Library Database

Iteration 4

Dewey Decimators (Harminder Simplay, Joshua Sommer, Pavel Peev)

CSS 475

Spring 2023

UNIVERSITY *of* WASHINGTON

BOTHELL

Table of Contents

Link to Database repository: https://github.com/CSS-475-Databases-final-project/LibraryDatabase
Link to hosted web page for database: https://paveldatabaseapp.azurewebsites.net/

Updates

Updates since Iteration 0:

- Added title page
- Added Entity Relationship diagram
- Added Relation Model
- Added further explanation of the relations and attributes chosen
- Added reasoning, concerns, and limitations to the decisions made

Updates since Iteration 1:

- Added tooling and tooling assessment
- Various editing and format changes for cohesiveness

Updates since Iteration 2:

- Updated project schedule of iteration 3 to reflect the new due date (5/27)
- Set up and added a link to our GitHub repository for the Library database under the Table of Contents
- Hosted the MySQL database on an Azure web page
- Revised domains and constraints in the table creations to better fit the use of the database

Updates since Iteration 3:

- Updated example queries to better reflect the User Scenarios put in place for the UI
- Added Normalization section discussing the normal form of each table in the database
- Added SQL queries section to show the query code used in our database and UI
- Added a discussion section breaking down our experience with this project
- Updated tooling to match what exactly we used.

Introduction

This document serves as a written overview of the Library Database created by **Dewey Decimators**. The database contains data related to a hypothetical library system and their inventory of books. The purpose of this database is to help the users of this library to determine the location of their desired book, or to find a book they may be interested in.

Data to be Stored in Database

- Library
  - Address
  - Zip Code
  - City
  - State
- Customers
  - Name
  - Library card number
  - Date of Birth
- Books
  - Author ID
  - Title

    - Language
    - Publisher
    - Genre
    - ISBN
    - Year published
- Author
    - Author ID
    - Author name
    - Author nationality
    - Author date of birth
- Library owns books
    - Location on shelf
    - Copies owned
- Books checked out
    - Return date
    - Reference ID
    - Overdue status
    - ISBN

Queries

Below are some examples of queries library users would enter. Such queries would support the application because the users need a quick, effective way to search for the books they want, even if the queries involve further steps such as aggregation, multiple tables, or grouping.

Example queries

- On what shelf and in what library can I find *Harry Potter and the Philosopher's Stone*?
- When will *Sapiens by Yuval Noah Harari* be available at any library in Seattle?
- What are all the available books written by *Stephen King*? *(Involves multiple tables)*
- What books are available that are written in *Spanish*?
- What books are in the database that contain 'Star' in the title?
- Find all books written by Mark Twain after 1860 (grouping).
- What books do I (a customer) have checked out, and what is the return date?
- Does a specific customer have any books overdue currently?
- What books are overdue in the system currently?
- Add a book to the library.

Project Schedule (Timeline)

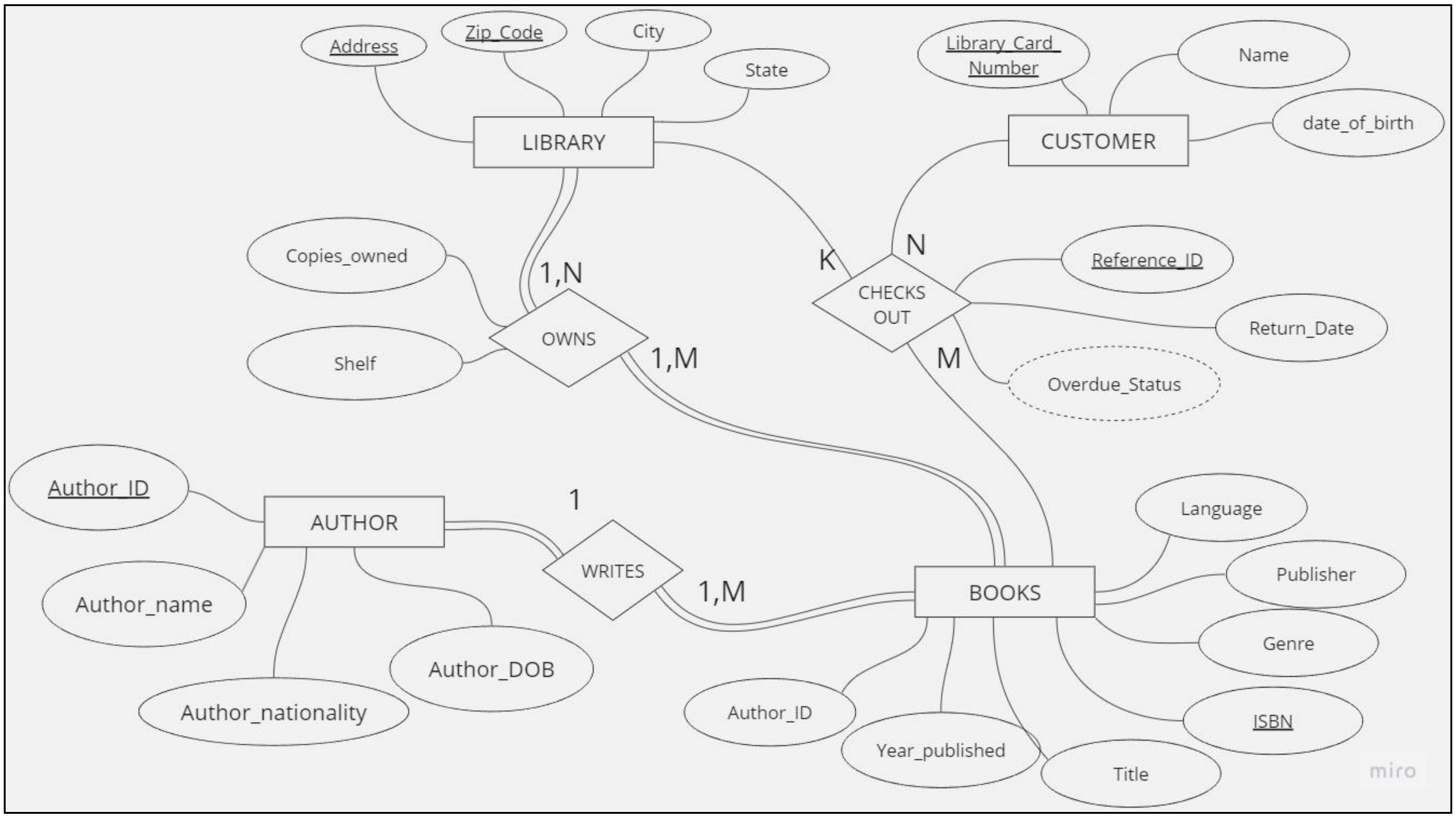
Project deliverables	**Description**	**Who will work on this part of the project**	**Due Date**
~~Refine project overview (Iteration 0)~~	~~Based on feedback from Iteration 0, provide an overview of our project, goals, purpose, what and why we are doing this.~~	~~ALL~~	~~April 21~~
~~Entity Relation Diagram~~	~~Create an entity relation diagram and give an introduction and description~~	~~ALL~~	~~April 22~~
~~Relational Data Model~~	~~Create a relational data model and give an introduction and description~~	~~ALL~~	~~April 22~~
~~Iteration 1~~	~~Combine all above into a cohesive document and submit for review~~	~~ALL~~	**~~April 23~~**
~~Tool Research & Assessment~~	~~Tools you plan to use. Research what tools to use for DBMS, UI, IDE, Hosting.~~	~~ALL~~	~~April 30~~
~~Slide and presentation creation~~	~~Use information gathered in Tool Research & Assessment to create a 3 minute presentation~~	~~ALL~~	~~May 7~~
~~Tooling and Big Picture Presentation~~	~~Give a brief, 3 minute presentation about project~~	~~ALL~~	**~~May 8~~**
~~Refine what project tools to use based on feedback~~	~~After our presentation and conversations with professor and graders, we will ensure that we have a clear intention on what tools to use for our project~~	~~ALL~~	~~May 11~~
~~Iteration 2~~	~~Combine all above into a cohesive document and submit for review~~	~~ALL~~	**~~May 12~~**
~~Create Database~~	~~Fully set up and host the database structure for our data, include SQL statements used to create database entities~~	~~ALL~~	~~May 19~~
~~Populate Database~~	~~Fill the database with sample data~~	~~ALL~~	~~May 21~~
~~Upload database~~	~~Upload our database file and list in~~	~~ALL~~	~~May 25~~

