Library Management System Database

University of Texas at Arlington

Department of Computer Science and Engineering

CSE 3330 – 004: Database Systems and File Structures

Dr. Nadra Guizani

Project 2 Part 3

Fall 2024

Chime Nguyen

Ivan Ko

Trung Nguyen

# HONOR CODE

|  |
| --- |
| I pledge, on my honor, to uphold UT Arlington’s tradition of academic integrity, a tradition that values hard work and honest effort in the pursuit of academic excellence.  I promise that I will submit only work that I personally create or that I contribute to group collaborations, and I will appropriately reference any work from other sources. I will follow the highest standards of integrity and uphold the spirit of the Honor Code. |

# TABLE OF CONTENTS

I.) [HONOR CODE](#_HONOR_CODE) ……………………………………………………………………2

II.) [INTRODUCTION](#_INTRODUCTION) ……………………………………………………………………4

IV.) [PART 3 SQL QUERY RESULTS](#_PART_3_SQL) ……………………………………………………5

V.) [PART 3 GUI IMPLEMENTATION](#_PART_3_GUI) ……………………………………………………9

a.) [Dropdown Menu And Complete Query Button For Queries](#_Dropdown_Menu_And) …………… 9

b.) [Phase 2 Queries](#_Phase_2_Queries) …………………………………………………… 11

c.) [Phase 3 Queries](#_Phase_3_Queries) …………………………………………………… 12

d.) [Phase 3 Requirements](#_Phase_3_Requirements) …………………………………………………… 30

e.) [Design Philosophy And Difficulties While Implementing The GUI](#_Design_Philosophy_And) …… 52

VI.) [CONCLUSION](#_CONCLUSION) ……………………………………………………………………53

VII) [REFERENCES](#_REFERENCES) ……………………………………………………………………54

a.) [Part 2 References](#_Part_2_References) …………………………………………………… 54

b.) [Part 3 References](#_Part_3_References) …………………………………………………… 54

VIII.) [CONTRIBUTION LIST](#_CONTRIBUTION_LIST) ……………………………………………………………55

a.) [Part 2 SQL Queries & GUI Implementation](#_Part_2_SQL) …………………………… 55

b.) [Part 3 Report](#_Part_3_Report) …………………………………………………………… 55

c.) [Part 3 Readme](#_Part_3_Readme) …………………………………………………………… 55

d.) [Part 3 SQL Queries & GUI Implementation](#_Part_3_SQL_1) …………………………… 56

**e.)** [Part 3 GUI Requirements](#_Part_3_GUI_1) …………………………………………… 55

# INTRODUCTION

The goal of this project is to demonstrate our comprehensive understanding and application of database design and SQL querying development skills acquired throughout this course using the Library Management System Database. We were tasked to create an ER Diagram and Database Schema, produce SQL tables and queries to work with a GUI, and write a report on our workflow and results. Part one of this project focused on developing the ER Diagram based on the requirements for the Library Management System. Part two of the project focused on implementing the SQL tables and SQL queries for the system and running the queries in SQLite alone. We kept Part 2 information in the report as it will be relevant for the GUI implementation in Part 3.

For part three, we were tasked with creating a Graphical User Interface (GUI), using Python and its Tkinter library. The GUI we designed performs specific queries rather than being a robust query engine - we implement several requirements and generalized all prior queries from Part 2 to be included in the GUI’s list of query options – this includes functionalities for adding new books, borrowers, and branches along with the ability to generate several different reports. Alongside the development of the GUI, we were required to write a separate Readme file that explains to the users on how to set up and use the GUI for the Library Management System and include this report that explains how these queries work along with GUI screenshots and results.

# PART 3 SQL QUERY RESULTS

## Query 1

### Add an extra column ‘Late’ to the Book\_Loan table. Values will be 0-for non-late retuns, and 1-for late returns. Then update the ‘Late’ column with '1' for all records that they have a return date later than the due date and with '0' for those were returned on time.

Alter: Add a new column late – boolean value for returned late or not

To add a new column to the table, the ALTER clause must be used with the table name BOOK\_LOANS and the new column’s name “Late” with its integer data type. The UPDATE query must be used to fill in the empty column with the boolean values. If the return date is past the due date, then late equals 1. Otherwise, the book loan is considered not late or not returned yet which results in late being 0 in both cases.

ALTER TABLE BOOK\_LOANS ADD COLUMN Late INTEGER;

UPDATE BOOK\_LOANS

SET Late = (

CASE

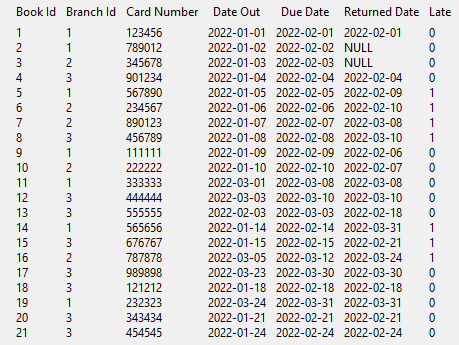
WHEN CAST(JULIANDAY(Returned\_date) > JULIANDAY(Due\_Date) AS INTEGER) THEN 1

WHEN CAST(JULIANDAY(Returned\_date) <= JULIANDAY(Due\_Date) AS INTEGER) THEN 0

ELSE 0

END

);



Above are the results of the query. The new “Late” column can be seen on the far right with its values being Booleans to represent if a book was turned in late or not

## Query 2

### Add an extra column ‘LateFee’ to the Library\_Branch table, decide late fee per day for each branch and update that column

Alter: Add a new column “LateFee” - decide the value for each branch

Adding a column to LIBRARY\_BRANCH is done with the ALTER TABLE query which adds a new column “LareFee” to hold FLOAR values. Next, the UPDATE query fills in the values for that column with respect to the cases. The cases are dependent on the library branch’s name and the late fee value is hardcoded with a random dollar amount.

ALTER TABLE LIBRARY\_BRANCH ADD COLUMN LateFee FLOAT DEFAULT 0.00;

UPDATE LIBRARY\_BRANCH

SET LateFee = (

CASE

WHEN Branch\_Name = 'Main Branch' THEN 100.00

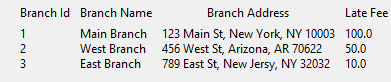
WHEN Branch\_Name = 'West Branch' THEN 50.00

WHEN Branch\_Name = 'East Branch' THEN 10.00

ELSE 420.69

END

);



The results of this query can be seen above with the new column being on the far right. Each Branch name in the query has the same late fee amount specified from the query.

## Query 3

### Create a **view** vBookLoanInfo that retrieves all information per book loan. The view should have the following attributes:

### Card\_No, Borrower Name, Date\_Out, Due\_Date, Returned\_date,

### Total Days of book loaned out as ‘TotalDays’ - you need to change weeks to days

### Book Title

### Number of days returned late – if returned before or on due\_date place 0

### Branch ID

### Total Late Fee Balance ‘LateFeeBalance’ - If the book was not returned late then fee = ‘0’

CREATE VIEW vBookLoanInfo AS

SELECT

BL.Card\_No,

BR.Name AS 'Borrower Name',

BL.Date\_Out,

BL.Due\_Date,

BL.Returned\_date,

CAST((JULIANDAY(BL.Returned\_date) - JULIANDAY(BL.Date\_Out)) AS INTEGER) AS TotalDays,

B.Title AS 'Book Title',

CASE WHEN BL.Late = 1

THEN CAST((JULIANDAY(BL.Returned\_date) - JULIANDAY(BL.Due\_Date)) AS INTEGER)

ELSE 0

END AS 'Days Returned Late',

BL.Branch\_Id,

CASE WHEN BL.Late = 1

THEN CAST((JULIANDAY(BL.Returned\_date) - JULIANDAY(BL.Due\_Date)) AS INTEGER) \* LB.LateFee

