COMP2411 G13 Project Report

HAN Wenyu 21097519D
LIU Zixian 19084087D
ZHOU Siyu 21094655D
HUANG Sizhe 20084413D
JIAO Shouyuan 20074803D
JIA Shangdong 20099629D

Introduction

Nowadays, not only in schools but also in public communities, the library plays a more and more important role in the study and work of the public. Therefore, under the background of the information age, scientific and efficient library management can effectively improve the working mode of the library, improve the efficiency of learning, and also inject power into the development of schools and society.

In this project, our group members designed a simple library management system to construct databases in the library system and to fit the demand of efficient management for the user-friendly purpose.

Designing Process

- ## Stage 1: Draw ER Diagram For Building Database (Test Data)

  Before designing codes, we first drew an ER diagram together. With the diagram, we clarified the basic logic structure, relationships between different entities, and the general design direction. Besides, we also build a schema to provide more detailed and clear connections between different entities and help us better prepare for writing SQL queries and codes.

- ## Stage 2: Writing Test Data and SQL Query for Different Function Implementation in Java Programming Codes

  We choose Java and SQL learned in class and combine them, and then we will use the Linux environment to run the Java codes. Then, following the ER diagram and schema we built before, we construct the test data for different entities and M:N relationships. The SQL query, based on test data, was written later, following the relationships and interfaces we will build later in the Java program.

- **Stage 3: Coding in Java to Write Different Interfaces and Logical Connection**

  According to the relationships in the ER diagram, we determine the main interfaces in this system: log in, user interface, manager interface, search, record, notice, report, reserve/borrow/desire books, cancel reservation, and deactivate/activate account. Then, we designed the format and contents shown in every interface to be clear and easy to follow for users.

  As for the logical connection part, we use different methods created in Java to build connections between different interfaces. What's more, to realize the interaction between the user and the system, we simplify the input made by the user for the purpose of making it more user-friendly.

  After these 3 stages, by establishing connections between SQL queries, Java codes, and Oracle, we formed the library management system. In order to test the normal operation of the system, we use the test data we have already created to prove it.

Fore End Explanation

- **Original Log in Interface: originFace()**

  As the first interface, the user or manager needs to type in their ID and password to log in to their account. If their entered ID does not exist in the database, then the system will ask you to enter your ID again. As for the password, if there are any problems, the system will notify the user or the manager to enter again. When a user account is recognized as a deactivated account, a notice will directly appear on the interface: "Your account is deactivated!" Please contact the AR department!

  *Notes: Due to the user ID and manager ID being saved in different SQL tables, we don't need the user or manager to choose their identity in order to let the user or manager process our log-in stage much easier.

- User Main Interface: userFace()

When a user enters the dashboard, the system will clearly show four main functions with rectangles outside that allow users to easily follow. We wrote “enter 1 for searching books”, “enter 2 for checking records”, “enter 3 for reading notice”and “enter 4 for return books”, “enter 5 for cancel reverse”,“enter quit for quitting”, so that they only need to type a number, or quit to process what the user wants to do. If they enter a number that is not within the range set by the system, they will also be reminded that the entry is illegal and to re-enter it.

*Notes: These terse-style designs for the interface will help users to use the library system more conveniently, in order to let the user process our choosing function stage much more easily.

- Manager Main Interface: managerFace()

Similarly to the user main interface, we also show functions with rectangles as we explained before, but we add one more report function to managers. For choosing a function, we still ask the manager to enter a number. As for the additional report functions, we have “enter 6 for generating reports ”, “enter 7 for deactivating users”, “enter 8 for activating users” to complete it.

*Notes: These terse-style designs for the interface will help managers to administrate the library management system more conveniently, which allows managers to process our choosing function stage much easier and raise working efficiency.

- Search Books Interface: searchFace()

When a user or manager wants to search for the book they desire, they will come into the book search interface. We choose four attributes of the book entity: they are book ID, name of the book, author of the book, and category.

Considering the user may not know all of the information about the book, they can follow the introduction on the interface: “Please input information of the book you desire below. If you don't know few information, please enter '/' ”, to enter the information they know.

When the book is available, the actions users want to perform can be done with only simple numerical input. Type 1 for borrow now, type 2 for reverse book and should come to the library to collect it in 3 days. After the operation has been performed correctly, the successful borrow or reverse action will also appear on the interface.

When the book is unavailable, a visual display of the book's current unavailability and a return to the main interface operation with the returned input to search for other books appear on the screen. Users also can choose the books they wish to borrow in the future, only simple numbers need to be entered to operate again. Meanwhile, a notice on the interface directly tells the user that they will be notified when the reserved books are available.

*Note: Based on the feature of the library management system that is user-friendly, the simple reminder on the top of the screen, which can lead the user to do their research as soon as possible, and get the book information they want in time.

- Patron’s Record Interface: recordFace()

When a user or manager wants to check their action record in the library, they will come into the book record interface. Then, the system will immediately present the results. There are three kinds of records: borrow, reserve, and desire. Various record lists will be shown separately under the record titles. Reverse cancellations can also be performed by following the prompts on the screen and displaying the result after a successful cancellation.