Project deliverables	**Description**	**Who will work on this part of the project**	**Due Date**
	~~the report the name of the tables used by our project, *brainstorm testing ideas for our database*~~		
~~**Iteration 3**~~	~~Combine all above into a cohesive document and submit for review~~	~~ALL~~	~~**May 27**~~
~~Update project based on feedback~~	~~Make any changes to previous iteration based on feedback, *start testing*~~	~~ALL~~	~~May 26~~
~~Add Discussion~~	~~Create a section in the project for discussion following guidelines on Iteration 4~~	~~ALL~~	~~May 27~~
~~Normalization~~	~~Assess all tables and discuss in the project what NF they are in and how we can normalize the data~~	~~ALL~~	~~May 28~~
~~Add SQL Query Statements~~	~~Create a section in the project for introducing and explaining all SQL statements used in the project~~	~~ALL~~	~~June 2~~
~~Poster finalization~~	~~Complete poster for Demo Presentation~~	~~ALL~~	~~June 3~~
~~Iteration 4 completion~~	~~Combine all above into a cohesive document~~	~~ALL~~	~~June 4~~
~~**Demo presentation**~~	~~Present our project to the class~~	~~ALL~~	~~**June 5**~~
~~**Iteration 4 submission**~~	~~Submit all above as a stand alone project~~	~~ALL~~	~~**June 7**~~
~~**Evaluations**~~	~~Submit *Team Presentation Evaluations* and *Project: Peer Evaluation*~~	~~ALL~~	~~**June 9**~~

Entity Relationship Diagram and Relational Model

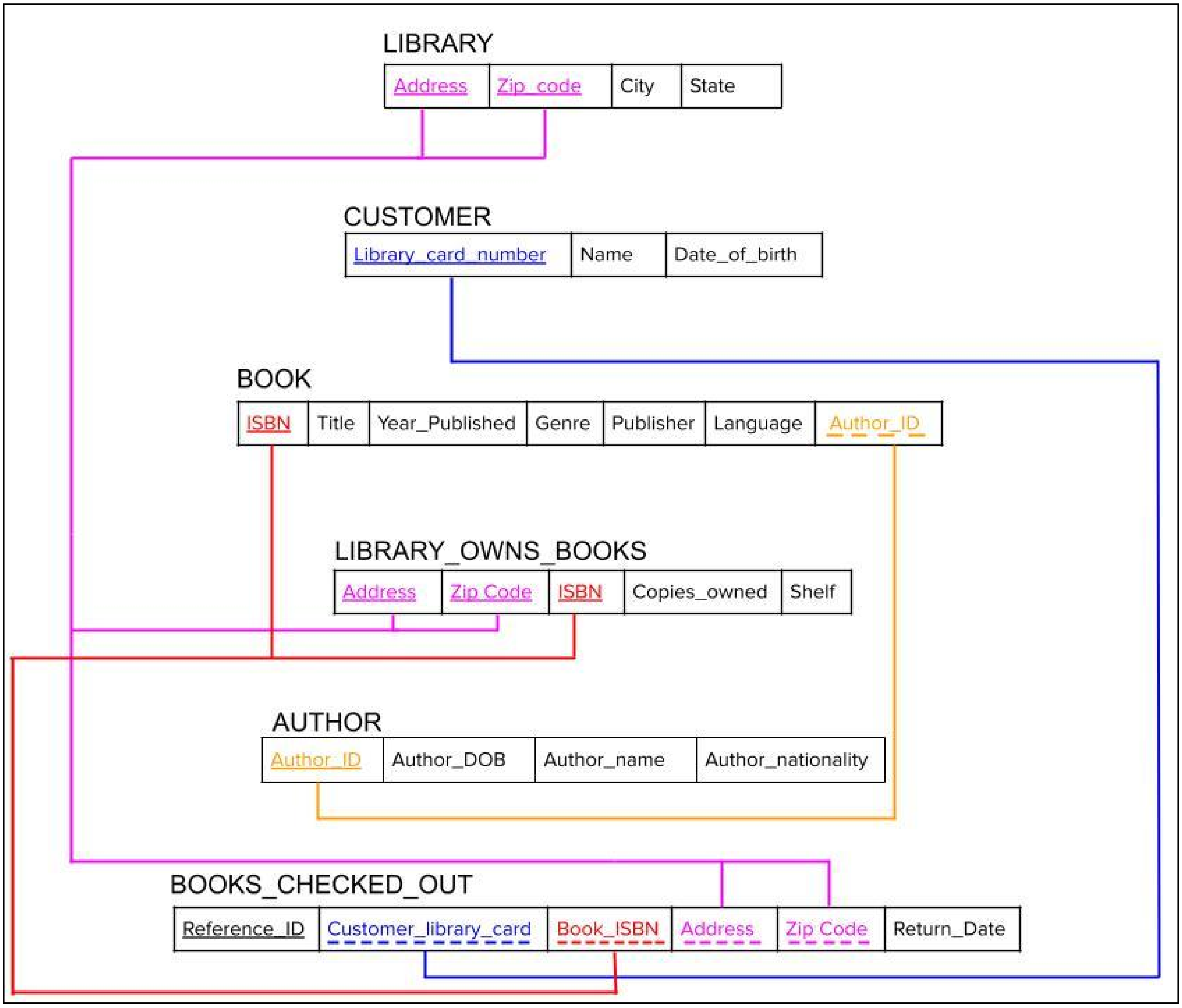
The Entity-Relationship Diagram and Relational Model for our project are created to show a visual overview of the project. This aids users and stakeholders in understanding the flow of data and organization of the project. This section also includes the assumptions made and domain of each attribute in the models.

Entity Relationship Diagram



Link for Better View: https://miro.com/app/board/uXjVMQcsAxM=/?share_link_id=341206603349

Relational Model



Description and Domain Constraints

LIBRARY

Represents the Library entity itself. The attributes of the library are its ***address***, ***zip_code***, ***city***, and ***state***. The library has multiple locations, hence why all attributes are related to location.
Domain constraints:

- Address - VARCHAR(250) NOT NULL
- Zip_code - INT(5) NOT NULL
- City - VARCHAR(50) NOT NULL
- State - CHAR(2) NOT NULL
- PRIMARY KEY (Address, Zip_code)

CUSTOMER

Represents the Customer entity. Precisely, it represents the customer of the library. The typical attributes of a customer are their ***date_of_birth***, ***name***, and ***library_card_number***. To become a customer, you need to visit the library and register for a library card to be able to check out books. Each customer has a unique library card number that can be used at any library in the system.
Domain constraints:

- Library_Card_Number - INT(8) NOT NULL
- Name - VARCHAR(50) NOT NULL
- Date_of_birth - DATE NOT NULL
- PRIMARY KEY (Library_card_number)
- CHECK(Date_of_birth < DATE(‘2023-05-31’))

BOOK

Represents a book. A book can be checked out by a customer or held in a library.
Domain constraints:

- ISBN - VARCHAR(13) NOT NULL
- Title - VARCHA(100) NOT NULL
- Year_published - INT(4) NOT NULL
- Genre - VARCHAR(10) NOT NULL
- Publisher - VARCHAR(50) NOT NULL
- Language - VARCHAR(50) NOT NULL
- Author_ID - VARCHAR(10) NOT NULL
- PRIMARY KEY (ISBN)
- FOREIGN KEY (Author_ID) REFERENCES AUTHOR(Author_ID)
- CHECK(Genre = 'Fiction' OR
  Genre = 'Nonfiction' OR
  Genre = 'Drama' OR
  Genre = 'Poetry' OR
  Genre = 'Folktale')

AUTHOR

Carries information about the author, authors write books.