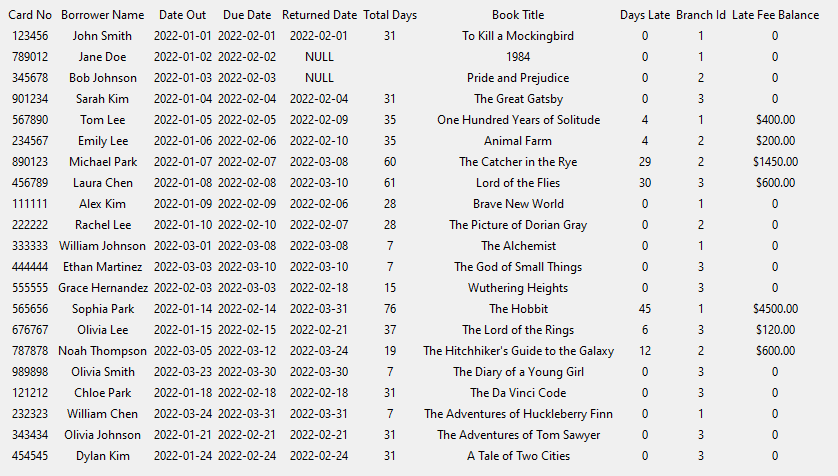
END AS LateFeeBalance

From BOOK\_LOANS BL

JOIN BORROWER BR ON BL.Card\_No = BR.Card\_No

JOIN BOOK B ON BL.BOOK\_Id = B.Book\_Id

JOIN LIBRARY\_BRANCH LB ON BL.Branch\_Id = LB.Branch\_Id;

To check if the view was successfully made, the command ‘SELECT \* FROM vBookLoanInfo’ can be used to list the whole view. The following screenshot shows the execution of the query returned to the GUI:

We can see the view is listing in order from left to right the Card\_No, Borrower Name, Date\_Out, Due\_Date, Return\_date, TotalDays, Book Title, Late Day count, Branch Id, and LateFeeBalance.

# PART 3 GUI IMPLEMENTATION

This section of the report is about how we implemented certain features for the GUI for the Library Management System. This section will also go into further detail on each of the individual queries. At the end of this section, we will also discuss the difficulties when creating the GUI.

## Dropdown Menu And Complete Query Button For Queries

For the GUI, we decided to go with a dropdown menu approach where the user could select any query, and the “Complete Query” button would initiate the code to complete it. Most of the code repeats with additional code that is needed with queries with more than one attribute output.

This part of the code deals with displaying details about each of the individual queries in the select\_from\_dropdown function:

A screen shot of a computer program

Description automatically generated

… and it repeats for every option that has a query from options [2] through [29] …

A screen shot of a computer program

Description automatically generated

This part of the code deals with creating the connection, calling the query’s specific function based on what the dropdown is currently selected in the do\_query function, returning the desired results from the function, and displaying the results, before closing the connection:

A screen shot of a computer program

Description automatically generated

… and it repeats for every option that has a query from options [2] through [29] …A screen shot of a computer code

Description automatically generated

## Generalization of Part 2’s Queries

Query 1 – We took the insert command from part 2 took the inputs and stored them as new borrowers rather than just a one-time command for one specific person.

Query 2 – Similar to Query 1, we took the idea of updating more than just one person’s phone number and generalized it to update a specified borrower’s number.

Query 3 – We generalized this query by allowing the user to increase the number of book copies in any branch by any value.

Query 4a – We generalized the Harry Potter query from the last part to be able to insert any new book with its author and publisher.

Query 4b – We generalized the query from the last phase to be able to insert any new branch along with its address rather than just the two branches specified.

Query 5 – We generalized the query to be able to search for books between any two dates rather than the specified dates from the last phase.

Query 6 – We reformatted this query to be able to search for a borrower tied to a book line by any return date.

Query 7 – Stayed the same since it creates a report.

Query 8 – It stayed the same since it lists the maximum number of days a book was loaned out.

Query 9 – We generalized this query to be able to create a report for any borrower rather than just one hard-coded specified person.

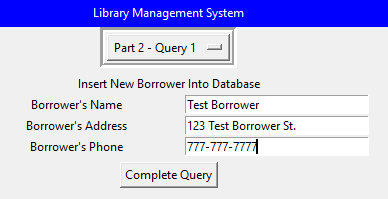
Query 10 – We generalized this query to be able to create a report for any branch and have it return the borrowers’ information.

## Phase 2 Queries

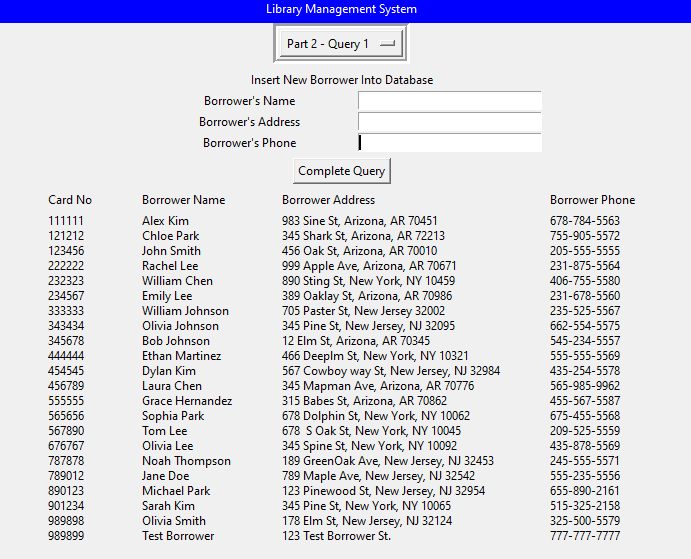
**Query 1:**

Objective: Insert New Borrower into the database

Input: New Borrower’s Name, Address, and Phone Number



Output: Inserted on the last row – Card No generated and given



Brief Description: The implementation of this query fulfills the query structure of “INSERT INTO BORROWER (Name, Address, Phone)” with the values given from the entry boxes. The card number is automatically given as an increment from the last known card number.

This function used the following query structure as its basis:

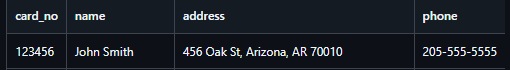
INSERT INTO BORROWER (Name, Address, Phone)

VALUES ('Chime Nguyen','701 S Nedderman Dr, Texas, TX 76019', '211-211-2112');

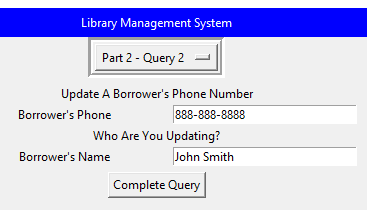
**Query 2:**

Objective: Update Borrower’s phone number given their name

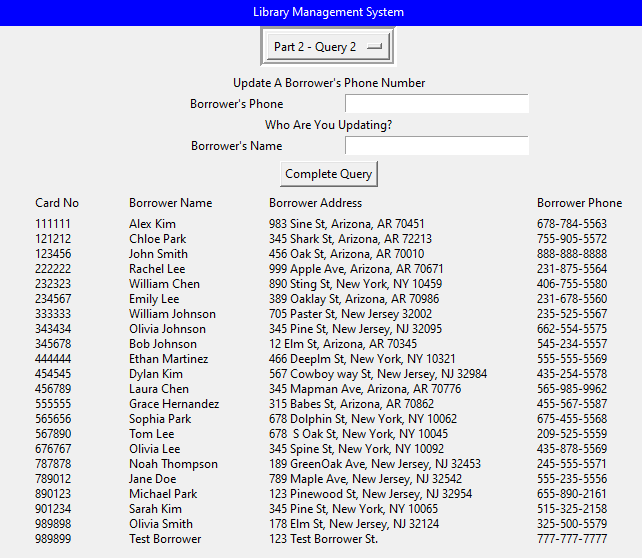
Original: John Smith has phone number 205-555-5555



Input: Borrower’s Name and their new Phone Number



Output: Updated on third row – phone number is now 888-888-8888



Brief Description: This function gets the input of borrower’s name and UPDATES the Phone tied to the Borrower Name inputted.