*Note: In order to save user and manager time, all of the records will show on the interface directly without needing any other operation, such as choosing a number. Simple and quick information presented on the terminal can promote users' experiences.

- Notice Information Interface: noticeFace()

When a user or manager wants to know about the notice information related to their account, the status of desired books and borrow deadlines, they will enter in the notice board. Then, the system will immediately present the results.

*Note: For deadline notice, this kind of information can help users to remind them that their borrowed book should be returned on time, to keep the account active. Realization of the availability notice of the desired book, it will assist users in obtaining the book's availability on time.

- Report (Only For Manager) Interface: reportFace()

To enable managers to administer the library more efficiently, they can enter the report interface to acquire reports of popular book records and deactivated accounts.

In order to lead managers to gain a better understanding of which books are more in demand by users, the report shows how many times each book has been borrowed, desired and reserved, which will be presented by numbers following the book ID.

As for deactivated records, the number of accounts and the deactivated ID will be clearly shown.

*Note: Intuitive data reporting feedback allows managers to get clear feedback on book requirements and deactivated accounts, which is very user-friendly for their productivity. Overall, reports' information will be beneficial to the further development of the library.

- Deactive Account Interface(Only for Manager): deactivateFace()

To enable managers to manage the patron's status, the deactivation function of the patron's account was designed in this management system. They can enter the deactivate account interface to ask the system to realize the deactivation function.

Firstly, the manager enters the patron's account id, and then we find the account in the database, then, we check if the entered one matches any ID in the database. If the entered account is incorrect, this interface will ask you to enter the Patron's account ID again. If correct, the interface will update the status information in the system.

*Note: For the deactivation function, the system only needs the manager to enter the patron's account. It only needs a few times, when this action is realized, then the

system will react directly. Overall, this deactivation function simplifies the process of finding and editing in the database by manager.

- ### Activate Account: activateFace()

  The activation feature of the patron's account was also created here to allow managers to control the patron's status. They can ask the system to implement the deactivation function by going to the activate account interface.

  The manager enters the customer's account ID, we locate the account in the database, and then we determine whether the account ID entered matches any deactivated IDs in the database. This screen will prompt you to enter the Patron's account ID again if the entered account is invalid. If accurate, the interface will update the system's status information.

  *Note: The system only requires the manager to enter the patron's account for the activation function. Overall, this deactivation feature streamlines only the manager's search and editing processes in the database.

Back End Explanation

SQL Test Data

We prepare several tables for entities and relationships, and insert records to each table.

```
DROP TABLE BOOK;
DROP TABLE E_BOOK;
DROP TABLE STAT;
DROP TABLE PUBLISHER;
DROP TABLE BORROW_ORDER;
DROP TABLE RESERVE_ORDER;
DROP TABLE PATRON_ACCOUNT;
DROP TABLE REPORT;
DROP TABLE MANAGER;
DROP TABLE DESIRE;
DROP TABLE BOOK_RECORD;
```

Table BOOK:

Record information of each book, the primary key is BOOK_ID.

Table E-BOOK:

Record information of each e-book, the primary key is E_BOOK_ID

Table STAT:

Record status of books, the primary key and foreign key is BOOK_ID.

Table PUBLISHER:

Record information of each publisher, the primary key is BOOK_ID.

Table BORROW_ORDER:

B-ORDER_ID is generated by default, two foreign keys are BOOK_ID and ACCOUNT_ID.

Table RESERVE_ORDER:

R-ORDER_ID is generated by default, two foreign keys are BOOK_ID and ACCOUNT_ID

Table PATRON'S ACCOUNT:

Record information of each user account, primary key is ACCOUNT_ID.

Table REPORT:

Record the reports' details, the primary key is R_ID.

Table MANAGER:

Record information of each manager account, primary key is M_ID.

Table DESIRE:

Record information of desired actions, both of attributes, BOOK_ID, and ACCOUNT_ID are foreign keys.

Table BORROW RECORD:

Record information of all borrowed book records, foreign key is BOOK_ID.

- ### SQL Data abstraction (data type)

  We have used several data types in our program.

  The first one is the VARCHAR. We have used the type, *VARCHAR, which* is used to store character string, and maximum of the set length specified. For example, we use *VARCHAR* to store *ACCOUNT_ID*, *ACCOUNT_PASSWORD*, *EMAIL*, *IS_DEACTIVARED*, in table *PATRON_ACCOUNT*. As for why we need to store *ACCOUNT_ID* in a *VARCHAR*, if we only use *INT* to store *ACCOUNT_ID*, and if user or manager input a character mistakenly, then there will be a datatype different here, and the program will not be processed.

  The second one is *INT*, which is used for integer values. For example, we store *B_ORDER_ID* in table *BORROW_ORDER* store as *INT* type. Also, it was generated by default as identity. Then, other primary keys(originally all primary keys are *INT*) that do not need SQL query operation will not be changed to *VARCHAR*.

