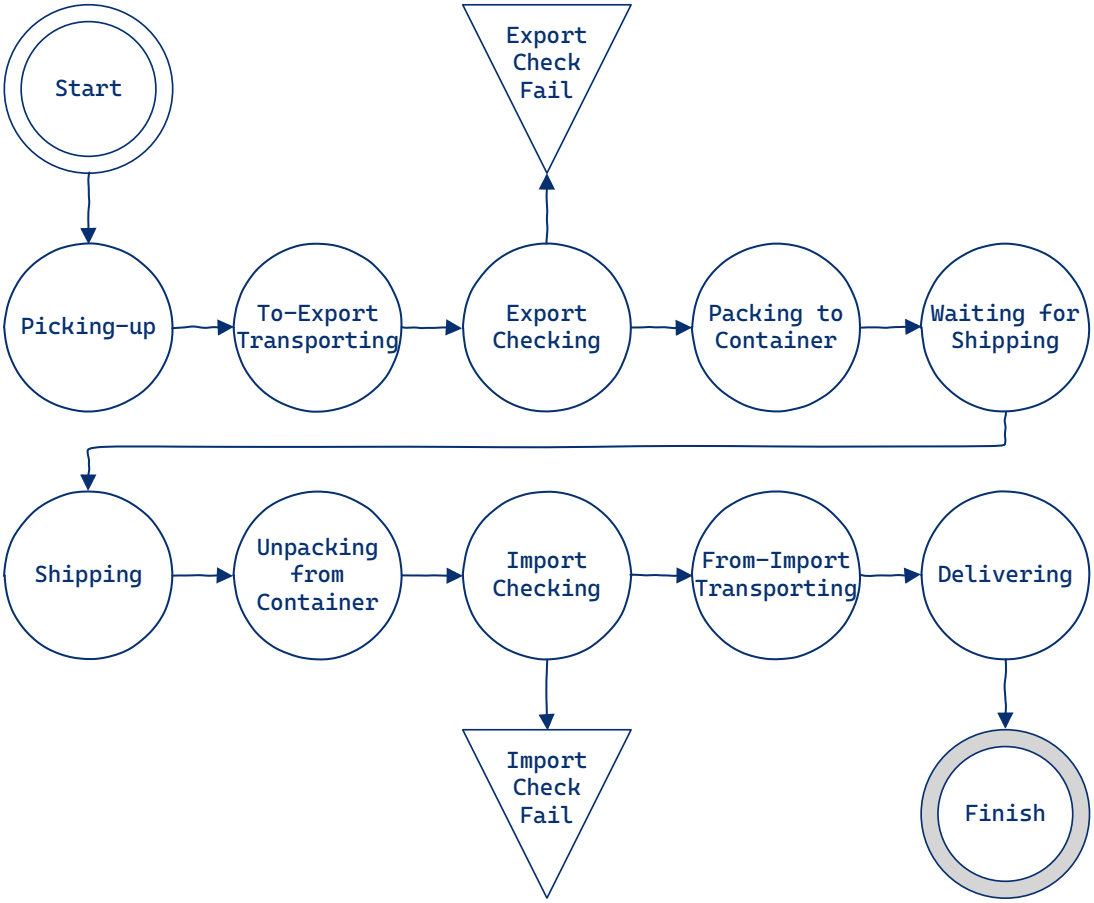

Figure 1 The state flow of an item.

Here is the explanation of the columns of the `staffs.csv` file:

- Name: the unique name of the staff.
- Type: the authorization type of this staff, can be Courier, Company Manager, Seaport Officer, SUSTC Department Manager.
- Company: the company name that this staff works for. Will be EMPTY if "Type" is not "Courier" nor "Company Manager".
- City: the working city of the staff. Will be EMPTY if "Type" is not "Courier" nor "Seaport Officer".
- Gender: the gender of the staff.
- Age: the age of the staff
- Phone: the phone number of the staff.
- Password: the password of the staff.

1.3 Notices

If an item failed to pass the export/import checks, none of the proceeding procedures can be done and the corresponding record properties shall always be EMPTY. For simplicity, SUSTC stops tracking such items once they failed.

Please do not forget that the records have states to maintain now. Also, remember that the permission controls are mandatory, and the database user management is required. Therefore, it is not allowed to use the root user except table creation, import and deletion. However, it is allowed to create only one database user for one type of actual user for simplicity.

