# 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. |

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 2

Fall 2024

Chime Nguyen

Ivan Ko

Trung Nguyen

# TABLE OF CONTENTS

I.) HONOR CODE ……………………………………………………………… 1

II.) INTRODUCTION ……………………………………………………………… 4

III.) DATA ……………………………………………………………………………… 5

III.) SQL TABLES CREATION ……………………………………………………… 12

III.) SQL QUERY RESULTS ……………………………………………………… 17

III.) CONCLUSION ……………………………………………………………… 29

IV.) CONTRIBUTION LIST ……………………………………………………… 30

a.) Report

b.) SQL Tables

c.) SQL Queries

V.) REFERENCES ……………………………………………………………… 32

# INTRODUCTION

The goal of this project as a whole is to demonstrate what we have learned 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. We then made some assumptions before translating the ER Diagram into a Database Schema. One of our major assumptions was to exclude an “Author” table because “Author\_name” was the only attribute associated with that table. Another assumption we had was that a table called “BORROWS\_FROM” would be required since it seemed like multiple foreign keys applied to this entity. In hindsight, we could have improved our ER Diagram in the aspects of how we linked the entities together, and our assumptions regarding the Author table could have been done differently. After completing part one, we solidified our understandings on ER diagrams and reassessed our assumptions in consideration of part two.

Part two of the project tasked us to implement the Library Management System Database in SQL. This documentation explains how we created the tables, imported data from the CSV files, and wrote queries to interact with the database, as well as any issues that we faced while implementing the tables and queries.

# DATA

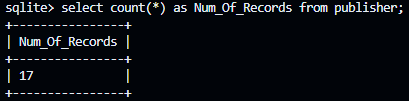
Upon opening SQLite, we first created the tables in the SQL TABLE CREATION section. Then, we have to use a few SQLite commands before we can import the data into our tables:

**.mode table** sets up the table in table format

**.separator ,** allows us to separate the data in the CSV files as they use commas for separating data.

From there, the data below is the kind of data that we are working with for our database, excluding any insertions that came from the queries themselves.

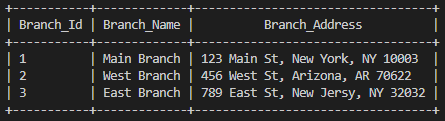
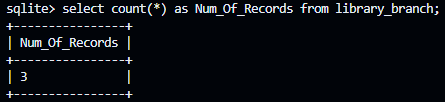
## Publisher



**Import Command**: **.**import /Path/To/File/Publisher.csv PUBLISHER --skip 1

Skipping the first line of the CSV file, the table imports and stores information about the book publishers, including their Publisher Name, Phone Number, and Address.

## Library Branch

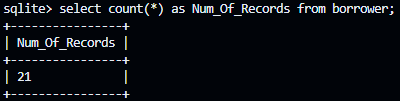
  


**Import Command**: **.**import /Path/To/File/Library\_Branch.csv LIBRARY\_BRANCH --skip 1

Skipping the first line of the CSV file, the table imports and stores information about different library branches, including their Branch ID, Branch Name, and Branch Address.

## Borrower

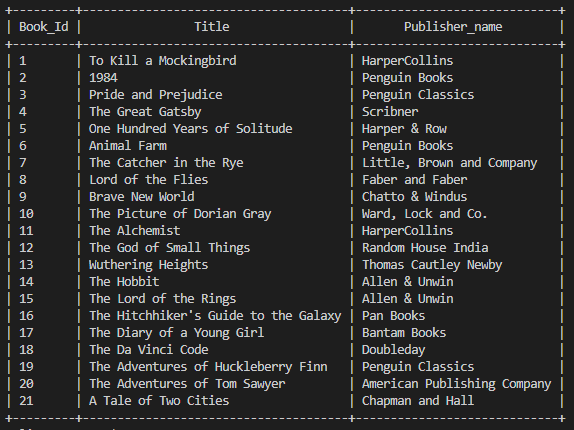
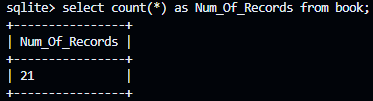
A screenshot of a computer

Description automatically generated  


**Import Command**: **.**import /Path/To/File/Borrower.csv BORROWER --skip 1

Skipping the first line of the CSV file, the table imports and stores information about library card holders by their Card Number, Name, Address, and Phone Number.