This function used the following query structure as its basis:

UPDATE BORROWER

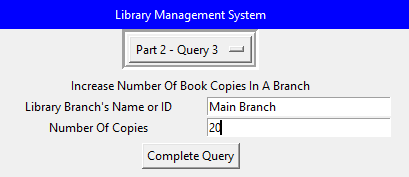
SET Phone = '837-721-8965'

WHERE Name = 'Chime Nguyen';

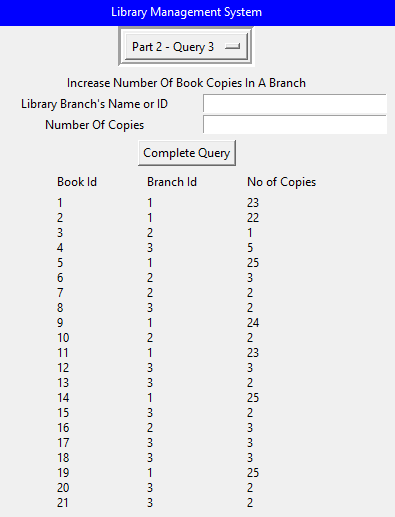
**Query 3:**

Objective: Update the number of copies by a given amount for every type of book in a branch

Input: Branch name and the number of copies to be added to the Branch’s inventory



Output: Every book in Main Branch (ID = 1) increased by 20 copies



Brief Description: Using the Branch Name or ID from the entry box, the query executes an UPDATE command to add some x number of copies for the specified Branch.

This function used the following query structure as its basis:

UPDATE BOOK\_COPIES

SET No\_Of\_Copies = No\_Of\_Copies + 1

WHERE Branch\_Id = (SELECT Branch\_Id

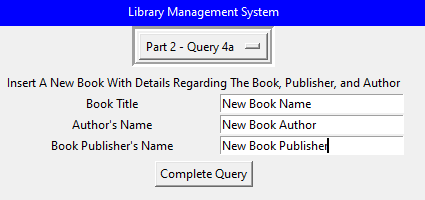
FROM LIBRARY\_BRANCH

WHERE Branch\_Name = 'East Branch'

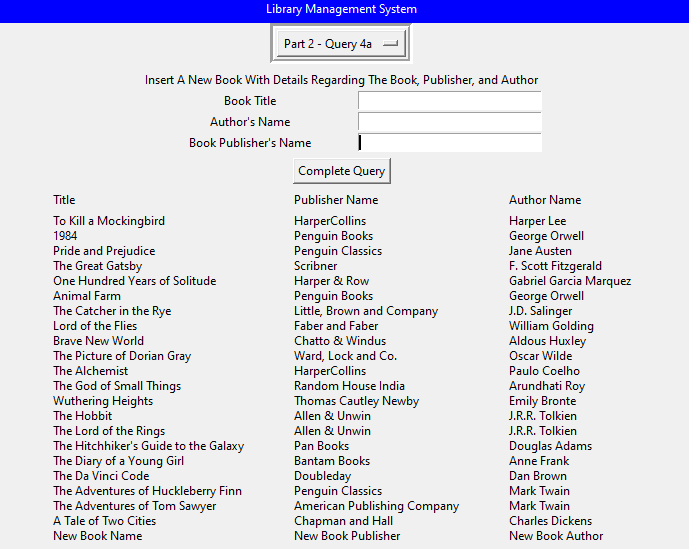
**Query 4a:**

Objective: Insert a new book into the database

Input: New Book’s Title, Author’s name, and Publisher’s name



Output: Last row – New Book Name is now a new book



Brief Description: This query function requires that all fields be filled and uses them to fulfill several subqueries like inserting new publishers, books, and their corresponding book author to fully complete the new book entry.

This function used the following query structure as its basis:

INSERT INTO PUBLISHER (Publisher\_Name, Phone, Address)

VALUES ('Oxford Publisheing', 'NULL', 'NULL');

INSERT INTO BOOK (Title, Publisher\_name)

VALUES ('Harry Potter and the Sorcerer''s Stone', 'Oxford Publisheing');

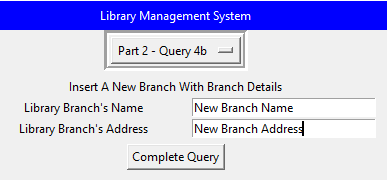
INSERT INTO BOOK\_AUTHORS (Book\_Id, Author\_Name)

VALUES ((SELECT Book\_Id FROM BOOK WHERE Title = 'Harry Potter and the Sorcerer''s Stone'), 'J.K. Rowling');

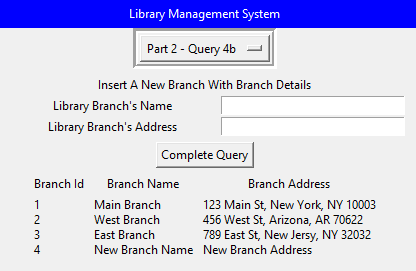
**Query 4b:**

Objective: Insert a new branch location into the database

Input: New Branch’s Name and Address



Output: Last Row – New Branch Name is now a new branch in the system



Brief Description: This query function uses the required entries to fulfill the query structure of: “INSERT INTO LIBRARY\_BRANCH (Branch\_Name, Branch\_Address)”

This function used the following query structure as its basis:

INSERT INTO LIBRARY\_BRANCH (Branch\_Name, Branch\_Address)

VALUES ('North Branch', '456 NW, Irving, TX 76100');

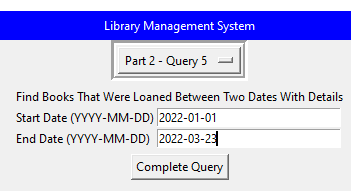
INSERT INTO LIBRARY\_BRANCH (Branch\_Name, Branch\_Address)

VALUES ('UTA Branch', '123 Cooper St, Arlington TX 76101');

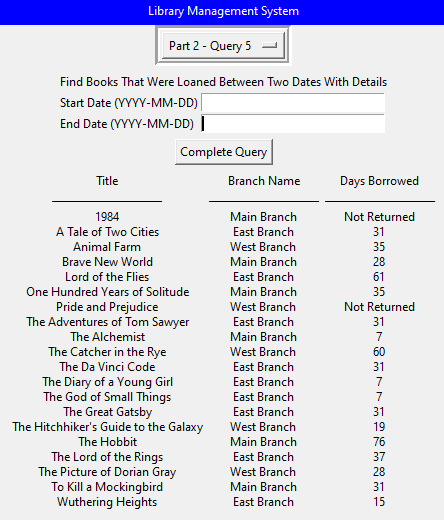
**Query 5:**

Objective: Select the books loaned between two dates

Input: Start and End dates in the format YYYY-MM-DD



Output: A collection of books checked out between Jan 1, 2022, and March 23, 2022, with their respective days borrowed and branch they were checked out from.



Brief Description: This query function does date calculations to retrieve the query results between the given dates.

This function used the following query structure as its basis:

SELECT B.Title, LB.Branch\_Name,

CASE WHEN BL.Returned\_date IS NOT 'NULL' THEN

CAST(JULIANDAY(BL.Returned\_date) - JULIANDAY(BL.Date\_Out) AS INTEGER)

ELSE CAST(JULIANDAY(CURRENT\_DATE) - JULIANDAY(BL.Date\_Out) AS INTEGER)

END AS Days\_Borrowed

FROM BOOK\_LOANS BL JOIN BOOK B ON BL.Book\_Id = B.Book\_Id

JOIN LIBRARY\_BRANCH LB ON BL.Branch\_Id = LB.Branch\_Id

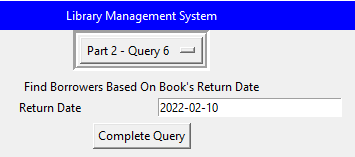
WHERE BL.Date\_Out BETWEEN '2022-03-05' AND '2022-03-23'

ORDER BY B.Title, LB.Branch\_Name;

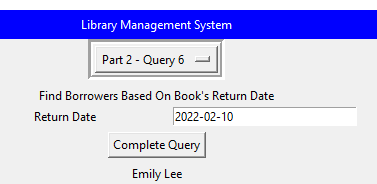
**Query 6:**

Objective: Select borrowers who returned a book on a given day

Input: Return Date to check



Output: Names of borrowers – One person returned a book on Feb 10, 2022



Brief Description: This query function does a join between borrower and book loans to search for distinct borrowers of any loans that match the given date.