Domain constraints:
- Author_ID - VARCHAR(10) NOT NULL
- Author_DOB - DATE NOT NULL
- Author_name - VARCHAR(50) NOT NULL
- Author_nationality - VARCHAR(50) NOT NULL
- PRIMARY KEY (Author_ID)
- CHECK(Author_DOB < DATE(‘2023-05-31’))

LIBRARY_OWNS_BOOKS (Relationship)
Represents the entity for the *location* of the books. Location refers to both the location within the library, and which library owns the book.
Domain constraints:
- Address - VARCHAR(250) NOT NULL
- Zip_code - INT(5) NOT NULL
- ISBN - VARCHAR(13) NOT NULL
- Copies_owned INT(2) NOT NULL
- Shelf - VARCHAR(4) NOT NULL
- PRIMARY KEY (Address, Zip_code, ISBN)
- FOREIGN KEY (Address, Zip_code) REFERENCES LIBRARY(Address, Zip_code)
- FOREIGN KEY (ISBN) REFERENCES BOOK(ISBN)
- CHECK(Copies_owned >= 0)

BOOKS_CHECKED_OUT (Relationship)
Represents when a customer checks out a book to take home. Shows a reference ID, who checked out the book, where it was checked out from, and when the book should be returned.
Domain constraints:
- Reference_ID - INT(12) NOT NULL
- Customer_library_card - INT(8) NOT NULL
- Book_ISBN - VARCHAR(13) NOT NULL
- Address - VARCHAR(250) NOT NULL
- Zip_code - INT(5) NOT NULL
- Return_date - DATE NOT NULL
- PRIMARY KEY (Reference_ID)
- FOREIGN KEY(Customer_library_card) REFERENCES CUSTOMER(Library_card_number)
- FOREIGN KEY(Book_ISBN) REFERENCES BOOK(ISBN)
- FOREIGN KEY (Address, Zip_code) REFERENCES LIBRARY(Address, Zip_code)

Reasoning

- A customer and library can check out more than one book at a time, and the book can be the same ISBN but multiple copies, so a reference ID (which refers to the copies checked out) for the check out is needed.

- An author can write multiple books, but each book only has one author, in the case of more than one author, the primary author will be listed.
- Many libraries can own the same book ISBN and Title, and many different books can be owned by the same library.
- Customer library card number must be less than 100,000,000 This number is chosen because our database is not meant to be nationwide, and the largest population state is California with 39 million. Each person can have a card with a unique number.

Concerns, Assumptions, and Limitations

Concerns

- Must use Author ID rather than name in book attributes, because authors may have the same name.
- As people are born and pass, and people move to and from the area, the library card number will need to be addressed eventually by removing users to add new ones.
- The borrowing history of a book is not tracked, which may be useful for analyzing trends in book popularity or identifying books that require replacement due to wear and tear.
- The database does not track the condition of books, which may impact their availability for checkout.

Assumptions

- A customer can exist in the system without checking out a book, but customers ***are*** required to have a valid library card in order to check out books in case they decide to do so in the future.
- Library books are assumed to be available for checkout unless they are marked as "reserved" or "checked out."
- Every Author in our database has at least one book, and every book has 1 author.
- Author ID is a unique key generated by our database for the author.
- Reference_ID is a unique key generated by our database for the copies checked out.
- The library only carries physical copies of books, rather than digital copies.
- The borrowing period for books is assumed to be uniform across all customers, even though it may vary depending on the type of book or the customer's borrowing history.

Limitations

- Can only have one author per book, even though multiple authors can collaborate to write a book. We are not using a multiple value attribute for the author.
- The library does not carry eBooks, research journals, magazines, or media such as films and music. This database is to start off with physical books only because books are the center focus of any library and they are an easy component of the database to start with.
- For Copies_owned, the number of copies of any book should not exceed 2 digits. This means that the number of copies should be between 0-99. Reason being is because if a value of 1000 was entered, that would exceed a realistic limit of books owned, no matter the library's size.
- Reference_ID must be 12 digits, no more or less than that. Also, it only accepts integers and cannot be null.

Tooling Assessment

Presentation Link

Project Overview

- *Overview:*
  Our team's database is for a library system with multiple libraries. It tracks each library's customers (meaning someone who registered for a library card with the library to be able to check out books) and the books they have on loan, as well as the books in stock and where they're found. Additionally, it keeps track of authors and what books they published.

- *Purpose:*
  The database allows the library's employees to help customers track books they're looking for, and see if they are in stock at this location, another branch, and when they should be returned if they are on loan.

- *Data:*
  Data is to be gathered from real books, with fabricated customers and branch locations.

Tooling

- DBMS
  - We will be using MySQL, which is the standard database for SQL
  - Most common application to practice SQL code. Therefore more resources for the group to learn it faster.
- Web Framework/UI
  - Flask for Python framework and HTML/CSS for frontend development
  - We as a team are more used to coding in Python, which is why we'll be using Flask.
  - HTML/CSS is the simplest UI framework for us to use and the easiest to learn. We don't need anything too fancy, so no point in learning a more complicated framework for our website.
- IDE
  - PyCharm and/or Visual Studio Code both work with Python
  - This is an either/or situation. We had either in mind and are figuring out which one to selectively focus on. However, for now, we know that it will be one of those two IDEs we'll be using.
- Hosting
  - Azure SQL hosting and/or Azure VM because both are easy to use and offer the user free credits.
  - Much like above, this is also an either/or situation. We're set on either of those two, but unsure of which one of the two we'll be using.
- Version Control
  - GitHub, standard version control to track our progress and save previous versions of our code in case we need to retrieve some of it.
- Generation of test data
  - Mockaroo is a website that generates tuples of a SQL database within given constraints, while these constraints aren't fully customizable, they are close enough in most cases to work for this application

- ChatGPT is an AI chat assistant, using ChatGPT allowed us to create any tuples that had foreign keys. ChatGPT can remember the values already in the table, and use those when writing a tuple with a foreign key, most of the time this is correct, but the insertions must still be inspected for errors.

Database Creation and Population

MySQL Database Access Parameters

hostname = pavelserver.mysql.database.azure.com. Username = thesquashedman. Port = 3306. Password = #cooldude2. Database = library
Web Application

https://paveldatabaseapp.azurewebsites.net/
GitHub Repository
https://github.com/CSS-475-Databases-final-project

Test Data

Data for this database was generated by https://www.mockaroo.com/, and was ensured to match referential and domain constraints by hand. All data is generated by AI and has no reference to real books or real libraries. This database is purely hypothetical.

Code snippets from insertions:
(See Database Creation for full implementation) - Dewey Decimators - Database Creation

INSERT INTO LIBRARY (Address, Zip_code, City, State) VALUES ('2 Hayes Plaza', '98206', 'Everett', 'WA');

INSERT INTO CUSTOMER (Library_card_number, Name, Date_of_birth) VALUES (87455864, 'Rikki', '1962-10-29');

INSERT INTO AUTHOR (Author_ID, Author_name, Author_nationality, Author_DOB) VALUES ('3259877789', 'Karlan Willoughway', 'Argentina', '1951-03-26');

INSERT INTO BOOK (ISBN, Title, Year_published, Genre, Publisher, Language, Author_ID) VALUES ('7537347041701', 'War on Democracy, The', 2000, 'Nonfiction', 'Champlin Inc', 'Yiddish', '3259877789');

INSERT INTO LIBRARY_OWNS_BOOKS (Address, Zip_code, ISBN, Copies_owned, Shelf) VALUES ('2 Hayes Plaza', '98206', '7537347041701', 5, 'A1');

INSERT INTO BOOKS_CHECKED_OUT (Reference_ID, Customer_library_card, Book_ISBN, Address, Zip_code, Return_date) VALUES (1, 87455864, '7537347041701', '2 Hayes Plaza', 98206, '2023-06-06');

Bad Test Data

CUSTOMER
INSERT INTO CUSTOMER (Library_card_number, Name, Date_of_birth) VALUES (12345678, 'John', '2024-01-01');
-- Date of birth in the future

Why the bad test data was chosen: The customer has a date of birth set in the future. This violates the Check constraint under the Date_of_birth column because the customer's DOB must always be in the past.

AUTHOR
INSERT INTO AUTHOR (Author_ID, Author_name, Author_nationality, Author_DOB) VALUES ('9876543210', 'Jane Doe', 'United States', '2024-05-31');

-- Date of birth in the future

Why the bad test data was chosen: The author has a birth date set in the future. Just like CUSTOMER, this also violates the Check constraint under the Date_of_birth column because the author's DOB must always be in the past.

BOOK
INSERT INTO BOOK (ISBN, Title, Year_published, Genre, Publisher, Language, Author_ID) VALUES ('12345678901234', 'Invalid Book', 12345, 'Unknown', 'Publisher', 'English', '1234567890');

-- Invalid ISBN, Year_published, Genre, and Author_ID

Why the bad test data was chosen: The book has invalid values for ISBN, Year_published, Genre, and Author_ID. First off, the ISBN must have a length of 13 digits. In this case, it has 14, violating the domain constraint. Secondly, the year is set to 12345. The year must be a four digit integer. Thirdly, the Genre is set to 'Unknown', which violates the Check constraint because there is no 'Unknown' genre option. Lastly, the Author_ID is set to 12345678901234, which doesn't correspond to any existing authors in the AUTHOR table, violating the referential constraint.