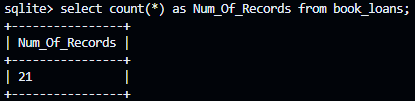
## Book

  
  
Import Command: **.import /Path/To/File/Book.csv BOOK --skip 1**

Skipping the first line of the CSV file, the table imports and stores information about books by their Book ID, Title, and Publisher Name. This is the central table to the database regarding all book information.

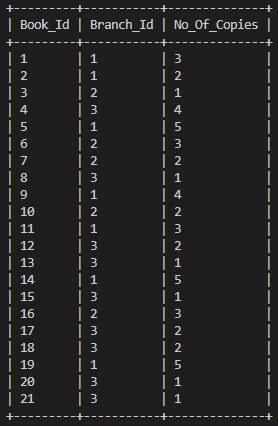
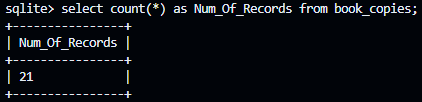
## Book Loans

A black screen with numbers and numbers

Description automatically generated  
  
Import Command: **.import /Path/To/File/Book\_Loans.csv BOOK\_LOANS --skip 1**

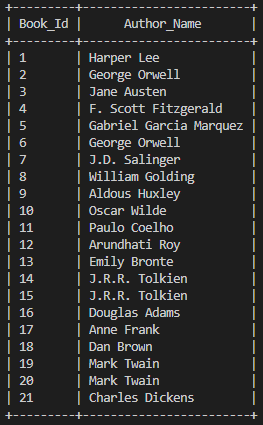
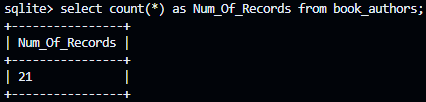
Skipping the first line of the CSV file, the table imports and stores records of book loan transactions. It holds information regarding which book was borrowed from which branch, by whom, and its key dates (check-out, due date, and return date). This is the central table of the database that ties all tables together.

## Book Copies

  
  
Import Command: **.import /Path/To/File/Book\_Copies.csv BOOK\_COPIES --skip 1**

Skipping the first line of the CSV file, the table imports and tracks the number of copies of each book available at different library branches by their Book ID, Branch ID and Number of Copies.

## Book Authors

  
  
Import Command: **.import /Path/To/File/Book\_Authors.csv BOOK\_AUTHORS --skip 1**

Skipping the first line of the CSV file, the table imports and stores the relationship between books and their authors by their Book ID and Author name, thus allowing multiple authors per book and vice versa.

Each Table is designed to work together in a relational database, with foreign keys connecting related information across tables. The table creation and key data relations are described in the following section of the report.

# SQL TABLES CREATION

The following SQL tables below demonstrate how we created our tables, as well as describe why certain values are of certain types, are certain keys, and have certain constraints.

## PUBLISHER Table

|  |
| --- |
| CREATE TABLE IF NOT EXISTS PUBLISHER (  -- Attributes  Publisher\_Name VARCHAR(27) NOT NULL,  Phone CHAR(12) NOT NULL,  Address VARCHAR(120) NOT NULL,  -- Primary Key & Secondary Keys  PRIMARY KEY (Publisher\_Name)  ); |

The reason the attribute Phone is using type CHAR is because of how long a phone number can typically be. Both Address and Publisher\_Name are using type VARCHAR since the length of the attribute can vary greatly and there is no minimum or maximum size of the string.

The primary key for the PUBLISHER table is the Publisher\_Name as there can be only one name per publisher and it is strongly related to the PUBLISHER table. Additionally, the Publisher\_Name is being used in the BOOK table, which highly suggests that the Publisher\_Name is the primary key. The Phone attribute is considered to be a unique key as the attribute closely ties with the PUBLISHER table but is not as strong of a key as the Publisher\_Name.

None of the attributes should be empty or NULL, and in the case that an insertion were to happen, it would be necessary to add the string ‘NULL’ to the queries.