  The third one is *DATE*, which is in the format of YYYY-MM-DD. For the deadline of return books, we use data type *DATE* for mathematical calculation to calculate, for example *DUE_DATE*, and *NOTE_TIME* in table *BORROW_ORDER* is in data type of *DATE*.

SQL running query (for Java codes implementation)

We designed a few procedures in SQL, which includes log in, checking, recording, searching, borrow, desire, reserve. And when users do conduct any behavior, the system can be updated.

First I will introduce the procedure of log in. Our command is here:

select ACCOUNT_PASSWORD, IS_DEACTIVATED from patron_account

where account_id = ('&account_id');

select M_ID, M_password from manager

where M_ID = ('&account_id');

When there is a value selected, it means that the user has already inputted a correct id, so we can find the matching password, and if the password in the database meets with the password that user inputted, it can meet with the basic condition. If the IS_DEACTIVATED term is ‘NO’, then the user can enter the management system.

After entering the system, we can conduct several different operations, for example, the user can check their book holding record. The command is following:

select a.book_name

from book a

where (a.book_id in (select

book_id from borrow_order

where (account_id = '&account_id'

and

due_time < sysdate)

));

By this procedure, the system can find the book that is already overdue, so that users can make sure they return their books in time and avoid being deactivated.

and when some patrons did not return their books after the deadline for a long period, the manager account has the ability to deactivate the accounts by following the command below:

update patron_account

set IS_deactivated = 'YES'

where account_id IN (select account_id

from borrow_order

```
where
due_time < sysdate
and
Is_returned = 'NO');
```

By this procedure, the system can quickly track the unconventional accounts, which enables the manager to maintain the normal use of the library.

After logging into the system, the patrons are able to check their borrow record and be reminded about when to return books. The command is here:

```
select book_name
from book a
where(a.book_id in (select book_id
        from borrow_order b
        where (b.account_id = '&account_id')
        and
        b.note_time < sysdate)
);
```

By this procedure, the system will provide the borrow record of the patron, and give details of the book status and deadline.

Since the patrons can select desired books, they can also check if their desired books are available after logging in. The codes are below:

```
select a.book_name from book a
where(a.book_id IN (select b.book_id from stat b
        where( b.book_id IN (select c.book_id from desire c
                where c.account_id = '&account_id')
        and b.Is_borrowed = 'NO'
        and b.IS_RESERVED = 'NO'))
```

);

By this procedure, the system will check the status of the desired books, and enable the patron to do further operations.

The following command below is for searching books, which are the basic SQL parts of the system. The patrons should input the book name, author or category for the system to check.

select distinct a. from book a, stat b*

where (upper(a.book_name) like upper('%&name%')

or upper(a.author) like upper('%&author%')

or upper(a.category) like upper('%&category%'))

and a.book_id = b.book_id

and b.Is_borrowed = 'NO'

and b.Is_reserved = 'NO';

By this procedure, the system will compare this information with the book id in the database, and give out the information and status of the book.

Similarly, the patrons can follow instructions below to search e-books:

*select distinct * from e_book*

where upper(e_name) like upper('%&name%')

or upper(e_author) like upper('%&author%')

or upper(e_category) like upper('%&category%');

If a book is borrowed by the patrons, the system will update the borrow status currently by following this command:

set IS_borrowed = 'YES'

where book_id = '&id';

insert into borrow_order(book_id,account_id,is_returned,due_time,note_time)

values('&id','&account','NO',sysdate+28,sysdate+25);

By this procedure, the system will change the status of the books. The borrowed books cannot be borrowed by other patrons. Meanwhile, the system will calculate the deadline of returning the books, and remind the patron when this day is approaching.

At the same time, the system will collect the times for a book being borrowed by this command:

set BORROW_times = BORROW_times + 1

where book_id = '&BOOK_ID';

By this procedure, the times being borrowed for a book will be counted, and the most popular books will be provided to the manager.

When the patrons are interested in some books, but they don’t want to borrow them now, they can choose to desire that book, and the system will record their desire, the desire times will be recorded and be reported to the manager. The manager will know which book is more popular.

--update desire:

insert into desire

values('&book_id', '&account_id');

update book_record

set desire_times = desire_times+1

where book_id = '&book_id';

When users need to return the books, they can operate the system to conduct the procedure. the return procedure will update the record of the borrow order and the book status.

--update return:

update stat

set Is_borrowed = 'NO'

where book_id = '&id';

update borrow_order

set Is_returned = 'YES'

where book_id = '&id';

When the user wants to borrow a specific book in the following days, it can use the “reserve” function. The book will be reserved for 3 days. If the user doesn't borrow the book in time, the reserve order will be canceled automatically. Also the reserve record will be recorded in the record table.