LIBRARY_OWNS_BOOKS
INSERT INTO LIBRARY_OWNS_BOOKS (Address, Zip_code, ISBN, Copies_owned, Shelf) VALUES ('Invalid Address', '12345', '1234567890123', 1000, 'D9');

-- Invalid Address, Zip_code, ISBN, Copies_owned

Why the bad test data was chosen: The bad data in question is 'Invalid Address' with a zip code of '12345'. Precisely, the library address and zip code violate the referential integrity constraint because they both do not correspond to a valid library in the LIBRARY table. ISBN is also invalid and violates the referential constraint because it doesn't correspond to an existing value in the BOOK table. Lastly, the book in question contains 1000 copies, which exceeds a realistic limit. The number of copies shouldn't exceed 2 digits.

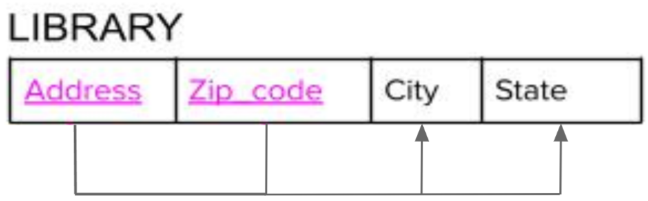
BOOKS_CHECKED_OUT

INSERT INTO BOOKS_CHECKED_OUT (Reference_ID, Customer_library_card, Book_ISBN, Address, Zip_code, Return_date) VALUES (123456789012123, 12345678, '1234567890123', 'Invalid Address', '12345', '2021-01-01');

-- Invalid Reference_ID, Customer_library_card, ISBN, Address, Zip_code, and Return_date

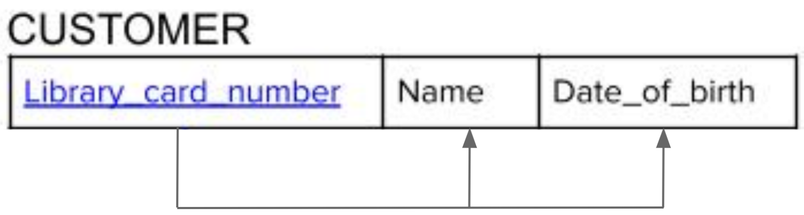
***Why the bad test data was chosen*:** The Reference_ID doesn't follow a realistic length. As specified in Description and Domain Constraints, it should be a max of 12 integers long. Customer_library_card violates the referential constraint because it doesn't correspond to an existing customer in the CUSTOMER table. Likewise with ISBN with the BOOK table. Address and Zip_code also violate the referential constraint because they don't correspond to a valid address in LIBRARY. Lastly, the Return_date is set to '2021-01-01', which is in the past.

Normalization

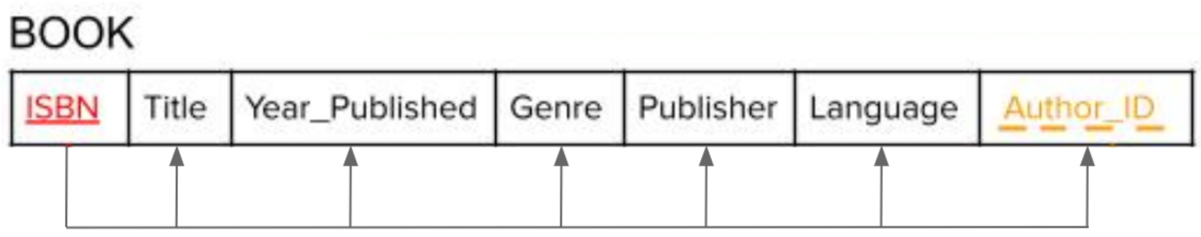


Given the fact that zip codes can cross both city and state lines, zip codes alone cannot determine city or state.

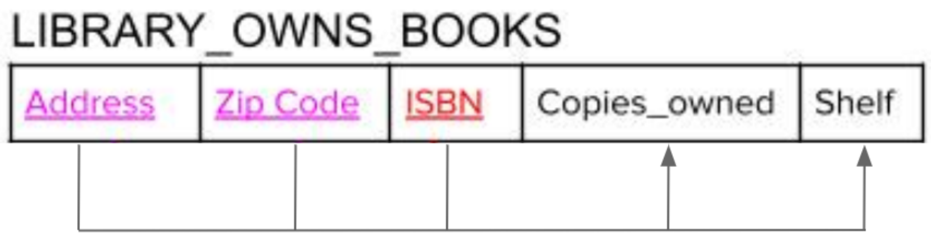
The CUSTOMER table is in BCNF as it does not have any multi-value attributes, the primary key cannot be broken up, there are no transitive dependencies, and there are no primary key attributes with functional dependency on any part outside of the primary key.



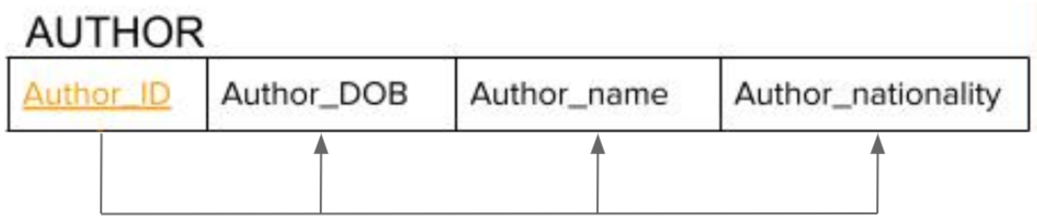
The CUSTOMER table is in BCNF as it does not have any multi-value attributes, the primary key cannot be broken up, there are no transitive dependencies, and there are no primary key attributes with functional dependency on any part outside of the primary key.



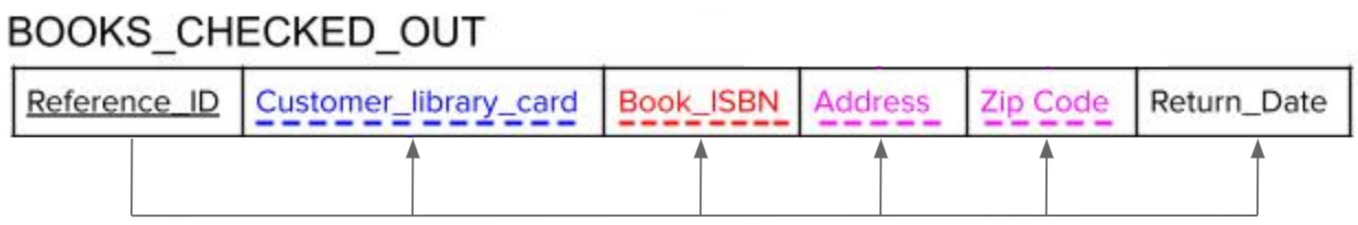
The BOOK table is in BCNF as it does not have any multi-value attributes, the primary key cannot be broken up, there are no transitive dependencies, and there are no primary key attributes with functional dependency on any part outside of the primary key.



The LIBRARY_OWNS_BOOKS table is in BCNF as it does not have any multi-value attributes, the attributes all rely on the entire primary key, there are no transitive dependencies, and there are no primary key attributes with functional dependency on any part outside of the primary key.



The AUTHOR table is in BCNF as it does not have any multi-value attributes, the primary key cannot be broken up, there are no transitive dependencies, and there are no primary key attributes with functional dependency on any part outside of the primary key.



The BOOKS_CHECKED_OUT table is in BCNF as it does not have any multi-value attributes, the primary key cannot be broken up, there are no transitive dependencies, and there are no primary key attributes with functional dependency on any part outside of the primary key.