## LIBRARY\_BRANCH Table

|  |
| --- |
| CREATE TABLE IF NOT EXISTS LIBRARY\_BRANCH (  -- Attributes  Branch\_Id INTEGER PRIMARY KEY NOT NULL,  Branch\_Name VARCHAR(11) NOT NULL,  Branch\_Address VARCHAR(120) NOT NULL  ); |

The reason the attribute Branch\_Id is using a type of INT called type INTEGER is because it is easier to track the Id in terms of an integer number, Both Branch\_Name and Branch\_Address uses type VARCHAR as the length of either name or address can be lengthy. In addition, INTEGER has the unique property that if combined with the keyword PRIMARY KEY, it would allow the query to automatically increment or decrement when inserting or deleting a value.

The primary key for the LIBRARY\_BRANCH table is the Branch\_Id, as it is primarily the only possible unique thing about the branch itself. There can be branches with the same name, or have multiple branches in the same address, and it is safer to use the Id as it makes every row unique.

None of the attributes should be empty or NULL as the Id will automatically increment or decrement with the primary key, and both the name and address are required to identify the branch.

## BORROWER Table

|  |
| --- |
| CREATE TABLE IF NOT EXISTS BORROWER (  -- Attributes  Card\_No INTEGER PRIMARY KEY NOT NULL,  Name VARCHAR(64) NOT NULL,  Address VARCHAR(120) NOT NULL,  Phone CHAR(12) NOT NULL,  -- Primary Key & Secondary Keys  UNIQUE (Phone)  ); |

The BORROWER table stores data for book borrowers at a library branch. “Card\_No” is an integer value that uniquely identifies all borrowers making it a valid candidate for the primary key. Borrower names are stored in the VARCHAR variable “Name” which is specified as NOT NULL since all borrowers must have a name. Similarly, “Address” is another VARCHAR variable that stores the borrower’s home address. Lastly, Phone is of type CHAR that is fixed to store the borrower’s phone number in the format XXX-XXX-XXXX. Phone is also specified as unique since all phone numbers differ in digits. Between the unique attributes, Card\_No and Phone, we decided that Card\_No would be the better primary key as it relates more to the context of a library system.

## BOOK Table

|  |
| --- |
| CREATE TABLE IF NOT EXISTS BOOK (  -- Attributes  Book\_Id INTEGER PRIMARY KEY NOT NULL,  Title VARCHAR (128) NOT NULL,  Publisher\_name VARCHAR(64) NOT NULL,  -- Foreign Keys & Foreign Key Constraints  FOREIGN KEY (Publisher\_name) REFERENCES PUBLISHER (Publisher\_Name)  ); |

To store every book in the Library Management system, we have the BOOK table. Every book is uniquely defined by the INTEGER variable “Book\_Id.” Another attribute is the “Title” which is a NOT NULL VARCHAR variable to store the book’s name. Next, we have the “Publisher\_name” which is a place to store the book’s publishing company’s name. This value is a foreign key referenced from the PUBLISHER table. Since “Book\_Id” is the only unique attribute to identify books, we made it the primary key for this table.

## BOOK\_LOANS Table

|  |
| --- |
| CREATE TABLE IF NOT EXISTS BOOK\_LOANS (  -- Attributes  Book\_Id INTEGER NOT NULL,  Branch\_Id INTEGER NOT NULL,  Card\_No INTEGER NOT NULL,  Date\_Out TEXT NOT NULL,  Due\_Date TEXT NOT NULL,  Returned\_date TEXT NOT NULL,  -- Primary Key & Secondary Keys  PRIMARY KEY (Book\_Id, Branch\_Id, Card\_No),  -- Foreign Keys & Foreign Key Constraints  FOREIGN KEY (Book\_Id) REFERENCES BOOK (Book\_Id)  ON UPDATE CASCADE ON DELETE CASCADE,  FOREIGN KEY (Branch\_Id) REFERENCES LIBRARY\_BRANCH (Branch\_Id)  ON UPDATE CASCADE ON DELETE CASCADE,  FOREIGN KEY (Card\_No) REFERENCES BORROWER (Card\_No)  ON UPDATE CASCADE ON DELETE CASCADE  ); |

The foreign keys are of type INTEGER because of their respective primary keys mentioned earlier in this section. The reason the dates are of type TEXT instead of type DATE is because SQLite and regular SQL can perform date calculations on TEXT as long as they are in YYYY-MM-DD format, which in our data for BOOK\_LOANS, we have the correct format. In addition, the Returned\_date has the string ‘NULL’, restricting us to use TEXT as it would not be in the right format for the DATE type.