This function used the following query structure as its basis:

SELECT DISTINCT bo.Name AS Borrower\_Names

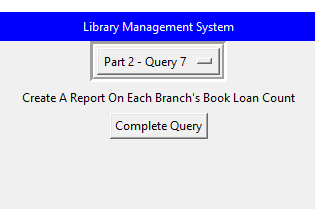
FROM BORROWER bo JOIN BOOK\_LOANS bl ON bo.Card\_No = bl.Card\_No

WHERE Returned\_date = 'NULL';

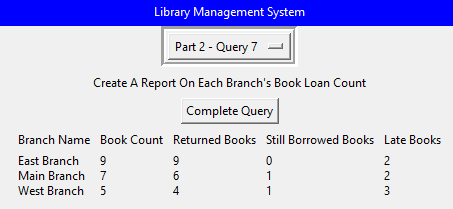
**Query 7:**

Objective: Select all information on book loans in every branch

Input: No Input



Output: Data regarding the returned status of all books in each branch



Brief Description: This query function creates a report by counting the number of books under each branch, then doing a date comparison and calculation to find the number of returned books, currently borrowed books, and late books.

This function used the following query structure as its basis:

SELECT lb.Branch\_Name,

COUNT(\*) AS Book\_Count,

COUNT(CASE WHEN bl.Returned\_date IS NOT 'NULL' THEN 1 END) AS Returned\_Books,

COUNT(CASE WHEN bl.Returned\_date IS 'NULL' THEN 1 END) AS Still\_Borrowed\_Books,

COUNT(CASE WHEN bl.Returned\_date IS NOT 'NULL' AND bl.Returned\_date > bl.Due\_Date THEN 1 END) AS Late\_Books

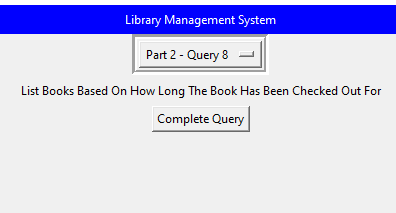
FROM LIBRARY\_BRANCH lb JOIN BOOK\_LOANS bl ON lb.Branch\_Id = bl.Branch\_Id

GROUP BY lb.Branch\_Name;

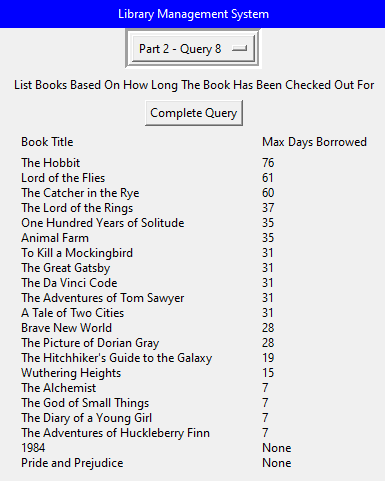
**Query 8:**

Objective: Select the max number of days a certain book was borrowed for.

Input: No Input



Output: List of all books with their longest loan instance in descending order



Brief Description: This query function creates a report by doing date calculations for each book that was loaned.

This function used the following query structure as its basis:

SELECT b.Title, MAX(CASE WHEN bl.Returned\_date = 'NULL' THEN NULL

ELSE CAST(JULIANDAY(bl.Returned\_date) - JULIANDAY(bl.Date\_Out) AS INTEGER)

END) AS Days\_borrowed

FROM BOOK b JOIN BOOK\_LOANS bl ON b.Book\_Id = bl.Book\_Id

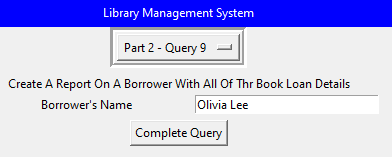
GROUP BY b.Book\_Id

ORDER BY Days\_borrowed DESC; -- Optional, but is included because it is easier to see the maximum number of days

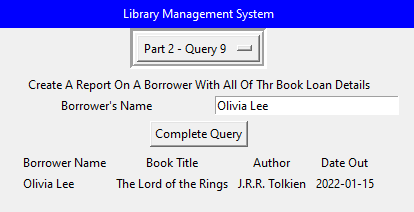
**Query 9:**

Objective: Select all books loaned by a borrower

Input: Borrower’s Name



Output: Olivia Lee checked out one book “The Lord of the Rings”



Brief Description: This query function takes the entry box input and matches it with the borrower’s ID to get the book tied to the ID in book loans, returning the Borrower’s name the book(s) they borrowed, the book’s author, and the date of check-out.

This function used the following query structure as its basis:

SELECT bo.Name, b.Title, ba.Author\_Name AS Author, bl.Date\_Out,

CASE WHEN bl.Returned\_date IS NOT 'NULL' THEN

CAST(JULIANDAY(bl.Returned\_date) - JULIANDAY(bl.Date\_Out) AS INTEGER)

ELSE 'NULL'

END AS Days\_borrowed,

CASE

WHEN bl.Returned\_date IS 'NULL' THEN 'Not Returned'

WHEN bl.Returned\_date IS NOT 'NULL' AND JULIANDAY(bl.Returned\_date) > JULIANDAY(bl.Due\_Date)

THEN 'Late' ELSE 'On Time'

END AS Return\_Status

FROM BOOK b JOIN BOOK\_LOANS bl ON b.Book\_Id = bl.Book\_Id

JOIN BORROWER bo ON bl.Card\_No = bo.Card\_No

JOIN BOOK\_AUTHORS ba ON b.Book\_Id = ba.Book\_Id

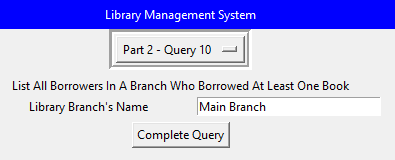
WHERE bo.Name = 'Ethan Martinez'

ORDER BY Date\_Out DESC;

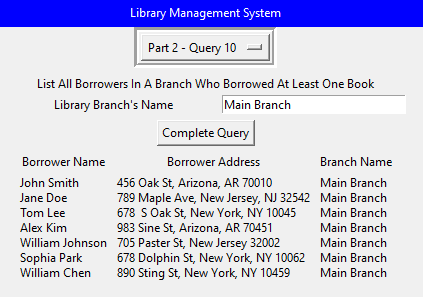
**Output 10:**

Objective: Select all borrowers at a certain branch

Input: Branch’s Name



Output: Main Branch had 7 borrowers



Brief Description: This query function takes the entry box input and prints out every borrower with an ID tied to the input branch.

This function used the following query structure as its basis:

SELECT DISTINCT bo.Name, bo.Address, lb.Branch\_Name

FROM BORROWER bo JOIN BOOK\_LOANS bl ON bo.Card\_No = bl.Card\_No

JOIN LIBRARY\_BRANCH lb ON bl.Branch\_Id = lb.Branch\_Id

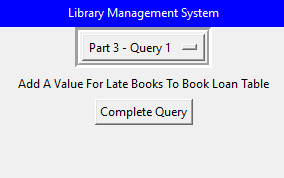
WHERE lb.Branch\_Name = 'West Branch';

## Phase 3 Queries

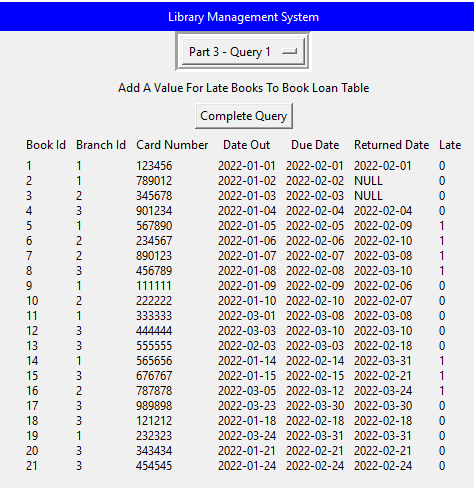
**Query 1:**

Objective: Alter table BOOK\_LOANS by adding a new column called ‘Late’ and Update the value to a boolean for lateness.