--update reserve:

update stat

set Is_reserved = 'YES'

where book_id = '&BOOK_id';

insert into reserve_order(book_id,account_id,cancel_time,is_borrowed,is_canceled)

values('&BOOK_id','&account',sysdate+3,'NO','NO');

update book_record

set reserve_times = reserve_times+1

where book_id = '&BOOK_id';

Java

- **Modular Design(how to realize each functions using different method)**

```
import oracle.jdbc.driver.OracleConnection;
import oracle.jdbc.driver.OracleDriver;

import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.sql.*;
import java.util.ArrayList;
import java.util.Properties;
import java.util.Scanner;
```

Firstly, We have imported the SQL package and Oracle package of Java to realize the connection and operation with the database and mail package makes the system able to send email to the patrons;These Util packages are imported to call the tool package of java itself to implement arraylist data structure and input.

```
public static String msg;
40 usages
static String accountId;
2 usages
static Scanner scanner = new Scanner(System.in);
24 usages
static OracleConnection conn;
```

Then we use object oriented programming, In the controller class, we use the static String attribute msg to update the user's input, the accountId to record the ID of the logged in user, and the scanner to read the information. We save conn to store the connection with oracle, so that we can maintain the connection even if we use different methods.

```
String username = "\"20084413d\"";          // e.g. "98765432d"
String pwd = "yedwebcs";
// Connection
DriverManager.registerDriver(new OracleDriver());
conn = (OracleConnection) DriverManager.getConnection(
        url: "jdbc:oracle:thin:@studora.comp.polyu.edu.hk:1521:dbms", username, pwd);
```

We use our own student account to log in Oracle with the same method in class.

```
stmt.executeQuery( sql: "update patron_account set IS_deactivated = 'YES' where account_id IN (select account_id " +
        "from borrow_order where due_time < sysdate and Is_returned = 'NO')");
```

Then by SQL we can update and check if the account is deactivated. If so, the system will stop it logging in and remind the user of the account to contact with management department.

```
String uOrm = originFace();
```

Then we call the originFace method and return to the login status, dividing it into “man”, ”use” and “deacticated”.

```
public static String originFace() throws SQLException {
    String passWord;

    Statement stmt = conn.createStatement();

    System.out.println("|-------------------------------------------------------------|");
    System.out.println("|                Library Management System                    |");
    System.out.println("|                          Log in                             |");
    System.out.println("|   ---------------------                                     |");
    System.out.println("|  | Please enter your ID |:");
```

At originFace(), first, we connect stmt to oracle, print the interface, and ask the user to enter the ID,

```
updateMsg();
accountId = msg;
String type;
ResultSet rset = stmt.executeQuery( sql: "select ACCOUNT_PASSWORD, IS_DEACTIVATED from " +
        "patron_account where account_id = '" + msg + "'");
if (rset.next()) {
    passWord = rset.getString( columnLabel: "ACCOUNT_PASSWORD");
    String status = rset.getString( columnLabel: "IS_DEACTIVATED");
    if (status.equals("YES")) {
        System.out.println("\033[31mYour account is DEACTIVATED!\033[0m");
        return "block";
    }
    type = "use";
```

Then use SQL to read the user's input, and select the account password and status in the database. If the account has been deactivated, stop the program immediately and notify the user that he is unable to log in.

```
else {
 rset = stmt.executeQuery( sql: "select M_password from manager where M_ID = '" + msg + "'");
 if (rset.next()) {
     passWord = rset.getString( columnLabel: "M_password");
     type = "man";
 } else {
```

If the user account information is not extracted, we will test whether it is a manager account,

```
lse {
 while (true) {
     System.out.println("The ID is not exist, please enter again!");
     updateMsg();
     accountId = msg;
     rset = stmt.executeQuery( sql: "select ACCOUNT_PASSWORD, IS_DEACTIVATED from " +
             "patron_account where account_id = '" + msg + "'");
     if (rset.next()) {
         passWord = rset.getString( columnLabel: "ACCOUNT_PASSWORD");
         String status = rset.getString( columnLabel: "IS_DEACTIVATED");
         type = status.equals("YES") ? "quit" : "use";
         break;
     } else {
         rset = stmt.executeQuery( sql: "select M_password from manager where M_ID = '" + msg + "'");
         if (rset.next()) {
             passWord = rset.getString( columnLabel: "M_password");
             type = "man";
             break;
```

If not, repeat the process until the user enters valid account.

```
System.out.println("|   ----------------------                                  |");
System.out.println("|                                                           |");
System.out.println("|   ----------------------                                  |");
System.out.println("|  | Please enter your password |:");
updateMsg();
String password = msg;
while (!password.equals(passWord)) {
    System.out.println("The password is not right, please enter again!");
    updateMsg();
    password = msg;
}
```

For the password, we call the previously stored password variable. Only when the user enters the correct password, will we allow him to log in.

```
System.out.println("|   ----------------------                                  |");
System.out.println("|                                                           |");
System.out.println("|                          Welcome!                         |");
System.out.println("|-----------------------------------------------------------|");
```

Then we will shoe the main page.