The foreign keys are the Book\_Id, the Branch\_Id, and the Card\_No as they are originally primary keys from the BOOK, LIBRARY\_BRANCH, and BORROWER tables, respectively.

The primary key is the combination of all three of the foreign keys together as they were once foreign keys and the other attributes (Date\_Out, Due\_Date, Returned\_date) do not provide the relevant information to connect each book, branch, and person who loaned any particular book.

None of the attributes are allowed to be NULL based on our implementation. However, since Returned\_date has the string ‘NULL’, this can be used as a placeholder for any NULL values.

## BOOK\_COPIES Table

|  |
| --- |
| CREATE TABLE IF NOT EXISTS BOOK\_COPIES (  -- Attributes  Book\_Id INTEGER NOT NULL,  Branch\_Id INTEGER NOT NULL,  No\_Of\_Copies INT NOT NULL,    -- Primary Key & Secondary Keys  PRIMARY KEY (Book\_Id, Branch\_Id),  -- Foreign Keys & Foreign Key Constraints  FOREIGN KEY (Book\_Id) REFERENCES BOOK (Book\_Id)  ON UPDATE CASCADE ON DELETE CASCADE,  FOREIGN KEY (Branch\_Id) REFERENCES LIBRARY\_BRANCH (Branch\_Id)  ON UPDATE CASCADE ON DELETE CASCADE  ); |

The BOOK\_COPIES table uses INTEGER for Book\_ID, Branch\_Id, and No\_Of\_Copies attributes. The foreign keys are of type INTEGER because of their respective primary keys mentioned earlier in this section. No\_Of\_Copies is also INTEGER as it represents a whole number count of book copies.

The Primary Key for this table is a combination of Book\_Id and Branch\_Id, as this uniquely identifies each book-branch combination. This key ensures that each book at each branch is a unique entry. Both Book\_ID and Branch\_Id are foreign keys that reference BOOK and LIBRARY\_BRANCH tables, respectively.

All attributes in this table are marked with NOT NULL, to ensure that each field has a value. We made this design choice to ensure incomplete records with faulty Insertions. We did come across queries that may need a placeholder “NULL” string to circumvent the NOT NULL as it stores it as TEXT, but this design would typically require all three fields to be provided with valid values.

We utilized the ON UPDATE CASCADE and ON DELETE CASSCADE clauses on the foreign key constraints to ensure that any changes or deletions in the referenced tables, in this case would be BOOK and LIBRARY\_BRANCH, are automatically modified in the BOOK\_COPIES table.

## BOOK\_AUTHOR Table

|  |
| --- |
| CREATE TABLE IF NOT EXISTS BOOK\_AUTHORS (  -- Attributes  Book\_Id INTEGER NOT NULL,  Author\_Name VARCHAR(30),  -- Primary Key & Secondary Keys  PRIMARY KEY (Book\_Id, Author\_Name),  -- Foreign Keys & Foreign Key Constraints  FOREIGN KEY (Book\_Id) REFERENCES BOOK(Book\_Id)  ON UPDATE CASCADE ON DELETE CASCADE  ); |

The BOOK\_AUTHORS table uses type INTEGER for the Book.Id attribute and VARCHAR(30) for the Author\_Name. The foreign key is of type INTEGER because of their respective primary keys mentioned earlier in this section. The Author\_Name is VARCHAR(30), with an arbitrary 30 value, to accommodate varying author name lengths- the longest name within the CSV file had 22 characters.

The primary key for the table is a composite key consisting of both Book\_Id and Author\_Name. This combination uniquely identifies each book-author relationship, allowing multiple authors per book and vice versa. The Book\_Id, marked with NOT NULL, serves as a foreign key by referencing the BOOK table, which forms the relationship between books and their authors by the tables.

Similarly to all tables above, the ON UPDATE CASCAE and ON DELETE CASCADE clauses ensure that any changes or deletions in the referenced tables are automatically reflected in the BOOK\_AUTHORS table.