Input: No Inputs



Output: Late column added to far right - Boolean value of 1 for late books and 0 for on time returns or books not returned yet



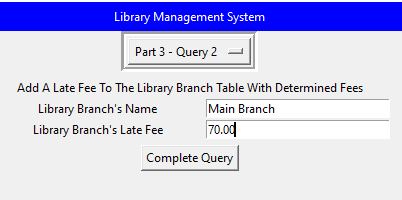
Description: This query takes no input and makes use of the “ALTER TABLE” statement – it creates a new column “Late” and fills it in with Boolean values marking it late or not late. The update logic ensures that loans are correctly marked as late based on return and due dates and the query function returns the updated book loans table.

*The structure of the query used for this function is described on page 5 of this document*

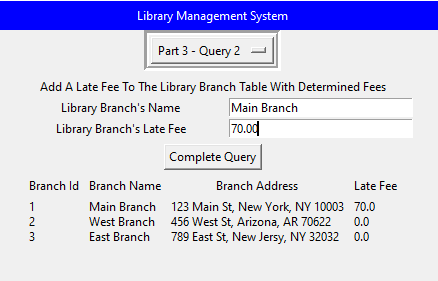
**Query 2:**

Objective: Alter table LIBRARY\_BRANCH by adding column “LateFee” and Update it to a certain amount

Input: Branch name and its desired Late fee value in dollars



Output: Late Fee column added – Main Branch’s fee is set to $70



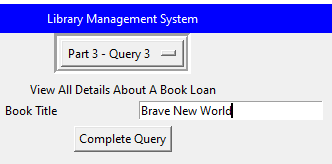
Description: This query function utilizes the “UPDATE” statement and SETs the value given by the user. The function ensures that the ‘Late Fee’ column exists in the ‘LIBRARY\_BRANCH’ table, takes the user’s input from the entry boxes, and then updates the late fee for the specified branch.

*The structure of the query used for this function is described on page 6 of this document*

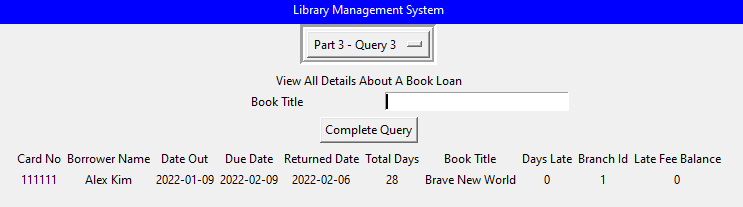
**Query 3:**

Objective: View all information about a specific book loan

Input: Name of the book to check



Output: Brave New World was loaned to Alex Kim



Description: This query function first creates an updated vBookLoanInfo view – specified to have the Card No, Borrower’s Name, check-out date of the book, due-date of the book, return date of the book, the number of days it was borrowed for, the book title, the number of days the return was late, the branch the book was from, and the incurred late fee if any – and uses the user input as the query filter for the view, and displays the result.

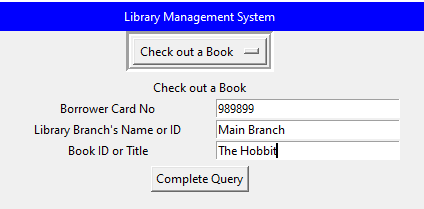
*The structure of the query used for this function is described on page 7 of this document*

## Phase 3 Requirements

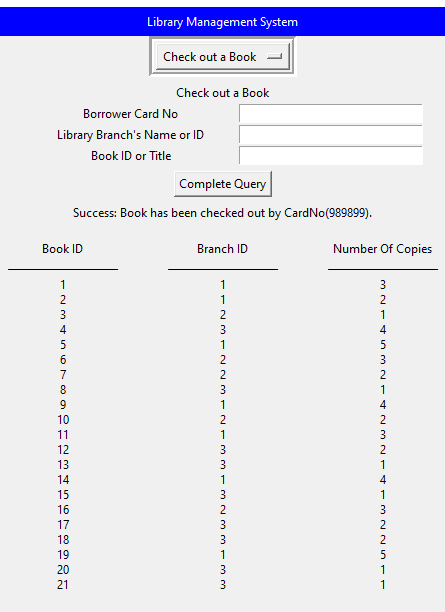
**Requirement 1:**

Objective: Check out a book from a branch

Input: Borrowers card number, name or ID of branch checked out from, title or ID of book to be checked out



Output: After a borrower checks out a book, the number of copies of that book is decreased from the respective branch it was borrowed from. In the case above, copies of “The Hobbit” decreased from 5 to 4 in the Main Branch after the borrower with card number 989899 loaned the book.



Description: This query takes in the input card number, name or ID of the branch, and the title or ID of the book. First, the program determines whether the name or ID of the branch was input, and it does the same to determine the book title or ID. Next, it validates if the book and library branch are entries in the database. Once validated, the program can then SELECT to find if the requested book is being held at the selected branch. Lastly, it ensures that the borrower has already checked out a copy of the book and has not returned it yet. After validation is complete, a new book loan entry is INSERTed and a TRIGGER command is used to decrease the number of book copies from the branch. Output is the borrower’s card number and a table showing the branch with one less copy of the selected book.

The following queries are what makes up the primary function of this requirement:

SQL Query (Replace the ? with your input for the query)

CREATE TRIGGER IF NOT EXISTS reduce\_book\_copy\_on\_branch

AFTER INSERT ON BOOK\_LOANS

BEGIN

UPDATE BOOK\_COPIES

SET No\_Of\_Copies = No\_Of\_Copies - 1

WHERE Branch\_Id = NEW.Branch\_Id

AND Book\_Id = NEW.Book\_Id;

END;

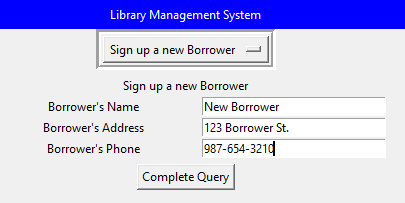
INSERT INTO BOOK\_LOANS (Book\_Id, Branch\_Id, Card\_No, Date\_Out, Due\_Date, Returned\_date)

VALUES (?, ?, ?, CURRENT\_DATE, DATE(CURRENT\_DATE, '+1 month'), 'NULL');

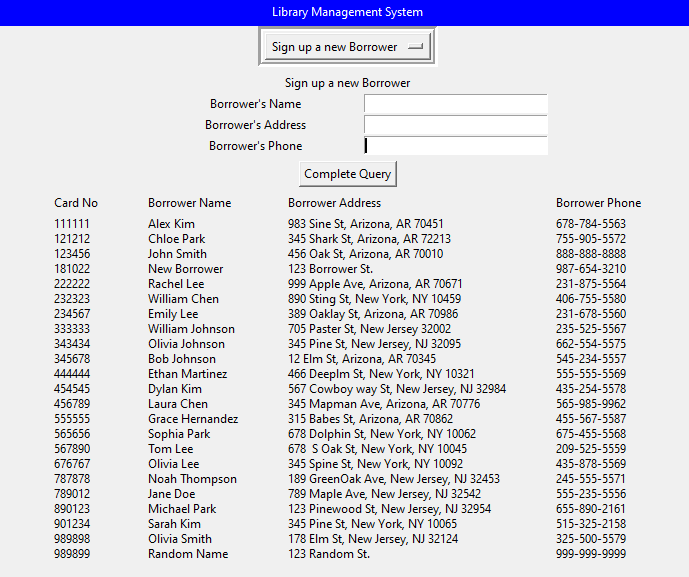
**Requirement 2:**

Objective: Add a new borrower to the database

Input: New borrower’s name, address, and phone number



Output: New Borrower added with automatically generated unique card number



Description: This query takes in the name, address, and phone number of the new borrower. Next, it automatically assigns a 6-digit card number and checks it against the database of existing borrowers’ card numbers to ensure uniqueness. It also checks if the borrower was already added to the system by checking the unique phone number against the existing phone numbers. Once it knows the borrower has not been added and their assigned card number is unique, an INSERT query is used to log them into the database. The updated BORROWER table with the new borrower is displayed as the output.