When the user logs in successfully, the system will immediately detect whether there is an email to send to him (return or desire book available). If so, we will send an email to the user's saved mailbox.

```
public static String userFace() throws SQLException {
    Statement stmt = conn.createStatement();
    ResultSet rset;
    System.out.println(" ------------------------------------------------------------ ");
    System.out.println("|                          Main Page                         |");
    System.out.println("|                                                  --------    |");
    System.out.println("|                What do you want to do?          | Quit |    |");
    System.out.println("|               Enter '1' for searching books      --------    |");
    System.out.println("|               Enter '2' for checking records                 |");
    System.out.println("|               Enter '3' for reading notice                   |");
    System.out.println("|               Enter '4' for return books                     |");
    System.out.println("|               Enter '5' for cancel reserve                   |");
    System.out.println("|               Enter 'quit' for quiting                       |");
    System.out.println("|                                                              |");
    System.out.println("|  --------------       --------------      -------------      |");
    System.out.println("|  | Search for |       | Check your |      | Important |      |");
    System.out.println("|  |    Books   |       |   Records  |      |  Notice   |      |");
    System.out.println("|  --------------       --------------      -------------      |");
    System.out.println("|                                                              |");
    System.out.println("|          --------------          -------------               |");
    System.out.println("|          |   Return   |          |  Cancel   |               |");
    System.out.println("|          |   Books    |          |  Reserve  |               |");
    System.out.println("|          --------------          -------------               |");
    System.out.println(" ------------------------------------------------------------ ");
```

For patrons, we will print the patron's main page; Otherwise we will print manager's main page.

```
public static String managerFace() {
    System.out.println(" ------------------------------------------------------------ ");
    System.out.println("|                          Main page                         |");
    System.out.println("|                                                  --------    |");
    System.out.println("|                What do you want to do?          | quit |    |");
    System.out.println("|               enter 1 for searching books        --------    |");
    System.out.println("|               enter 2 for checking records                   |");
    System.out.println("|               enter 3 for reading notice                     |");
    System.out.println("|               enter 4 for return books                       |");
    System.out.println("|               enter 5 for cancel reserve                     |");
    System.out.println("|               enter 6 for report                             |");
    System.out.println("|               enter 7 for deactivating users                 |");
    System.out.println("|               enter 8 for activating users                   |");
    System.out.println("|               enter quit for quiting                         |");
    System.out.println("|                                                              |");
    System.out.println("|  --------------       --------------      -------------      |");
    System.out.println("|  | search for |       | check your |      | Important |      |");
    System.out.println("|  |    books   |       |   record   |      |  Notice   |      |");
    System.out.println("|  --------------       --------------      -------------      |");
    System.out.println("|                                                              |");
    System.out.println("|  --------------         ----------        -------------      |");
    System.out.println("|  |   return   |         | report |        |  cancel   |      |");
    System.out.println("|  |   books    |         ----------        |  reserve  |      |");
    System.out.println("|  --------------                           -------------      |");
    System.out.println(" ------------------------------------------------------------ ");
```

After that, we let the user input numbers to choose the funcitons.

```
while (!choice.equals("1") && !choice.equals("2") && !choice.equals("3") && !c
    System.out.println("The enter is not legal, please enter 1 or 2 or 3 or 4!
    updateMsg();
    choice = msg;
}
```

```
public static int searchFace() throws SQLException {

    Statement stmt = conn.createStatement();
    System.out.println("|-------------------------------------------------------------|");
    System.out.println("|                Library Management System                    |");
    System.out.println("|                     Searching system                        |");
    System.out.println("|    ----------------------                                   |");
    System.out.println("|      Please input information of the book you desire below   |");
    System.out.println("|      If you don't know few information, please enter '/'     |");
    System.out.println("|    ----------------------                                   |");
    System.out.println("|                                                             |");
    System.out.println("|    ----------------------                                   |");
    System.out.println("|   | Name of the book |:");
    updateMsg();
    String bookName = msg;
    System.out.println("|    ----------------------                                   |");
    System.out.println("|                                                             |");
    System.out.println("|    ----------------------                                   |");
    System.out.println("|   | Author of the book |:");
    updateMsg();
    String author = msg;
    System.out.println("|    ----------------------                                   |");
    System.out.println("|                                                             |");
    System.out.println("|    ----------------------                                   |");
    System.out.println("|   | Category |:");
    updateMsg();
```

Then we search books in the database by let users input the book name, author, and category.

```
ResultSet rset = stmt.executeQuery( sql: "select distinct a.* from book a where (upper(a.book_name)
if (!rset.next()) {
    do {
        System.out.println("According to given information, there is no book available.");
        System.out.println("If you want to search again, please enter 0. If you want to go back t
        updateMsg();
    } while (!msg.equals("0") && !msg.equals("back"));
    return msg.equals("0") ? 0 : -1;
```