# SQL QUERY RESULTS

The queries below have the description of each query, the entire code for the SQL query with the comments removed and display the results below, along with a description for the query, the issues, if any, that we had while making the query, and the results of the query. In addition, there will be two screenshots showing before and after for questions 1 through 4b, while questions 5 through 10 will have one screenshot that was taken before questions 1 through 4b.

## Question 1

### Insert yourself as a New Borrower. Do not provide the Card\_no in your query.

As the person who created this query, Chime Nguyen, I decided to insert myself as a borrower while only inserting the name, address, and phone number. For the query to work, I had to specify which attributes that I wanted to insert as the first attribute starts with the primary key Card\_No.

While it worked initially without having to include the INTEGER PRIMARY KEY in the BORROWER table, Question 4-a gave us a bit of an issue as you will see in in Question 4-a. In the end, it was easier to use INTEGER along with the PRIMARY KEY keywords together as it automates the increment for us, which solves the issue of not needing to provide the Card\_No attribute. Below is the query and the results of our query.

|  |
| --- |
| INSERT INTO BORROWER (Name, Address, Phone)  VALUES ('Chime Nguyen','701 S Nedderman Dr, Texas, TX 76019', '211-211-2112'); |

**Before Query** **After Query**

A screenshot of a computer screen

Description automatically generated A screenshot of a computer screen

Description automatically generated

The only one record was affected in the query is the insertion of the name ‘Chime Nguyen’, the address ‘701 S Nedderman Dr, Texas, TX 76019', and the phone number ‘211-211-2112’, which is at the bottom of the query. We did not insert a card number in our query, and our table autoincremented the card number by one from the last card number at the bottom of the table.

## Question 2

### Update your phone number to (837) 721-8965.

This question is a direct continuation of question 1, except this time, we have to update our phone number to a specific phone number. Instead of using the INSERT clause, we chose to op for the UPDATE clause where we update the BORROWER table and set the phone number to the phone number ‘(837) 721-8965’ to the name ‘Chime Nguyen’. While the phone number itself appears to be fine, it does not fit the format of how the phone numbers are lay out in the BORROWER database. By changing the phone number from ‘(837) 721-8965’ to ‘837-721-8965’ in the SET clause, we no longer have to worry about inserting the phone number into the query.

|  |
| --- |
| UPDATE BORROWER  SET Phone = '837-721-8965'  WHERE Name = 'Chime Nguyen'; |

**Before Query**  **After Query**

A screenshot of a computer screen

Description automatically generated A screenshot of a computer

Description automatically generated

The before query is the same as the after query in question 1. The changes that happened between the before and after queries is the row with the yellow box around the changed phone number. Between the two queries, the phone number ‘211-211-2112’ was changed to ‘837-721-8965’. One record was affected.

## Question 3

### Increase the number of book\_copies by 1 for the 'East Branch'.

Update: +1 to all East Branch’s book copies

First, we specify the table BOOK\_COPIES for updating and SET its column No\_Of\_Copies to the original value + 1. To select only the book copies from East Branch, we nested a SELECT query inside the WHERE clause.

|  |
| --- |
| 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'  ); |

**Before Query** **After Query**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

For the convenience of reading the before and after queries, the data that would be changed by the query in the before query screenshot is underlined in red while the changes in the after-query screenshot will have a yellow box. Nine records were affected.

## Question 4-a

### Insert a new BOOK with the following info: Title: 'Harry Potter and the Sorcerer's Stone' ; Book\_author: 'J.K. Rowling' ; Publisher\_name: 'Oxford Publisheing'

To insert a new BOOK, this question requires three different INSERT operations to add related data across multiple tables in the database in order to maintain data integrity. The first INSERT operation is for a new publisher “Oxford Publisheing” (note the typo here – we followed the question description and matched the typo) into the PUBLISHER table. We needed to add the Publisher first so that it could be reflected and linked in a relationship with the book getting added in the next INSERT operation. Because only the publisher name was given, we opted to put placeholder ‘NULL’ text values for the Phone and Address attributes since we have the NOT NULL constraints on. Next, the INSERT operation inserts a book with the title “Harry Potter and the Sorcerer’s Stone”, and “Oxford Publisheing” as the Publisher, linking the publisher we just added. Finally, we INSERT into BOOK\_AUTHORS an Author to the newly added book record, linking an Author to the Book set as “J.K. Rowling”. The statement uses a query within the VALUES portion of the operation to find the Book\_Id of the newly inserted book to link to the correct record.