SQL Query Statements

SQL Statement	Purpose

CREATE TABLE LIBRARY (
    Address VARCHAR(250) NOT NULL,
    Zip_code INT(5) NOT NULL,
    City VARCHAR(50) NOT NULL,
    State CHAR(2) NOT NULL,
    PRIMARY KEY (Address, Zip_code)
);
	Create the table holding the libraries

CREATE TABLE CUSTOMER (
    Library_card_number INT(8) NOT NULL,
    Name VARCHAR(50) NOT NULL,
    Date_of_birth DATE NOT NULL,
    PRIMARY KEY (Library_card_number),
    CHECK (Date_of_birth < DATE('2023-05-31'))
);
	Create the table holding the customers

CREATE TABLE AUTHOR (
    Author_ID VARCHAR(10) NOT NULL,
    Author_name VARCHAR(50) NOT NULL,
    Author_nationality VARCHAR(50) NOT NULL,
    Author_DOB DATE NOT NULL,
    PRIMARY KEY (Author_ID),
    CHECK(Author_DOB < DATE('2023-05-31'))
);
	Create the table holding the authors

CREATE TABLE BOOK (ISBN VARCHAR(13) NOT NULL, Title VARCHAR(100) NOT NULL, Year_published INT(4) NOT NULL, Genre VARCHAR(10) NOT NULL, Publisher VARCHAR(50) NOT NULL, Language VARCHAR(50) NOT NULL, Author_ID VARCHAR(10) NOT NULL, PRIMARY KEY (ISBN), FOREIGN KEY (Author_ID) REFERENCES AUTHOR(Author_ID), CHECK(Genre = 'Fiction' OR Genre = 'Nonfiction' OR Genre = 'Drama' OR Genre = 'Poetry' OR Genre = 'Folktale'));	Create the table holding the books
CREATE TABLE LIBRARY_OWNS_BOOKS (Address VARCHAR(250) NOT NULL, Zip_code INT(5) NOT NULL, ISBN VARCHAR(13) NOT NULL, Copies_owned INT(2) NOT NULL, Shelf VARCHAR(4) NOT NULL, PRIMARY KEY (Address, Zip_code, ISBN), FOREIGN KEY (Address, Zip_code) REFERENCES LIBRARY(Address, Zip_code), FOREIGN KEY (ISBN) REFERENCES BOOK(ISBN), CHECK(Copies_owned >= 0));	Create the table that holds the data of what libraries hold what books and where they are within the library
CREATE TABLE BOOKS_CHECKED_OUT (Reference_ID INT(12) NOT NULL, Customer_library_card INT(8) NOT NULL, Book_ISBN VARCHAR(13) NOT NULL, Address VARCHAR(250) NOT NULL, Zip_code INT(5) NOT NULL, Return_date DATE NOT NULL, PRIMARY KEY (Reference_ID), FOREIGN KEY(Customer_library_card) REFERENCES CUSTOMER(Library_card_number), FOREIGN KEY(Book_ISBN) REFERENCES BOOK(ISBN), FOREIGN KEY (Address, Zip_code) REFERENCES LIBRARY(Address, Zip_code));	Create the table that holds the data of what books are checked out by who from what library,

SELECT CUSTOMER.Library_card_number FROM CUSTOMER WHERE CUSTOMER.Library_card_number = Search	Check if a customer exists in the database, Search is from the UI
SELECT BOOK.title, CHECKED.Name, CHECKED.Return_date FROM BOOK, (SELECT * FROM CUSTOMER, BOOKS_CHECKED_OUT AS BCO WHERE CUSTOMER.Library_card_number BCO.Customer_library_card) AS CHECKED WHERE BOOK.ISBN = CHECKED.Book_ISBN AND CHECKED.Return_date < DATE('now') AND Library_card_number = Search	A query that displays all customers that have overdue books so the library can notify them, with an optional last line of searching for a specific customer with Search being an input from the UI
SELECT BOOK.ISBN FROM BOOK WHERE BOOK.ISBN = ISBN	Check if a book exists in the database, ISBN is from the UI
SELECT AUTHOR.Author_ID FROM AUTHOR WHERE AUTHOR.Author_ID= Author_ID	Check if an author exists in the database, Author_ID is from the UI
INSERT INTO BOOK (ISBN, Title, Year_published, Genre, Publisher, Language, Author_ID) VALUES (ISBN, Title, Year_published, Genre, Publisher, Language, Author_ID);	Insertion into book database used on the librarian page, ISBN, Title, Year_published, Genre, Publisher, Language, Author_ID are all from the UI user input
SELECT MAX(Reference_ID) FROM BOOKS_CHECKED_OUT	Gets the current highest reference ID to assign the next integer to the next book checked out

SELECT L.copies_owned - COUNT(C.book_isbn) as copies_available
FROM (
    SELECT B.*, address, zip_code, copies_owned, shelf
    FROM (
        SELECT *
        FROM BOOK
        WHERE ISBN = isbn) as B
        left join LIBRARY_OWNS_BOOKS as L on B.isbn = L.isbn
    WHERE L.address = address
    AND L.zip_code = zip) as L
LEFT JOIN
BOOKS_CHECKED_OUT as C
ON L.isbn = C.book_isbn
AND L.address = C.address
AND L.zip_code = C.zip_code
GROUP BY L.isbn, L.address, L.zip_code
	Check that library has book and that it isn't checked out isbn, address, and zip are all user inputs from the UI

INSERT INTO BOOKS_CHECKED_OUT (Reference_ID, Customer_library_card, Book_ISBN, Address, Zip_code, Return_date) VALUES (checkoutid, customer, isbn, address, zip, DATE_ADD(now(), INTERVAL 7 DAY));")	An insert query to insert a new check out instance from the checkout page on the website. checkoutid, customer, isbn, address, zip are all user inputs from the UI, and the return date is calculated as today's date plus 7 days

```
SELECT L.ISBN,
       L.Title,
       L.Year_published,
       L.Genre, L.Publisher,
       L.Language, A.Author_name,
       total_copies,
       copies_checked_out,
       total_copies - copies_checked_out as available_copies
FROM (
       SELECT B.*, SUM(copies_owned) as total_copies
       FROM (
            SELECT *
            FROM BOOK
            WHERE title like '%name%'
            )
       as B
       LEFT JOIN
       LIBRARY_OWNS_BOOKS as L
       ON B.isbn = L.isbn
       GROUP BY B.isbn
       )
       as L,
       (
       SELECT B.*, COUNT(C.book_isbn) as copies_checked_out
       FROM (
              SELECT *
             FROM BOOK
             WHERE title like '%name%'
             )
        as B
        LEFT JOIN BOOKS_CHECKED_OUT as C
            ON B.isbn = C.book_isbn
            GROUP BY isbn
            )
            as C,
            Author as A
            WHERE C.isbn = L.isbn AND A.author_ID = L.author_ID;
```

Get all books whose titles contain the name, as well as finding how many copies of the book exist in all libraries and how many copies of the book are checked out from all libraries. By changing "Where title like '%name'" to some other book attributes, we can search for books with those attributes.

```
SELECT
   L.ISBN,
    L.Title,
    L.address,
    Z.city,
    Z.state,
    L.zip_code,
    L.shelf,
    L.copies_owned,
    COUNT(C.book_isbn) as copies_checked_out,
    L.copies_owned - COUNT(C.book_isbn) as copies_available,
    MIN(C.return_date) as next_return_date
FROM
    (SELECT
      B.*,
      Address,
      Zip_code,
      copies_owned,
      shelf
   FROM
      (SELECT
        *
      FROM BOOK
      WHERE title like '%"+ name +"%')
    as B
    LEFT JOIN
    LIBRARY_OWNS_BOOKS as L
    ON B.isbn = L.isbn)
as L
LEFT JOIN
BOOKS_CHECKED_OUT as C
ON L.isbn =
   C.book_isbn
   AND L.address = C.address
   AND L.zip_code = C.zip_code,
LIBRARY as Z
WHERE Z.address = L.address AND Z.zip_code = L.zip_code
GROUP BY L.isbn, L.address, L.zip_code;
```

Get all books whose titles contain name, as well as what locations they are at and how many copies are owned and checked out from each location, as well as the next return date for a copy. By changing "Where title like '%name'" to some other book attributes, we can search for books with those attributes.

Discussion of Design and Results

User Scenarios

To develop a better understanding of our database's scope, we crafted several user scenarios - mainly for the customer and the librarian.

- **CUSTOMER**
  - As a customer, I want to search for books using any information I know about the book (Title, ISBN, Author, etc.).
  - As a customer, I want to find where a book is located, either location in the library, or library address that holds the book.
  - As a customer, I want to be able to reserve a book online to pick it up at the library.
  - As a customer, I want to see if a book is available, and if it is not, when it will be available.
  - As a customer, I want to be able to see what I currently have checked out and when it's due.
- **LIBRARIAN**
  - As a librarian, I want to be able to see if a customer has any overdue books.
  - As a librarian, I want to be able to see how many books are overdue in the system.
  - As a librarian, I want to be able to add a book to the library.