If the system can't find any book, we will let users to choose whether input information for one more time or back to the main page.

```
ArrayList<String> arrayList = new ArrayList<>();
int i = 1;
int size = 0;
rset = stmt.executeQuery( sql: "select distinct a.* from book a, stat b where upper(a.book_name) like upper('%" +
while (rset.next()) {
    String id = rset.getString( columnLabel: "BOOK_ID");
    String name = rset.getString( columnLabel: "BOOK_NAME");
    String author1 = rset.getString( columnLabel: "AUTHOR");
    String isBN = rset.getString( columnLabel: "ISBN");
    String category1 = rset.getString( columnLabel: "CATEGORY");
    String loc = rset.getString( columnLabel: "LOCAT");
    System.out.printf(
            "\033[32mNumber: %s\nBOOK_ID: %s\nBOOK_NAME: %s\nAUTHOR: %s\nISBN: %s\nCATEGORY: %s\nLOCATION: " +
                    "%s\n\u001B[0m%n", i, id, name, author1, isBN, category1,loc);
    i++;
    arrayList.add(id);
```

If any book is searched, we will print out all of the books that meet the requirement.

```
System.out.println("If you find your book, please enter the number of the book. " +
        "If you don't find your book, and you want to search again, please enter 0, " +
        "or you want to go back to main page, please enter \"back\"");
updateMsg();
if (msg.equals("0")) {
    return 0;
}
if (msg.equals("back")) {
    return -1;
}
boolean flag = true;
if (!msg.equals("")){
    flag = false;}
else {
    for (int j = 0; j < msg.length(); j++) {
        if (!(msg.charAt(j) <= '9' && msg.charAt(j) >= '0')) {
            flag = false;
            break;
        }
        if ((Integer.parseInt(msg) > arrayList.size()) || (Integer.parseInt(msg) < 1)) flag = false;
    }
}
```

Then we will ask the user whether he/she finds the target book. If yes, the customer can choose one of the books to do further behavior, or our user can choose to search again or back to the main page

```
while (true) {
    if (flag == false) {
        System.out.println("You have to enter a book number, '0' to search again, or 'back' to go back to main!");
        updateMsg();
        if (msg.equals("0")) return 0;
        if (msg.equals("back")) return -1;
        flag = true;
        for (int j = 0; j < msg.length(); j++) {
            if (!(msg.charAt(j) <= '9' && msg.charAt(j) >= '0')) {
                flag = false;
                break;
            }
            if ((Integer.parseInt(msg) > arrayList.size() + 1) || (Integer.parseInt(msg) < 1)) flag = false;
        }
    } else break;
}
```

If the book is available, our customer can choose to borrow it or reserve it.

```
if (isA) {
    System.out.println("The book is Available now! You can enter '1' (borrow)(take it now!), '2' (reserve)(take it in 3 days!),
```

We can judge the users' input by "if" command, and print out our input information, which includes whether the user borrow or reserve the book successfully.

```
if (msg.equals("1")) {
    stmt.executeQuery( sql: "update stat set IS_borrowed = 'YES' where book_id = '" + id + "'");
    stmt.executeQuery( sql: "insert into borrow_order(book_id,account_id,is_returned,due_time,note_time) values('" + id +
    stmt.executeQuery( sql: "update book_record set BORROW_times = BORROW_times + 1 where book_id = '" + id + "'");
    System.out.println("Congratulations, your borrow is successful!");
    return -1;
}
if (msg.equals("2")) {
    stmt.executeQuery( sql: "update stat set Is_reserved = 'YES' where book_id = '" + id + "'");
    stmt.executeQuery( sql: "insert into reserve_order(book_id,account_id,cancel_time,is_borrowed,is_canceled) values('" +
    stmt.executeQuery( sql: "update book_record set reserve_times = reserve_times + 1 where book_id = '" + id + "'");
    System.out.println("Congratulations, your reserve is successful!");
    return -1;
}
```

If the book is unavailable, we will enter the "desire" logic, in that page we will let the user choose to desire the book or not.

```
System.out.println("Sorry, the book is not Available now! You " +
        "can enter 1 (desire)(notify you when available), " +
        "0 (search again) or back (back to main menu)");
updateMsg();
if (msg.equals("0")) {
    return 0;
}
if (msg.equals("back")) {
    return -1;
}
if (msg.equals("1")) {
    stmt.executeQuery( sql: "insert into desire values('" + id + "', '" +
    stmt.executeQuery( sql: "update book_record set desire_times = desire
    System.out.println("Congratulations, your desire is successful!");
    return -1;
```

Then the user will back to the main page

In the record page, we will print out the all of the borrow order, whether it's returned, together with the due date.

```
Statement stmt = conn.createStatement();
System.out.println("|-------------------------------------------------------------|");
System.out.println("|                Library Management System                    |");
System.out.println("|                           Record                            |");
System.out.println("|   ---------------------                                     |");
System.out.println("|  |Borrow records:|                                          |");
ResultSet rset = stmt.executeQuery( sql: "select book_name, is_returned, due_time from book a, b
while (rset.next()) {
    String bookName = rset.getString( columnLabel: "book_name");
    String isReturned = rset.getString( columnLabel: "is_returned");
    Date dueDate = rset.getDate( columnLabel: "Due_time");
    System.out.println("|  BookId: " + bookName + " Isreturned: " + isReturned + " DueDate: "
}
```