An issue that we came across was how we were unable to insert the publisher name directly into the BOOK table as there is no on insert constraint on the foreign key that would do it for us. The way we handled it is by adding the required key attribute publisher name into the PUBLISHER table first before adding the rest of the required attributes to the BOOK table and then the BOOK\_AUTHORS table.

For inserting the book ID into the BOOK\_AUTHORS table without knowing which ID it is from the BOOK table, we decided to select the book ID from the BOOK table that has the title of the book.

Another issue that we had was how the primary key for the book ID did not automatically increment despite that the attribute was a primary key. The way we solved that issue was by using the keywords INTEGER and PRIMARY KEY together in the attribute itself instead of having INT for the attribute type and separating the PRIMARY KEY on a separate line in the table. While the AUTOINCREMENT keyword in the reference was not something that we have learned in class, the background section of the website helped gave us the idea that we needed to combine both INTEGER and PRIMARY KEY into one phrase.

|  |
| --- |
| 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'); |

**Before Query After Query**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

For the sake of readability, the two queries are on two different pages and highlighted the changes between the two tables after inserting the three queries. All of the issues that we had were addressed earlier.

## Question 4-b

### You also need to insert the following branches:

#### +----------------+------------------------------------------+

#### |North Branch |456 NW, Irving, TX 76100 |

#### +----------------+------------------------------------------+

#### |UTA Branch |123 Cooper St, Arlington TX 76101|

#### +----------------+------------------------------------------+

The request above asks to insert new entries into the columns “Branch\_Name” and “Branch\_Address” which are associated with the LIBRARY\_BRANCH table. For our INSERT queries, the structure is the same for both of the new entries. We specify the table and attributes of the table and use the VALUES clause to insert the data for each column.

|  |
| --- |
| 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'); |

**Before Query After Query**

A computer screen shot of a branch

Description automatically generated A computer screen shot of a branch

Description automatically generated

The branches that got added in the after query are in the yellow box. Only two records were affected.

## Question 5

### Return all Books that were loaned between March 5, 2022 until March 23, 2022. List Book title and Branch name, and how many days it was borrowed for.

The question asks for a report of all books loaned between March 5 and March 23 of 2022 by Book Title and Library Branch name as well as the number of days it was borrowed for. To get this record, we SELECT books by their Title from the BOOK table, the library branch by their branch name from the LIBARY\_BRANCH table, and perform a date subtraction to get the number of days a book was borrowed for. The CASE statement calculates the number of days by first checking if the Return Date from the BOOK\_LOANS table is ‘NULL’; if the value is not ‘NULL’ then it calculates the difference between Return\_date and Date\_Out - if the Return\_date is found to be ‘NULL’, the book is not returned yet, and so it calculates the difference between the current date and Date\_Out.

Here, we used the JUILIANDAY function to convert the text dates in the format of YYYY-MM-DD into Julian day numbers for the date difference operation and cast it as an INTEGER to get a whole number count of days. An issue we came across while handling this query was the date performing the difference operation. At first, we had set it to just subtract Return\_date and Date\_Out, but would get a result of 0. We then tried casting them to DATE types, thinking it was a TEXT type issue, but again got a result of zero. We solved this issue by using the JULIANDAY function, which converts dates and times into Julian day numbers – a representation of days since noon Universal Time on November 24, 4714 BC (the start of the Julian calendar), allowing us to calculate the date difference. The values we got by subtracting the two Julian dates returned at numbers with a decimal value of .0 so we casted the result to INTEGER to truncate the additional .0, giving us a whole number.

The FROM and JOIN clauses join the BOOK\_LOANS table with BOOK and LIBRARY\_BRANCH using their respective foreign keys. We filtered the books using the WHERE clause to filter loans where Date\_Out is between March 5, 2022, and March 23, 2022. Finally, we sort the results by Book Title and Branch Name with an ORDER BY clause.

|  |
| --- |
| 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; |

A black and white screen with white text

Description automatically generated

The result of this query shows the title, branch name, and the number of days that particular book has been booked out for. If the dates were changed, such that books that do not have a return date is included, those books with a ‘NULL’ return date will show up as an extremely large number of days that the book has been borrowed. In addition, using CURRENT\_DATE will take the current date and get the difference between it and the date out, which explains such a large number. Two records were affected.