The following query is what makes up the primary function of this requirement:

SQL Query (Replace the ? with your input for the query)

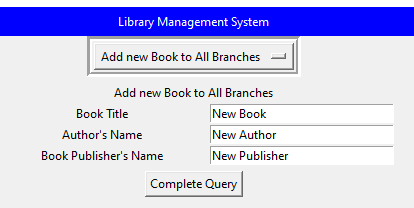
INSERT INTO BORROWER (Card\_No, Name, Address, Phone)

VALUES (?, ?, ?, ?);

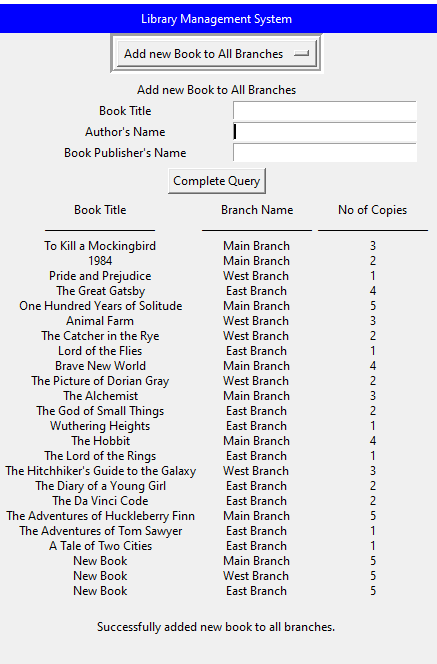
**Requirement 3:**

Objective: Add a new book to every branch

Input: New Book’s title, Author’s name, and Publisher’s name



Output: In the last three rows, New Book was added to every branch in the database



Description: This query takes in the inputs for the new book’s title, author’s name, and publisher’s name. If the book’s publisher does not exist in the database yet, an INSERT query adds it into the PUBLISHER table. It then INSERTs the book title along with the publisher's name into the BOOK table. To be able to INSERT the author into the AUTHORS table, a SELECT query must be performed first to retrieve the book’s ID. With this ID, the author’s name can be correctly INSERTed into the AUTHORS table. Lastly, every branch’s IDs are retrieved and used in an INSERT query to add 5 copies of the book to each branch. A joined table is displayed as the output to show the new book’s copies in the system.

The following queries are what make up the primary function of this requirement:

SQL Query (Replace the ? with your input for the query)

INSERT OR IGNORE INTO PUBLISHER (Publisher\_Name) VALUES (?)

INSERT INTO BOOK (Title, Publisher\_name) VALUES (?, ?)

SELECT Book\_Id FROM BOOK WHERE Title = ? AND Publisher\_name = ?

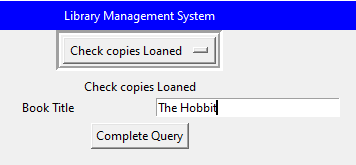
INSERT INTO BOOK\_AUTHORS (Book\_Id, Author\_Name) VALUES (?, ?)

INSERT INTO BOOK\_COPIES (Book\_Id, Branch\_Id, No\_Of\_Copies) VALUES (?, ?, ?);

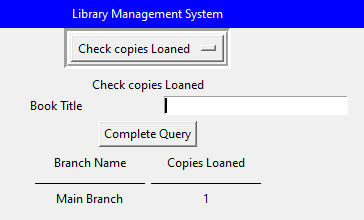
**Requirement 4:**

Objective: Select the number of copies loaned of a certain book

Input: Book Title to be searched



Output: The Hobbit was loaned once from Main Branch.



Description: This query takes in a single input for the book title. A Select query is used to retrieve the book’s ID. Once the ID is known, another SELECT query is used to retrieve data about every loan of the book. The branch name of where it was borrowed and the number of copies loaned are specified in the query output. Displayed is a joined table between book loans and library branch tables. Although not required, our query implementation also works with an input of the book’s ID instead of its title.

The following queries are what makes up the primary function of this requirement:

SQL Query (Replace the ? with your input for the query)

SELECT LB.Branch\_Name, COUNT(BL.Book\_Id) AS Copies\_Loaned

FROM BOOK\_LOANS BL JOIN LIBRARY\_BRANCH LB ON BL.Branch\_Id = LB.Branch\_Id

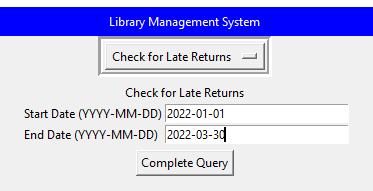
WHERE BL.Book\_Id = ? AND BL.Date\_Out IS NOT 'NULL'

GROUP BY LB.Branch\_Name;

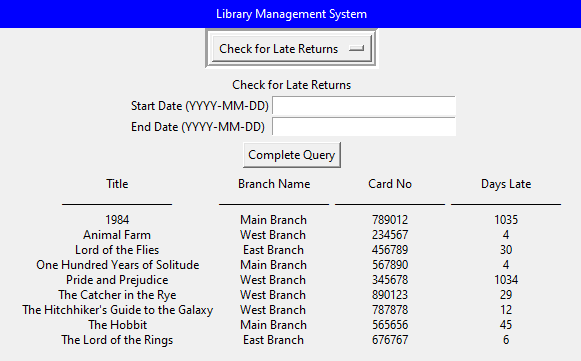
**Requirement 5:**

Objective: Select information on books loaned between two dates

Input: Start and End dates



Output: Between Jan 1, 2022, and March 30, 2022, 9 books were loaned.



Description: This query takes in the inputs Start and End dates. The program first finds all late or not returned books and calculates how many days they were late by. To filter out only the books that were loaned between the dates provided, the WHERE clause is needed. The output shows the book, the branch it was loaned from, the borrower’s card no, and the number of days the book was late.

The following queries are what makes up the primary function of this requirement:

SQL Query (Replace the ? with your input for the query)

SELECT B.Title, LB.Branch\_Name, BL.Card\_no,

CASE WHEN BL.Returned\_date IS NOT 'NULL' or NULL THEN

CASE WHEN JULIANDAY(BL.Returned\_date) - JULIANDAY(BL.Due\_Date) > 0 THEN

CAST(JULIANDAY(BL.Returned\_date) - JULIANDAY(BL.Due\_Date) AS INTEGER)

ELSE 'Not Late'

END

WHEN JULIANDAY(CURRENT\_DATE) > JULIANDAY(BL.Due\_Date) THEN

CAST(JULIANDAY(CURRENT\_DATE) - JULIANDAY(BL.Due\_Date) AS INTEGER)

ELSE 'Not Late'

END AS Days\_Late

FROM BOOK\_LOANS BL JOIN BOOK B ON BL.Book\_Id = B.Book\_Id

JOIN LIBRARY\_BRANCH LB ON BL.Branch\_Id = LB.Branch\_Id

WHERE BL.Due\_Date BETWEEN ? AND ?