Then we will print out the reserve list, we print it with while loop.

```
System.out.println("|                                                             |");
System.out.println("|   ---------------------                                     |");
System.out.println("|                                                             |");
System.out.println("|   ---------------------                                     |");
System.out.println("|  |Reserve records:|                                         |");
rset = stmt.executeQuery( sql: "select book_name, cancel_time from book a, reserve_order b w
while (rset.next()) {
    String bookName = rset.getString( columnLabel: "BOOK_name");
    Date cDate = rset.getDate( columnLabel: "cancel_time");
    System.out.println("|  BookName: " + bookName + " CancelDate: " + cDate);
}
```

In “notice” page, we will notice the user with the information of the book that should be returned within three days.

```
System.out.println("|   ---------------------                                     |");
System.out.println("|                                                             |");
System.out.println("|   ---------------------                                     |");
System.out.println("|  | Desired book is available |:                            |");
rset = stmt.executeQuery( sql: "select a.book_name from book a where(a.book_id IN (select b
while (rset.next()) {
    String name = rset.getString( columnLabel: "book_name");
    System.out.println("|  BookName: " + name);
}
System.out.println("|   ---------------------                                     |");
```

Then we will print out users’ desire and the available books in users’ desire list.

```java
public static void returnFace() throws SQLException {
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery( sql: "select book_name, " +
            "is_returned, due_time from book a, borrow_order b where" +
            " a.book_id = b.book_id and b.account_id = '" + accountId + "'");
    if (!rset.next()) {
        System.out.println("You don't borrow any book");
        return;
    }
    System.out.println("Your borrowed records are below:");
    ArrayList<String> arrayList = new ArrayList<>();
    int i = 1;
    rset = stmt.executeQuery( sql: "select a.book_id,book_name, is_returned, due_time from b
    while (rset.next()) {
        String bookId = rset.getString( columnLabel: "book_id");
        String bookName = rset.getString( columnLabel: "book_name");
        String isReturned = rset.getString( columnLabel: "is_returned");
        Date dueDate = rset.getDate( columnLabel: "Due_time");
        System.out.println("|  Number of book: " + i + "|  BookName: " + bookName + " Isre
        arrayList.add(bookId);
        i++;
    }
```

On the Return interface, we first select all borrow books from the database, and then print them out to let customers choose to return books

```java
System.out.println("Please enter the number of the book to return. If you don't find your book,
        " to go back to main page, please enter \"back\"");
updateMsg();
```

Then we judge it with a more complex logic

```
if (!msg.equals("")){
    flag = false;}
else {
    for (int j = 0; j < msg.length(); j++) {
        if (!(msg.charAt(j) <= '9' && msg.charAt(j) >= '0')) {
            flag = false;
            break;
        }
        if ((Integer.parseInt(msg) > arrayList.size()) || (Integer.parseInt(msg) < 1)) flag = fal
    }
}
while (true) {
    if (flag == false) {
        System.out.println("You have to enter a book number, or enter back to go back to main!");
        updateMsg();
        if (msg.equals("back")) {
            return;
        }
        flag = true;
        if (!msg.equals("")){
            flag = false;}
        else {
            for (int j = 0; j < msg.length(); j++) {
                if (!(msg.charAt(j) <= '9' && msg.charAt(j) >= '0')) {
                    flag = false;
                    break;
                }
```

If it's successful, we will change the data in the database.

```
int index = Integer.parseInt(msg) - 1;
String id = arrayList.get(index);
rset = stmt.executeQuery( sql: "update stat set Is_borrowed = 'NO' where book_id = '" + id + "'");
rset = stmt.executeQuery( sql: "update borrow_order set Is_returned = 'YES' where book_id = '" + i
System.out.println("return successfully");
```

```
ublic static void cancelFace() throws SQLException {
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery( sql: "select book_name, cancel_time from book a, reserve_
    if (!rset.next()) {
        System.out.println("You don't reserve any book");
        return;
    }
    System.out.println("Your reserved records are below:");
    ArrayList<String> arrayList = new ArrayList<>();
    int i = 1;
    rset = stmt.executeQuery( sql: "select a.book_id, book_name, cancel_time from book a, reserve
    while (rset.next()) {
        String book_id = rset.getString( columnLabel: "book_id");
        String book_name = rset.getString( columnLabel: "book_name");
        Date cancelDate = rset.getDate( columnLabel: "cancel_time");
        System.out.println("|  Number of book: " + i + "|  BookId: " + book_id + " BookName: "
        arrayList.add(book_id);
        i++;
    }
    System.out.println("Please enter the number of the book to cancel. If you don't find your b
    updateMsg();
    if (msg.equals("back")) {
        return;
    }
```