## Question 6

### Return a List borrower names, that have books not returned.

This query takes all of the names from the BORROWER table and finds any people who have not returned at least one book in the BOOK\_LOANS table. Here, the keyword DISTINCT is used so that any repeat names will only show up once. Below is the query and the results of our query.

|  |
| --- |
| 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'; |

A black background with white text

Description automatically generated

This is the Borrower Names table of people who have yet to return their books back to the library. If we check this against BOOK\_LOANS and the BORROWER tables for those names, their card numbers, and the ‘NULL’ books, they both do show up as people who haven’t turned in a borrowed book from the library. It may have been more beneficial to include the card number so that the person who queried this could quickly cross reference the tables to see if this data is accurate, but for the sake of the question, this is accurate. Two records were affected.

## Question 7

### Create a report that will return all branches with the number of books borrowed per branch separated by if they have been returned, still borrowed, or late.

In this query, we were asked to retrieve the branch name, the number of books borrowed per branch, and then the number of returned, still borrowed, and late books. Retrieving the branch name is simple, getting the number of books borrowed per branch was also easy, but the other three values to select posed more of a challenge.

In order to count the number of returned, borrowed, and late books, it required CASEs where the return date is not null, is null, and is both not null and greater than the due date respectively. At first, I interpreted the ‘NULL’s in the data as actual NULL values and not the string representation of NULL, thus I ran into some issues where it wouldn’t print anything until I added the single quotes around NULL in my CASEs. JOINing the LIBRARY\_BRANCH and the BOOK\_LOANS and GROUPing BY the branch name was also doable. In each of the cases, if a book met the case when condition, then it would add one to the count before ending the case. Below is the full query and the results below.

|  |
| --- |
| 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; |

A black and white text

Description automatically generated with medium confidence Here, the Book Count or the total number of books that were loaned out by each branch is shown in each branch. The number of Returned Books in this case is all of the books that were returned, regardless of if it was late or if it was turned in on time or early. This was later confirmed by our professor Guizani, who mentioned that she preferred if our data included the number of late books in this column. The Still Borrowed Books are the number of books that do not have any return date and instead has the string ‘NULL’ in the data. If we check this against question 6 and the data for the BOOK\_LOANS table, we can see that both the Main and West Branches both have one ‘NULL’ return date, which is reflected in this column. The Late Books column is based on the number of books that were turned in past the due date. Three records were affected.

## Question 8

### List all the books (title) and the maximum number of days that they were borrowed.

Retrieve: Book Title, Max days borrowed

To satisfy the needed retrievals, we select Title from the BOOK table and a MAX function over the Days\_borrowed, an alias for the number of days the book was borrowed. Nested within the MAX function, the exact number of days to borrowed is calculated by Returned\_date (day returned) minus Date\_out (day borrowed). JULIANDAY keyword is needed to perform the subtraction on dates. A CASE statement is needed in the MAX function to specify for cases when the Returned\_date is NULL, meaning the book has not been returned. For NULL cases, the days borrowed remain NULL since the calculation is not possible.

Since the query spans multiple tables, the tables BOOK and BOOK\_LOANS are joined on their shared attribute Book\_Id. Results are grouped by Book\_Id so that number of days borrowed is for a specific book instead of having it applied to multiple copies of the same book. Although not specified by the requirements, the results are displayed in descending order to show the MAX days borrowed for all books.

|  |
| --- |
| 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; |

A screen shot of a computer

Description automatically generated

The result table displays the title of every book borrowed and the number of days they were out. Some entries, such as books “1984” and “Pride and Prejudice,” have blanks for their Days\_borrowed because they are not returned yet and the value could not be calculated. In total, there are 21 entries in the result table.

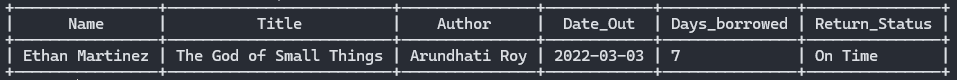
## Question 9

### Create a report for Ethan Martinez with all the books they borrowed. List the book title and author. Also, calculate the number of days each book was borrowed for and if any book is late being returned. Order the results by the date\_out.