Project Evaluation

- **Effort spent overall**
  - We spent a lot of time each week on the project. We initially started off with four members. But after Ilya's lack of participation, we had to figure out how to divide the work evenly amongst three people. This was a difficulty at first because in a major group project like this, one more person makes a difference. It means one more pair of eyes, more skillsets to bring to the table, different approaches one could take with the project, and much more. All three of us had drastically different schedules from each other which was another challenge of ours. We spent around 20 hours a week working on project tasks and had to deliberately set aside time together each week to balance a four-person workload amongst three people.
- **Successes**
  - We were able to accommodate each other's skillsets. Whatever one person couldn't do, another person could do it instead. And vice versa. Despite our initial conflicts with balancing our workload after Ilya's departure, we handled it well and took extra measures in ensuring that our work gets done on time and with deliberate effort. Despite our varying schedules, we did a good job at staying on top of our deliverables and spending quality time on the content of each deliverable. That way, we don't have to go back and constantly redo and resubmit

    them when there's other deliverables coming up. Whenever something went wrong, we were proactive in asking the professor and the TAs for feedback on our work so that we could improve our following deliverables. After our feedback for Iteration 1, we caught early on what our expectations were for the project overall.

- **Challenges**
    - Iteration 1
        - When we received feedback for Iteration 1, we realized that we were missing a lot of core elements that are put into a project iteration. This includes elements like a title page, organization of the iteration (i.e.: putting the ER model before the RM), and including our reasoning behind why we decided to structure our database as is. After receiving feedback from the TAs, we went back to fix our Iteration 1 document before beginning Iteration 2. This experience taught us that in order for our project to stay on track, we must begin with a solid foundation or else those shortcomings will bleed into further parts of the project. We also learned about the basics of project management through this experience. There are certain expectations that are placed on professional documents. In order for those expectations to be fulfilled, we first have to understand our audience so we can properly articulate our points with them.
    - **Learning curve with new tools**
        - Flask
            - Prior to this course, none of us had much experience with Flask or Python. Hence, there was a learning curve at first. Before we could even learn how to make our data work with Flask, we had to figure out how Flask worked first. We did have experience with other frameworks though. And we have knowledge on Java, C++, and HTML Therefore, we picked up on it fast.
        - Azure Web App
            - Prior to this course, we had limited experience with Azure. We had some knowledge about web applications. But there was still a lot we had to learn on the spot. In order to build and deploy our library database, we first needed to inform each other about how Azure Web App works and how our database's functionalities would be applied to it.
        - Azure Database for MySQL hosting
            - Much like Azure Web App, we also didn't understand MySQL hosting at first. We had to learn about how to host our database to

            this cloud database service to make it compatible with Azure Web App.
    - Determining the scope of our database
        - Instead of including other entities that real-world libraries include such as journals, newspapers, CDs, etc., we decided to stick to books for simplicity. We initially wanted to include other entities to make it closely resemble a real-world library. But given the time constraints for this quarter alongside our other commitments outside of class, we soon realized that adding more entities would not be feasible.

- **Future Updates**
    - Security measures for different users
        - Enforce access restrictions depending on the DB user's role (customer, librarian, etc.) because not every user should be able to access the same information in the library database. Not only is it impractical, it also increases the chance of critical issues like privacy leaks, inaccurate information displayed, susceptibility to hacking, and much more.
        - To do this, we would introduce library account types (librarian view versus customer view), usernames, passwords, etc.
    - Add more entities (journals, newspapers, CDs, etc.) to make it closely resemble a real-life library.
        - As stated above, we initially wanted to add more entities to our database to make it look more realistic. But due to time constraints and other commitments, we scrapped that idea and just opted for books as an entity for simplicity. If this were a real-world, ongoing project, adding more entities is definitely something we would invest time in.
    - Improvements we would focus on if we had another week
        - UX improvements
            - If we were given one more week, we would refine our front-end functions. By this, we mean that we would make it more visually appealing and easy for the users to navigate. Right now, it contains all the required functionalities. But there are still many ways to improve the UI.
        - Add Database Functionality
            - If given another week, we would also add more data to our database. By this, we mean that we would add more library branch locations, books, and authors to it. Currently, there are a limited

number of books and locations. We would continue using Mockaroo and ChatGPT to support this objective.

Feedback Implementation (Project Iterations, Design, Approach, etc.)

- We received positive feedback from Iterations 2 and afterwards. But as stated above, we received critique only for Iteration 1. As assessed by TA Saurav Jayakumar, here were our critiques:
    - **Project Introduction, Description, or Explanation for Diagrams**: Include a front page with your project title and team member names.
    - **Entity Relation diagram**: For relations with cardinality as n/m, restrict cases to "1..n" t if 0 is not logical. E.g., A customer cannot check out 0 books.
    - **Relational Model (specifically the diagram):**
        - RM diagram should follow your ER diagram as the former is derived from the latter. Also, it should be connected with arrows and should not have text in between.
        - Foreign keys should be underlined using dashed lines.
    - **Relational Data Model (relations, attributes, keys, & any other constraints, design and description)**: Domain constraints have not been mentioned.
    - **Discussion of design decisions, concerns, assumptions, and limitations**: Discussion of design decisions, concerns, assumptions, and limitations are missing.
- We also set aside time to speak with Saurav to discuss the aforementioned critiques in depth. Through this discussion, we learned about how professional documents should and shouldn’t look. After meeting with Saurav, we made sure to implement his feedback, fix Iteration 1 first, and then work on Iteration 2.
- Afterwards, our efforts paid off because we received positive feedback for the following iterations. It pushed us to think deeper about the decisions, concerns, assumptions, and limitations of our database (as found on page 10) because we realized several potential issues we could encounter as we perform more research on the project. It taught us to think of the target audience in mind - the manager(s). In this case, it is the TAs and professor. They must have a clear idea of what assumptions, limitations, and concerns revolve around the database to minimize ambiguity.

How We Generated Data for Our Database

- To generate data for our database, we used two AI tools:
    - Mockaroo
        - Mockaroo is an online service that allows users to generate realistic test data for software development and testing purposes. It provides a platform for creating customized mock data sets.

        - For this project, we used Mockaroo to help us generate test data for book titles, authors, ISBN, year published, book publisher, and language of the book. The data itself is real books, but generated by AI.
    - ChatGPT
        - ChatGPT is a conversational/generative AI model developed by OpenAI that is trained to answer questions and converse with users by using its repository of text data.
        - Similar to Mockaroo, we also used ChatGPT to help us generate test data, However, the advantage ChatGPT gave us over Mockaroo was our understanding of the tables. ChatGPT was able to generate (mostly) correct INSERTIONs while complying with the many Foreign Keys on our tables.

How We Tested Our Data

- As stated above, we generated special data by using Mockaroo and ChatGPT. Here is a rundown of our test cases. An in-depth analysis on our test cases can also found on pages 12 and 13:
    - (CUSTOMER and AUTHOR) Date of birth in the future
        - Customers and authors cannot be born in the future (2024 and beyond).
    - (BOOK) Invalid ISBN, Year_published, Genre, and Author_ID
        - ISBN must have a length of 13 digits.
        - A book cannot be published in the future (2024 and beyond).
        - Author_ID in the BOOK table must correspond to an existing author in the AUTHOR table to avoid referential constraint violations.
        - Genre cannot be null or unknown.
    - (LIBRARY_OWNS_BOOKS) Invalid Address, Zip_code, ISBN, Copies_owned
        - Both address and zip_code must correspond to a valid, existing library in the LIBRARY table to avoid referential constraint violations.
        - In this context, the ISBN must also point to an existing book in the BOOK table to avoid referential constraint violations.
        - The number of copies must be between 0-99 because realistically, no library would realistically have 1000 copies of the same book.
    - (BOOKS_CHECKED_OUT) Invalid Reference_ID, Customer_library_card, ISBN, Address, Zip_code, and Return_date
        - Reference_ID must be 12 integers long, no more or less than that. It also takes in integers only and cannot be null either.
        - Customer_library_card_number must correspond to an existing customer in the CUSTOMER table to avoid referential constraint violations.

        - ISBN must correspond to an existing book in the CUSTOMER table to avoid referential constraint violations.
        - Address and zip_code must correspond to an existing library's address in the LIBRARY table to avoid referential constraint violations.
        - Return_date cannot be in the past (at the point in time a customer checks out a book).