In the cancel page, like the return procedure, check whether book reserve books, and ask users to choose a book to cancel reserve.

```
Statement stmt = conn.createStatement();
System.out.println("|-------------------------------------------------------------|");
System.out.println("|                Library Management System                    |");
System.out.println("|                          Report                             |");
System.out.println("|   ---------------------                                     |");
System.out.println("|  |book_record:|                                             |");
ResultSet rset = stmt.executeQuery( sql: "select * from book_record order by borrow_TIMES");
int i = 1;
while (rset.next()) {
    String book_id = rset.getString( columnLabel: "book_id");
    int BORROW_TIMES = rset.getInt( columnLabel: "BORROW_TIMES");
    int DESIRE_TIMES = rset.getInt( columnLabel: "DESIRE_TIMES");
    int RESERVE_TIMES = rset.getInt( columnLabel: "RESERVE_TIMES");
    System.out.println("|  Number of book: " + i + "|  BookId: " + book_id + " Borrow_tim
    i++;
}
System.out.println("|   ---------------------                                     |");
System.out.println("|                                                             |");
System.out.println("|   ---------------------                                     |");
System.out.println("|  | DEACTVATED_record |:                                     |");
rset = stmt.executeQuery( sql: "select account_id from patron_account where IS_DEACTIVATED =
i = 1;
while (rset.next()) {
    String account_id = rset.getString( columnLabel: "account_id");
    System.out.println("|  Number of account: " + i + "|  account_id: " + account_id);
    i++;
}
```

In report page, we print out all of the deactivated account record, and the operating information of all of books.

- Java implementation of data type

We use the "*boolean*" type to illustrate if the user or manager wants to go to the last page, if the user enters "back" or "quit", the method *wantback()*, will return a boolean then the terminal will show the main page or directly quit this program. Also, we use boolean to check if the string extracted from *VARCHAR* from SQL testing data is the same as the user or manager input. The boolean flag in method *searchFace()* is to check if the enter is valid for the system.

As for string data type, the user or manager's input should always be set as the string. Although the input is always "1", "2", "3" which can be integers, if there's something wrong with user input, i.e. keyboard input of some alphabet, the system will be difficult to match with the original type, integer. Therefore, we changed the integer type to string. In the part of checking account ID and account password, we also change the integer type of account ID to string.

For getting the user or manager's response, we import scanner data type, and then convert Scanner into string msg. We use "integer" to store the number of books collected from *System.in Scanner* for easier calculation and comparison. To get the date from the test data, we use "date" to store due_time. We also store the book ID in "*Arraylist*" to print it out later in *searchFace()*, *recordFace()* and *cancelFace()*.

Oracle connection

Our command for oracle connection is as followed:

```
OracleConnection conn =
        (OracleConnection) DriverManager.getConnection(
            "jdbc:oracle:thin:@studora.comp.polyu.edu.hk:1521:dbms", username, pwd);
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery("/SQL query here/");
```

Advantages

For the fore-end, whatever user or manager can follow the introduction on the interface to easily master the library system after brief familiarity, even if it is the first time to use the system. Each choosing action can be completed by typing a simple number instead of typing the whole word string, which is efficient for the user to apply. The main page is clear for users to do their operations. Meanwhile, the system can provide different services to users with different identities by automatically identifying their account information, and lead them to different pages and functions. By following the instructions we provided, the work will be more efficient.

For the back-end, we provided a big database consisting of many samples to test the operation of the system. In order to realize the ideal function, along with the actual situation of the group and the specific situation in daily life, we used SQL to write logic and Java as

the basic program language to complete the construction of this system. This makes the system have clear logic and is self consistent. If possible bugs and problems occur, it is easy to locate and correct them.

The system has the potential of being implied to the actual situation. Library construction has always been an important part of university development planning. The library system we provide for programming language construction is consistent with this demand, and has the potential to further develop and popularize in a wide range.

Conclusion

Overall, the whole project follows all features of requirement and based on the user-friendly idea. Our system has unique designated pages and complete functions. By reading the user's guide, the users can access the system in many different ways. Also, the user can be identified by the system so that the system can provide different users with different rights. For managers, they can get more information than normal users and have more rights in this system, which implies the different layers of the database.

For the functions, our design is based on a user-friendly idea, so we try our best to simplify the users' operation. Our system doesn't contain any complex operations, they don't need to type any executive command, only they need to type is just a few numbers and the information they know.The users can operate the system easily just by simply clicking a few numbers, and our system will identify the users' target and respond to users behavior automatically. Besides, our system can identify the typing error and remind users to operate the system following the guide. The quitting operation appears in every step, so the user can quit this system anytime, that procedure implies the freedom of using this system.

By doing that project, we develop our creative ability to design the database system, our creativity helps us to design different functions. We believe that our project not only fulfills all of the project requirements, but also has the meaning of usage in normal daily life. After many times of testing, we think that our project has complete functions to support the library system. Our testing data includes 30 books, together with many other pieces of information, although we don't know the situation in the real world, this framework can also provide us

with a basic idea of what kind of logical framework we need in order to support a library system. We understand that our system can just be a supplementary system in the real world application. For example, if the user chooses to use the return function, we can't identify whether the book is really returned by the borrower, and in order to achieve this target, interaction with other systems is necessary, so our system needs to be modified further to use in the real world. However, we already try our best to make it as a useful system, so there is still referencing valuation in our system. We believe that under our modification, it can be a perfect application eventually.