Also, compared to Project I, the time-related columns in `records.csv` of Project II are all removed for simplicity and all the couriers now bind to retrieval and delivery cities rather than import and export cites.

1.4 Important Notice

To make sure that your work is not in vain, your report must contain the basic information of your group and workloads:

1. Names, student IDs, and the lab session of the group members.
2. The contributions and the percentages of contributions for each group member. Please clearly state which part(s) of the task(s) is/are done by which member in the group.
3. If you failed to link a task or its sub-tasks to one of the group members, we will NOT count the score for the part you miss (since we do not know who accomplished this task).

Section 2 Tasks

2.1 Database Design (15%)

As mentioned above, the database requirements are changed by a few. Please make a new E-R diagram of the new database you design based on the one in Project I (5% out of 15%). Then, you shall generate the database diagram via the "Show Visualization" feature of `DataGrip` and embed a snapshot or a vector graphics into your report (5% out of 15%). You are also required to submit the database user creation and permission granting descriptions (5% out of 15%).

2.2 Basic API Specification (70%)

To provide basic functionality of accessing a database system, you are required to implement a backend program using `Java` exposing a set of application programming interfaces (APIs). The detailed specification for each API is described below.

Note that the APIs are defined in a series of `Java` interfaces. Hence, you are not allowed to define your own API instead of predefined ones nor allowed to use any programming language other than `Java`. Please refer to the actual `Java` interface file if there were any conflict between this description and the provided `Java` interfaces. If not stated particularly, all the input parameters are IMMUTABLE, and no exceptions shall be thrown by the APIs.

Furthermore, it is NOT permitted to use file I/O to manipulate the data provided by the APIs. You MUST use `JDBC` to interact with the database to manipulate the provided data.

2.2.1 Table Manipulation

The following API requires root/super user to execute.

1. `Constructor(String database, String root, String pass);`
   The constructor of your implementation class that will be invoked at first by the testing program only for once. The `database` is the provided database address and port, the `root` is the root username, and the `pass` is the root user password. The database shall be created during the construction process. During the final tests, if any exception were thrown, the created database would be deleted automatically.
2. `void import(String recordsCSV, String staffsCSV);`
   Import a large amount of item shipping records as well as SUSTC managers, company managers, seaport officers and couriers all at once from the given CSV files' content. All the corresponding tables, such as courier and city, shall be updated as well. Since the provided data will be legal, this API has no return.

2.2.2 SUSTC Department Manager User

Except for the APIs above that do not require login information and are executed in root user priority, rest of the APIs CANNOT use root user. We will conduct tests to check whether you violate this constrain. Hence, all the following APIs requires a `LogInfo` input parameter which contains the login user's name, type, and password. If the login user does not exist or has wrong password, the APIs with `boolean` returns shall return `false`, the APIs with `object` returns shall return `null`, and other APIs shall return `-1`. All APIs with `boolean` returns shall return `true` if succeed. Note that `boolean` returns are replaced by `bool` in the following API descriptions to fit in the lines.

The following API requires SUSTC department manager permission to execute.

3. `int getCompanyCount(LogInfo log);`
   Look for the number of companies that joined SUSTC. No special return other than the ones stated above.
4. `int getCityCount(LogInfo log);`
   Look for the number of cities logged in SUSTC. No special return other than the ones stated above.
5. `int getCourierCount(LogInfo log);`
   Look for the number of couriers who joined SUSTC. No special return other than the ones stated above.
6. `int getShipCount(LogInfo log);`
   Look for the number of ships logged in SUSTC. No special return other than the ones stated above.
7. `ItemInfo getItemInfo(LogInfo log, String name);`
   Look for the item's full information according to its `name`. If `name` does not exist, returns `null`.
8. `ShipInfo getShipInfo(LogInfo log, String name);`
   Look for the ship's full information according to its `name`. If `name` does not exist, returns `null`.
9. `ContainerInfo getContainerInfo(LogInfo log, String code);`
   Look for the container's full information according to its `code`. If `code` does not exist, returns `null`.
10. `StaffInfo getStaffInfo(LogInfo log, String name);`
    Look for the staff's full information according to his/her `name`. If `name` does not exist, returns `null`.

2.2.3 Courier User

The following API requires company courier permission to execute.

11. `bool newItem(LogInfo log, ItemInfo item);`
    Add a new item that is currently being picked up. The `item` shall contain all necessary information as stated in Section 1.2. Returns `false` if `item` already exist or contains illegal information.