Retrieve: Ethan Martinez’s books borrowed (title, author, number of days borrowed, late books) and order by date\_out

For the SELECT, we select the “Name”, “Author\_Name”, and book “Title” from BORROWER, BOOK\_AUTHORS and BOOK respectively. In addition to these selections, we calculate the number of days borrowed with the JULIANDAY operation applied to the subtraction between the returned\_date and date\_out. The CASE statement is needed since returned\_date can be NULL which will cause an error with the arithmetic operation. Lastly, we also determined if a book was returned late by comparing if the returned\_date is past the due\_date. A CASE to check if returned\_date is NULL is also required here for the arithmetic. Since multiple tables are necessary for this query, we JOIN all the tables needed with their shared attributes. To get only books borrowed by Ethan Martinez, the WHERE clause only chooses those books that were borrowed by him. Finally, we ORDER BY Date\_out descending in accordance with the requirement.

|  |
| --- |
| 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; |

 As seen above, the columns match the required attributes to be retrieved by the query. Days\_borrowed represents the number of the days Ethan Martinez borrowed a book while Return\_status represents if he turned it in late. From the results, Ethan Martinez only borrowed one book called ”The God of Small Things”.

## Question 10

### Return the names of all borrowers that borrowed a book from the West Branch include their addresses.

Retrieve: Name and Address of borrowers from West Branch

In the SELECT clause, we select the NAME and ADDRESS from the BORROWER table and the Branch\_Name from the LIBRARY\_BRANCH table. Since BORROWER and LIBRARY\_BRANCH do not share a similar attribute, the table BOOK\_LOANS is needed to join all three tables. Lastly, the WHERE clause only filters those borrowers who borrowed from “West Branch.”

|  |
| --- |
| 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'; |

A address sign with white text

Description automatically generated Shown above are the results of this query with columns for borrower names, their addresses, and the Branch they borrowed from. The query asked for West Branch borrowers, so all entries’ branch is the West Branch. In total, there are 5 entries or 5 people who borrowed from West Branch.

# CONCLUSION

This portion of the project allowed us to create the entire Library Management System Database using the provided data, including us importing the data into our created tables and performing requested queries on the database itself. In the DATA section, we described the data used in the database as well as the commands necessary to load them from the CSVs into the SQL tables. The SQL TABLES CREATION chapter explains our designs for each entity table with our reasonings for specific data types and key identification. Lastly, SQL QUERY RESULTS contains entries for each query with explanations on their structure and results.

# CONTRIBUTION LIST

## Report

* Introduction – Chime Nguyen & Ivan Ko
* Data – Trung Nguyen
* SQL Tables Creation
  + PUBLISHER Table – Chime Nguyen
  + LIBRARY\_BRANCH Table – Chime Nguyen
  + BORROWER Table – Ivan Ko
  + BOOK Table – Ivan Ko
  + BOOK\_LOANS Table – Chime Nguyen
  + BOOK\_COPIES Table – Trung Nguyen
  + BOOK\_AUTHORS Table – Trung Nguyen
* SQL Query Results
  + Question 1 – Chime Nguyen
  + Question 2 – Chime Nguyen
  + Question 3 – Trung Nguyen
  + Question 4-a – Trung Nguyen & Chime Nguyen
  + Question 4-b – Trung Nguyen
  + Question 5 – Trung Nguyen & Chime Nguyen
  + Question 6 – Chime Nguyen
  + Question 7 – Chime Nguyen
  + Question 8 – Ivan Ko
  + Question 9 – Ivan Ko
  + Question 10 – Ivan Ko
* Conclusion – Chime Nguyen and Ivan Ko

## SQL Tables

* PUBLISHER Table – Chime Nguyen
* LIBRARY\_BRANCH Table – Chime Nguyen
* BORROWER Table – Ivan Ko
* BOOK Table – Ivan Ko
* BOOK\_LOANS Table – Chime Nguyen
* BOOK\_COPIES Table – Trung Nguyen
* BOOK\_AUTHORS Table – Trung Nguyen

## SQL Queries

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

# 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>