How We Ensured Correctness

- While we greatly benefitted from the assistance of Mockaroo and ChatGPT, we still had to go back and look at pieces of the data that fit our vision of the database and reflect it accurately, as it did generate some erroneous data.
    - Foreign keys were closely paid attention to because those take more thought into the whole implementation. Establishing the right foreign keys is the foundation to a well-functioning database. Wherever we detected anomalies in the foreign keys, we fixed those manually before building up the database.

Database Hosting

- We hosted our database with the help of the following platforms:
    - Azure Database for MySQL Hosting (cloud)
        - Azure Database for MySQL Hosting is a fully managed database service provided by Microsoft Azure for hosting MySQL-based relational databases in the cloud. It simplifies the deployment, management, and scaling of MySQL databases, allowing developers to focus on their applications rather than the underlying infrastructure.
        - We used Azure's MySQL platform to host our database's code and link it to Flask (which ultimately links to our database's website). We did this by first establishing a connection to the Azure Database for MySQL through the use of imports.
    - Azure Web App
        - Azure Web App is a platform-as-a-service (PaaS) offering provided by Microsoft Azure. It allows developers to deploy and host web applications without the need to manage the underlying infrastructure.
        - In order to deploy and host our database's web application on Flask, we first needed to create the web application, check that it's structured correctly, and run the command prompt to import its functionalities into Flask.
    - Flask
        - Flask is a lightweight web framework for Python that allows developers to build web applications quickly and easily. It is classified as a

micro-framework because it provides only the essential features needed to create web applications, making it simple and flexible.

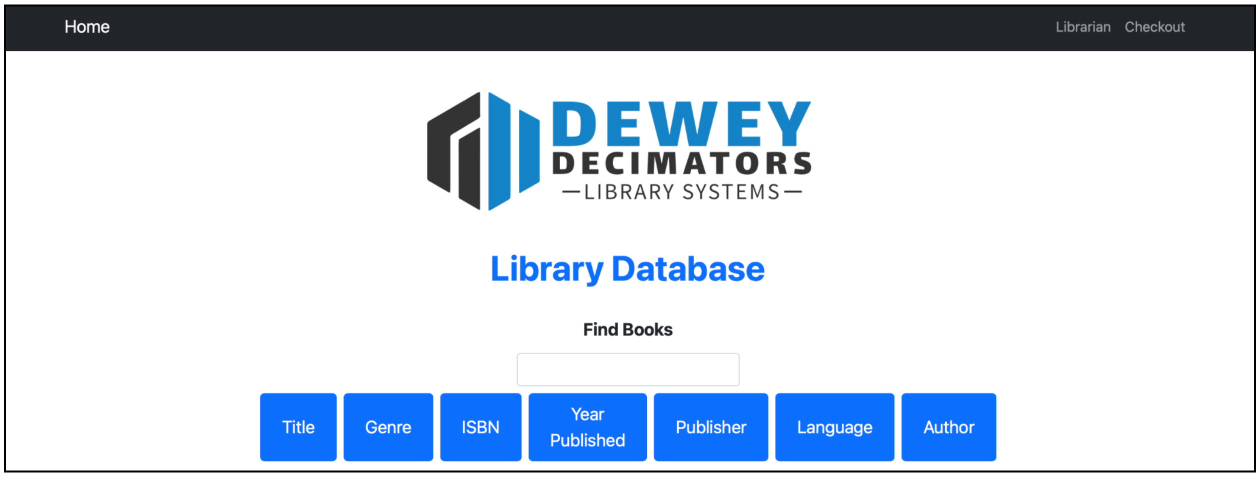
    - Flask is the framework that helped us build the web application, access the database items, and link all the information we had from Azure Web App and Azure Database for MySQL into one place.

- GitHub
    - GitHub is a web-based platform that provides hosting for version control repositories. It allows developers to collaborate, manage, and track changes to their code.
    - We used GitHub as a way to host and manage all of our data from each framework into one repository. It was an effective way for us to track each version of our work in the database in case we identified a problem and needed to go back to find patterns. It was also a good way for us to organize our data through the use of version control.

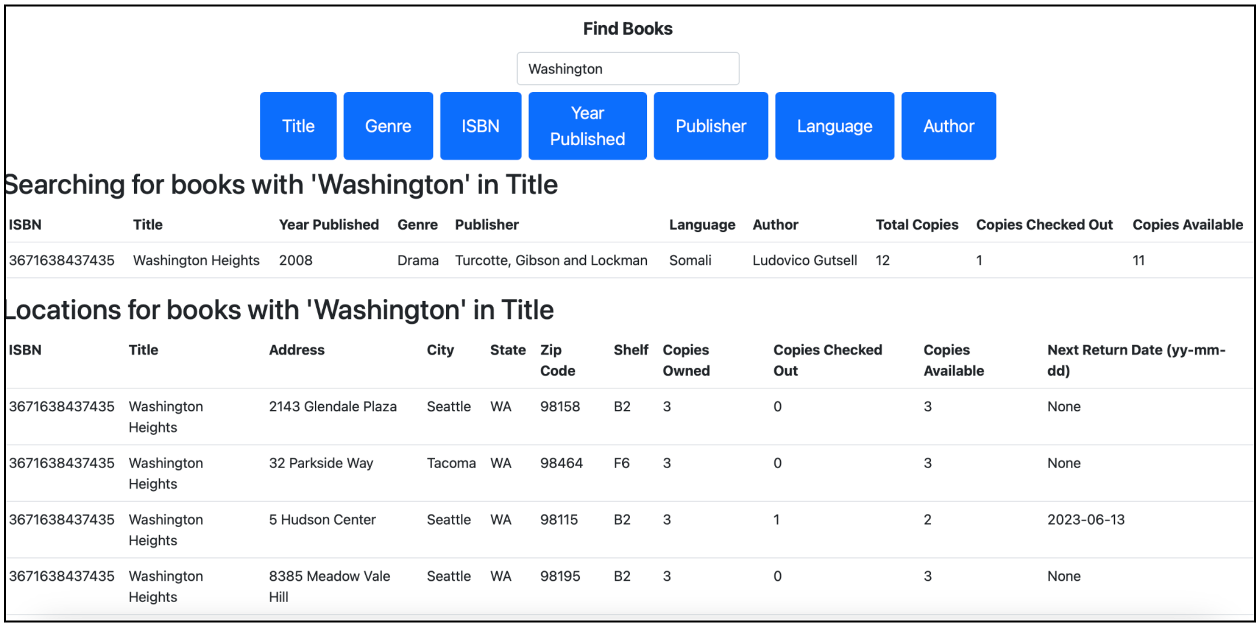
UI Screenshots

Screenshot 1 and 2 shows the homepage, this page fulfills the first user scenario for searching our database of books. The queries this page uses include Searching by any filter, and finding a location of books, on what shelf in what library.

Screenshot 1: Homepage



Screenshot 2: Book Search



Find Books

Washington

Searching for books with 'Washington' in Title

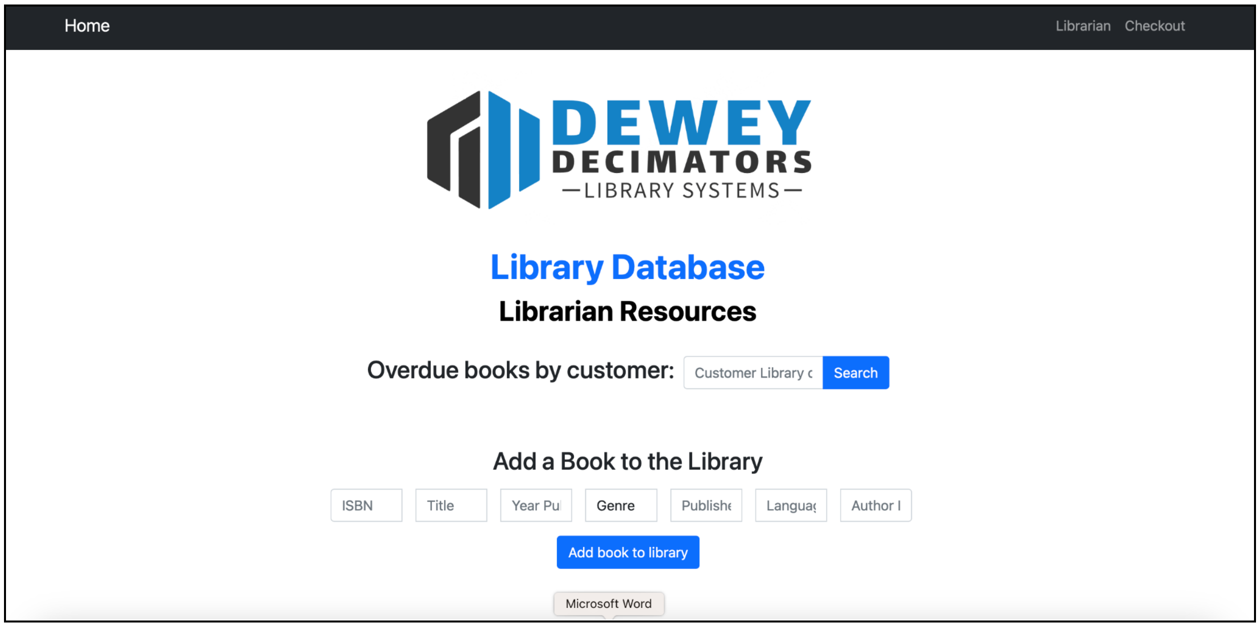
ISBN	Title	Year Published	Genre	Publisher	Language	Author	Total Copies	Copies Checked Out	Copies Available
3671638437435	Washington Heights	2008	Drama	Turcotte, Gibson and Lockman	Somali	Ludovico Gutsell	12	1	11