12. `bool setItemState(LogInfo log, String name, ItemState s);`
    Set the item's current state to the given state with given item `name`. A retrieval courier can only set an item of his/her correspondence to state as far as "Export Checking" while a delivery courier can only set an item of his/her correspondence to state as far as "Finish". Returns `false` if `item` already exist or `s` is illegal. Note that any courier working at the "Delivery City" of given item `name` can set "Item State" from "From-Import Transporting" to "From-Import Transporting" when its "Delivery Courier" is empty, meaning that he/she is going to deliver this item.

2.2.4 Company Manager User

The following API requires company manager permission to execute.

13. `double getImportTaxRate(LogInfo log, String city, String itemClass);`
    Look for the import tax rate of given `city` and `item class`. Return `-1` if any of the two names does not exist or `city` is not seaport city.
14. `double getExportTaxRate(LogInfo log, String city, String itemClass);`
    Look for the export tax rate of given `city` and `item class`. Return `-1` if any of the two names does not exist or `city` is not seaport city.
15. `bool loadItemToContainer(LogInfo log, String itemName, String containerCode);`
    Notify to the database that the given `item` is going to be packed into the given `container`. Returns `false` if the `item` cannot be loaded to the container (due to reasons like already loaded and illegal item state) or `container` is full or being shipped. For simplicity, one container can only pack one item. Note that only item that passed export checking (at "Packing to Container" state) can be loaded to container.
16. `bool loadContainerToShip(LogInfo log, String shipName, String containerCode);`
    Notify to the database that the given `container` is going to be loaded to the given `ship`. Returns `false` if the `container` cannot be loaded to the ship (due to reasons like already loaded) or `ship` is currently sailing. For simplicity, one ship can transport unlimited number of containers.
17. `bool shipStartSailing(LogInfo log, String shipName);`
    Notify to the database that the given `ship` is currently sailing with loaded containers. Returns `false` if the `ship` is already sailing.
18. `bool unloadItem(LogInfo log, String item);`
    Notify to the database that the given `item` is being unloaded from its container and ship. This API also implies that the corresponding container is being unloaded to "Import City" of the item and the ship is currently docking at "Import City". Returns false if the `item` is in illegal state.

19. `bool itemWaitForChecking(LogInfo log, String item);`
    Notify to the database that the given **item** is waiting at its "Import City" for import checking. Returns false if the **item** is in illegal state.

2.2.5 Seaport Officer User

The following API requires seaport officer permission to execute.

20. `String[] getAllItemsAtPort(LogInfo log);`
    Look for all the items that is currently waiting at this officer's working seaport. Returns an array of **String** containing the item names. No special return other than the ones stated above.
21. `bool setItemCheckState(LogInfo log, String itemName, bool success);`
    Set whether the **item**'s checking state is **success** or not. Returns **false** if **item** does not exist or is in illegal state.

2.3 Advanced APIs and Other Requirements (15%)

Based on the APIs defined above, you can easily implement a complete backend/server system that receives and deals with the requests from the frontend. It is not mandatory that you implement a GUI frontend, however, it must include the login, logout, permission control and other basic functionalities specified above. You are free to use any programming language(s) with any network communication protocol(s) to implement the frontend and the backend after finishing the basic requirements in **Java**. Also, you can define and implement other APIs that you think necessary for your client-server system.

Section 3 How to Test Your Program

First, implement the given **Java** interfaces in the provided **cs307.project2.interfaces.jar** JAR file (whose source code is also provided). During this process, you are encouraged to reuse or encapsulate your codes in Project I. Then, import your **Java** project to the given **cs307.project2.tests** project (which will be provided later) and run the **jUnit** tests with the database and root user set to your local ones. Finally, check the test results (and console outputs) for more information about correctness, running time, etc.

Note that during final tests, the provided version of **cs307.project2.tests** will not be used and thus you shall not upload it. Also, the performance will be tested: API(s) with running time(s) far larger than the baseline implementation(s) (which is not particularly high performant) will be terminated early and marked as fail. Note that the final tests can be conducted on any OS with your submitted source code, therefore, please do not submit any file other than **.java** and text-based configuration files.

Section 4 How to Submit

Submit the report in PDF format and the source code with necessary attachments (such as the database user creation and permission granting descriptions) on the Sakai website before 23:00 on 18th December 2022, Beijing Time (UTC+8). For all source code and attachments, please put them into separate directories based on their type and compress them into a `.zip` archive.

Section 5 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.