AND (BL.Returned\_date IS 'NULL' OR CAST(JULIANDAY(BL.Returned\_date) > JULIANDAY(BL.Due\_Date) AS INTEGER) > 0)

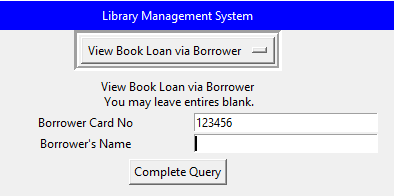
AND (Days\_Late != 'Not Late')

ORDER BY B.Title, LB.Branch\_Name;

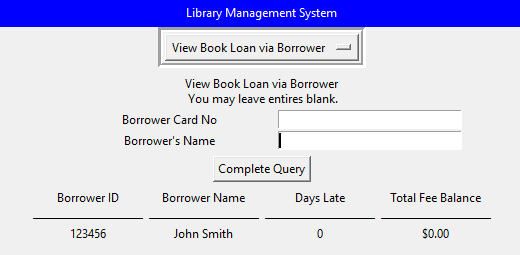
**Requirement 6a:**

Objective: View a borrower’s late books and fees

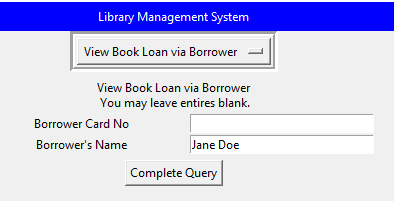
Input 1: Only Borrowers Card Number



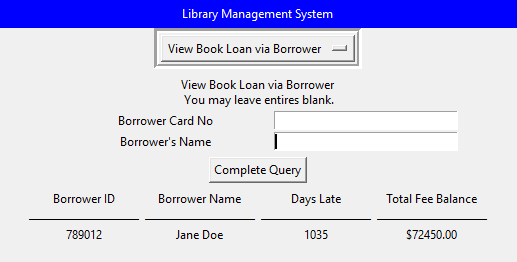
Output 1: John Smith (Card No: 123456) has no late books or fees



Description: This query can take no input, borrower Card No and/or borrower Name. The program then SELECTs borrowers who have book loans and calculates their respective late fees. Next, it determines whether the Card No or Name was inputted. Once this is determined, it filters out from the list of borrowers and late fees the requested borrower. Output is the borrower’s identification information, total days their loans were late, and a sum of all their late feesInput 2: Only Borrower’s Name

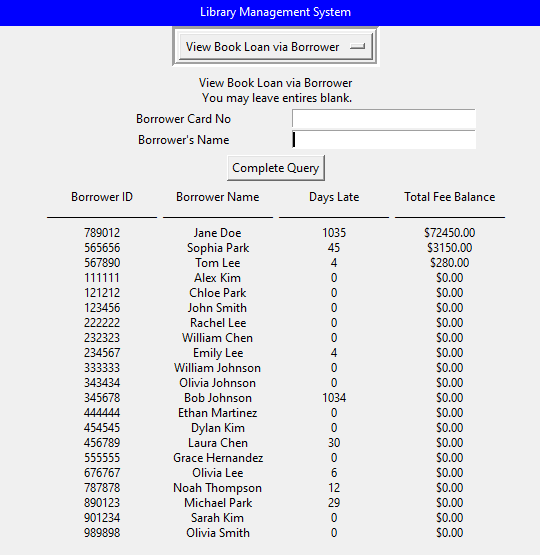


Output 2: Jane Doe (Card No: 789012) has a book late for 1035 days and owes $72K



Input 3: No Input

Output 3: Lists every loan with their respective late fees



SQL Query (Replace the ? with your input for the query):

SELECT BOR.Card\_No AS Borrower\_ID, BOR.Name AS Borrower\_Name,

    CASE

       WHEN BL.Returned\_Date IS NOT NULL THEN

           CASE

               WHEN JULIANDAY(BL.Returned\_Date) - JULIANDAY(BL.Due\_Date) > 0 THEN

                   CAST(JULIANDAY(BL.Returned\_Date) - JULIANDAY(BL.Due\_Date) AS INTEGER)

               ELSE 0

           END

       WHEN BL.Returned\_Date IS NULL AND JULIANDAY(CURRENT\_DATE) > JULIANDAY(BL.Due\_Date) THEN

           CAST(JULIANDAY(CURRENT\_DATE) - JULIANDAY(BL.Due\_Date) AS INTEGER)

           ELSE 0

   END AS Days\_Late,

   ROUND(SUM(CASE

       WHEN BL.Returned\_Date IS NOT NULL AND JULIANDAY(BL.Returned\_Date) - JULIANDAY(BL.Due\_Date) > 0 THEN

            (JULIANDAY(BL.Returned\_Date) - JULIANDAY(BL.Due\_Date)) \* LB.LateFee

       WHEN BL.Returned\_Date IS NULL AND JULIANDAY(CURRENT\_DATE) > JULIANDAY(BL.Due\_Date) THEN

            (JULIANDAY(CURRENT\_DATE) - JULIANDAY(BL.Due\_Date)) \* LB.LateFee

            ELSE 0

       END), 2) AS Total\_Fee\_Balance

   FROM BORROWER BOR

    JOIN BOOK\_LOANS BL ON BOR.Card\_No = BL.Card\_No

    JOIN LIBRARY\_BRANCH LB ON BL.Branch\_Id = LB.Branch\_Id

WHERE BOR.Card\_No = ? AND BOR.Name LIKE ?

GROUP BY BOR.Card\_No, BOR.Name

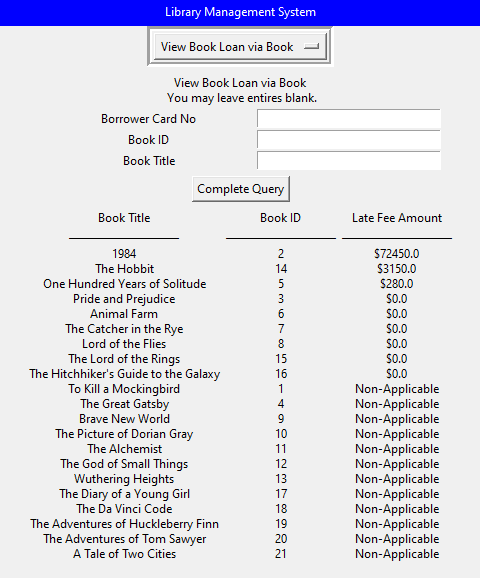
ORDER BY Total\_Fee\_Balance DESC;

**Requirement 6b**

Objective: View a Book’s late fees

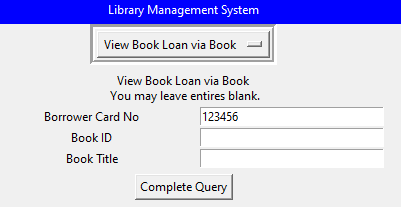
Input 1: No Inputs

Output 1: Lists every book and its respective highest late fee.

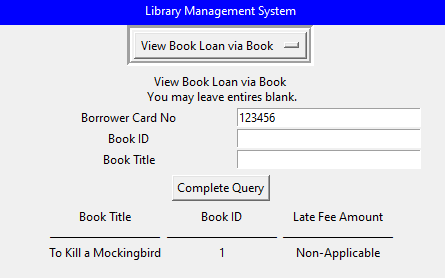


Description: This query takes either a borrower’s card no, a book’s ID, a book’s title, or a combination of any of the listed fields. If there is no input, the program will not apply a filter on the output. After receiving the input, the program retrieves all books with a sum of all their late fees. It then filters the output of this retrieval based on the input values. Output is a table with the requested book’s title, ID, and late fee.

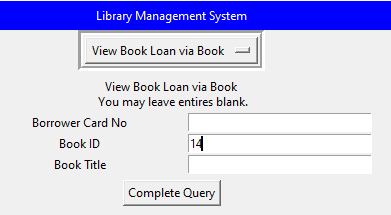
Input 2: Only Borrower’s Card Number



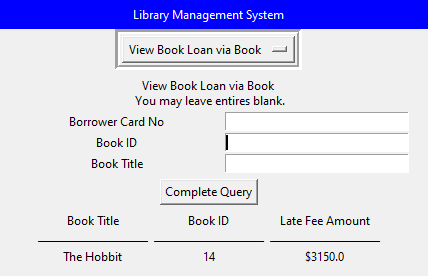
Output 2: John Smith (Card No: 123456) has one book loan “To Kill a Mockingbird” and has not returned the book yet.