Locations for books with 'Washington' in Title

ISBN	Title	Address	City	State	Zip Code	Shelf	Copies Owned	Copies Checked Out	Copies Available	Next Return Date (yy-mm-dd)
3671638437435	Washington Heights	2143 Glendale Plaza	Seattle	WA	98158	B2	3	0	3	None
3671638437435	Washington Heights	32 Parkside Way	Tacoma	WA	98464	F6	3	0	3	None
3671638437435	Washington Heights	5 Hudson Center	Seattle	WA	98115	B2	3	1	2	2023-06-13
3671638437435	Washington Heights	8385 Meadow Vale Hill	Seattle	WA	98195	B2	3	0	3	None

Screenshots 3 and 4 display the Librarian page where librarians would go to access the database from an administrative side. These pages satisfy the user scenarios of finding all or specific customers with overdue books, and adding a book to the library. The two queries regarding overdue books are used in the first, and an INSERT statement is used to add books.

Screenshot 3: Library Page



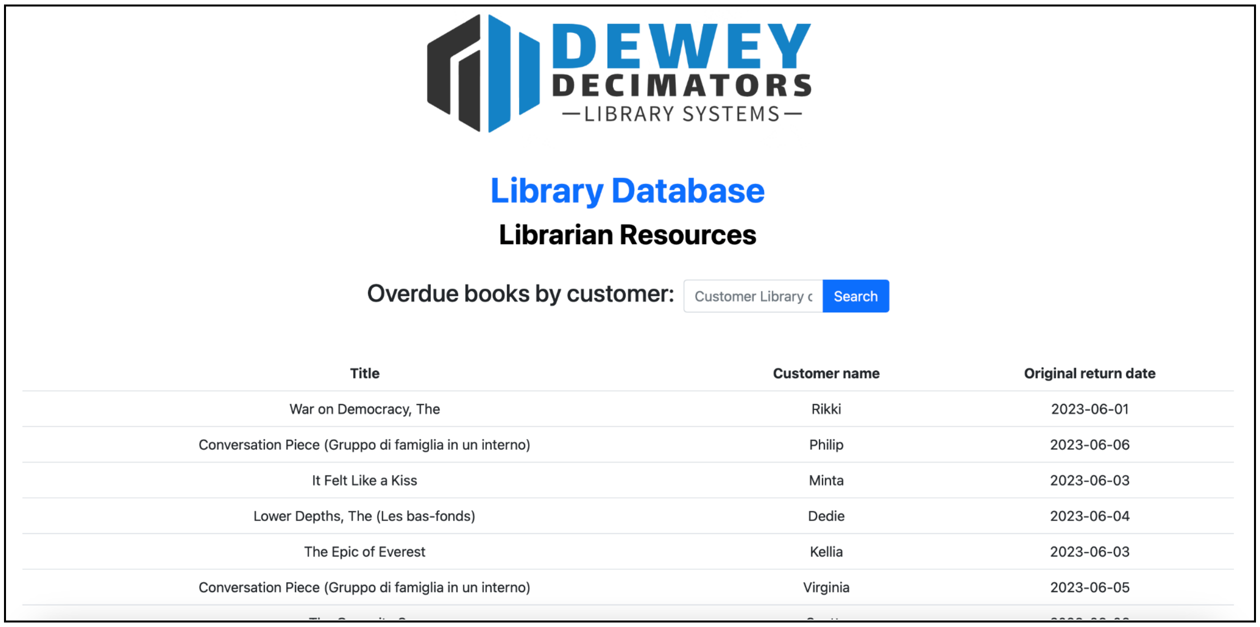
Library Database

Librarian Resources

Overdue books by customer: Customer Library c Search

Add a Book to the Library

Screenshot 4: Overdue Books


DEWEY DECIMATORS —LIBRARY SYSTEMS—

Library Database

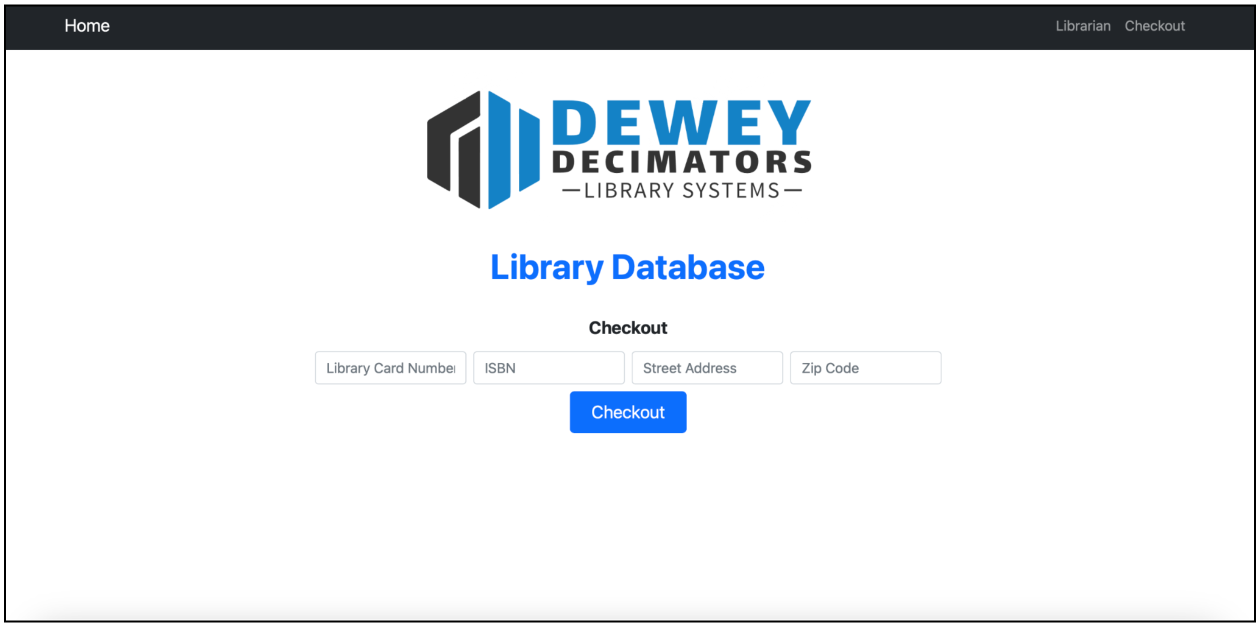
Librarian Resources

Overdue books by customer: Customer Library c Search

Title	Customer name	Original return date
War on Democracy, The	Rikki	2023-06-01
Conversation Piece (Gruppo di famiglia in un interno)	Philip	2023-06-06
It Felt Like a Kiss	Minta	2023-06-03
Lower Depths, The (Les bas-fonds)	Dedie	2023-06-04
The Epic of Everest	Kellia	2023-06-03
Conversation Piece (Gruppo di famiglia in un interno)	Virginia	2023-06-05

Screenshot 5 displays the checkout page. This page satisfies the user scenario of wanting to reserve a book to pick up at a certain library. This query checks if a book is available and then INSERTs a tuple to the relation.

Screenshot 5: Checkout Page


Home | Librarian | Checkout

DEWEY DECIMATORS —LIBRARY SYSTEMS—

Library Database

Checkout

Library Card Numbe | ISBN | Street Address | Zip Code

Checkout