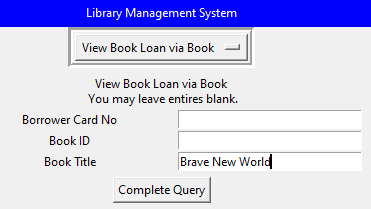
Input 3: Only Book ID



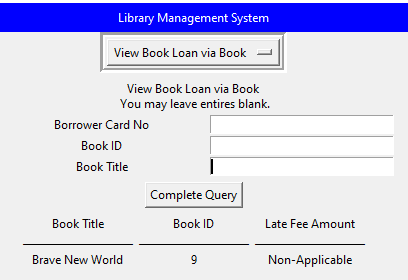
Output 3: “The Hobbit” (ID: 14) is past its due date and is worth $3k in fees.



Input 4: Only Book Title



Output: “Brave New World” (ID: 9) was loaned but has not been returned yet.



SQL Query (Replace the ? with your input for the query):

SELECT B.Title AS Book\_Title,

    BL.Book\_Id AS Book\_ID,

    CASE

        WHEN BL.Returned\_Date IS NOT 'NULL' OR NULL THEN

            CASE

                WHEN JULIANDAY(BL.Returned\_Date) - JULIANDAY(BL.Due\_Date) > 0 THEN

                    '$' || ROUND((JULIANDAY(BL.Returned\_Date) - JULIANDAY(BL.Due\_Date)) \* LB.LateFee, 2)

                ELSE 'Non-Applicable'

            END

        WHEN JULIANDAY(CURRENT\_DATE) > JULIANDAY(BL.Due\_Date) THEN

            '$' || ROUND((JULIANDAY(CURRENT\_DATE) - JULIANDAY(BL.Due\_Date)) \* LB.LateFee, 2)

        ELSE 'Non-Applicable'

    END AS Late\_Fee\_Amount

FROM BOOK B

    JOIN BOOK\_LOANS BL ON B.Book\_Id = BL.Book\_Id

    JOIN LIBRARY\_BRANCH LB ON BL.Branch\_Id = LB.Branch\_Id

WHERE BL.Card\_No = ? AND BL.Book\_Id = ? AND B.Title LIKE ?

ORDER BY

CASE

WHEN Late\_Fee\_Amount = 'Non-Applicable' THEN 0

END,

CAST(SUBSTR(Late\_Fee\_Amount, 2) AS FLOAT) DESC

## Design Philosophy And Difficulties While Implementing The GUI

We designed our GUI so that we could make our code modular from the top down as at the time of designing, we had an idea of what our Part 2 queries would look like, but no proper way to connect the two. This is partly why we decided to use a dropdown menu as it would allow us to perform each query independently. The other reason comes from another possibility: We could have designed a generalized table with all of the possible fields from all of the queries and displayed the table. The issue with that generalized approach was that we would need as many buttons as queries as well as need to check all of the fields to determine whether or not the correct information is being inputted into the query. While our approach to the GUI might have added another layer of complexity with the dropdown and the select button, it has made the implementation easier to work on without many merge conflicts as we all worked on different queries to implement.

One difficulty was time management which came from the other projects and plans that we had over Thanksgiving break. While we either had another major project for another class or had a ton of assignments between the week before and the week after Thanksgiving 2024, we managed to find the time to complete all of the queries and implement them into our GUI. While we did quickly finish the queries at the fundamental level, implementing them in the GUI was difficult as it took some time to work on and fix any bugs that we came across as unintended.

Another difficulty that we faced was the occasional merge conflicts and issues with the query implementation which were written by other people. Because there wasn’t one particular consistent way that we wrote the code as well as not enough details on what certain parts of the code do, it was hard to not only read the code but also track where is issue occurred. When using the GUI with Python, the only way that we could figure out the root of the issue was to use print statements and track what values from some variables or functions were causing the issue based on the error in the terminal. While it is not possible to fix this difficulty due to the nature of how we write code and our methodology for solving each problem, we did mitigate a good amount of the risk by modulating our code so that it would not affect the rest of the queries.

# CONCLUSION

This part of the project allowed us to create the Graphical User Interface (GUI) for the Library Management System Database, building upon the foundation of the work we did from Part 2 and integrating the further advanced functionalities of Part 3. Developing the GUI to be relatively user-friendly proved to be a challenge, but despite the learning curve, we believe the program can perform a variety of tasks for the hypothetical librarians and administrators effectively. By using the baseline of the queries, we developed in part 2, we designed the GUI that facilitates book checkouts, inventory management, and the generation of detailed reports about borrower information and late fees.

Developing the GUI helped us further develop our skills and understanding of database design and SQL querying, but it also allowed us to translate those skills into a hypothetically practical and interactive application. Throughout this project, we were met with several challenges, including data integrity issues, user input error issues, and query faults, but of all the challenges we faced, the most difficult challenge was group availability. As we reached the end of the semester, other courses were assigning projects due all around the same time, tests and quizzes were being held at the last minute, and each of us had differing schedules that made it hard to get together to work on this. Despite these challenges, we were able to overcome many of the issues we initially faced, and the final product is now a “robust and intuitive” Library Management System that can be used by a hypothetical library staff to manage their operations effectively.

# REFERENCES

## Part 2 References

* Elmasri, R., & Navathe, S. (2016). *Fundamentals of Database Systems Chapter 3 Slides* (7th ed.). Pearson.
* Elmasri, R., & Navathe, S. (2016). *Fundamentals of database systems* (7th ed.). Pearson.
* Guizani, N. (2024, September 5). *Live Lecture* [Live video recording].
* SQLite Foreign Key Support. (n.d.). [Www.sqlite.org](http://Www.sqlite.org). <https://www.sqlite.org/foreignkeys.html>
* SQLite Autoincrement. (n.d.). [Www.sqlite.org](http://Www.sqlite.org).

<https://www.sqlite.org/autoinc.html>

## Part 3 References

* Guizani, N. (2024, October 31). *Gui\_004.py* [Python code].
* With, M. (2019, April 30). Dropdown Menus With TKinter - Python Tkinter GUI Tutorial #18. YouTube.  
  <https://youtu.be/3E_fK5hCUnI>

# CONTRIBUTION LIST

## Part 2 SQL Queries & GUI Implementation

* Query 1 – Chime Nguyen
* Query 2 – Chime Nguyen
* Query 3 – Trung Nguyen
* Query 4-a – Trung Nguyen
* Query 4-b – Trung Nguyen
* Query 5 – Trung Nguyen
* Query 6 – Chime Nguyen
* Query 7 – Chime Nguyen
* Query 8 – Chime Nguyen
* Query 9 – Ivan Ko
* Query 10 – Ivan Ko

## Part 3 Report

* Introduction – Chime Nguyen
* Part 3 Query Results
  + Query 1 – Ivan Ko
  + Query 2 – Ivan Ko
  + Query 3 – Trung Nguyen
* Part 3 GUI Implementation – Chime Nguyen, Trung Nguyen, & Ivan Ko
* Conclusion – Chime Nguyen & Trung Nguyen
* Proofreading & Editing – Chime Nguyen, Ivan Ko, & Trung Nguyen

## Part 3 Readme

* Download the Code.zip and Databases – Trung Nguyen & Chime Nguyen
* Installing Python with Tkinter – Chime Nguyen
* Locating the Folder and Running the Program – Trung Nguyen
* Using The Program – Ivan Ko
* Example Workflow – Trung Nguyen
* Proofreading & Editing – Ivan Ko

## Part 3 SQL Queries & GUI Implementation

* Query 1 – Ivan Ko
* Query 2 – Ivan Ko
* Query 3 – Trung Nguyen

## Part 3 GUI Requirements

* Requirement 1 – Chime Nguyen & Trung Nguyen
* Requirement 2 – Ivan Ko
* Requirement 3 – Trung Nguyen
* Requirement 4 – Trung Nguyen
* Requirement 5 – Trung Nguyen
* Requirement 6 – Trung Nguyen