BiblioTech

University Library Database System

Project Description

Introduction:
After working on the nine stages of our project's development, we have completed our university library database management system called Bibliotech. From the planning stage, to the development stage, the normalization stage, the debugging stage and finally, to the conclusion of our project.

Learning Outcome:
By working on this DBMS project, we learned how to:

1. Design a relational database management system (RDBMS)
2. Make an ER diagram of our DBMS
3. Define data types, structures, and constraints of the data in our database
4. Create and manage our database using DDL and DML statements (inserting, modifying, deleting data)
5. Create views for our database
6. Execute our queries at UNIX shell menu commands
7. Show functional dependencies
8. Normalize the database in 1NF, 2NF, 3NF and BCNF
9. Ensure data efficiency by making sure there is no redundant data other than absolutely necessary
10. Perform data manipulation by querying the database using SQL to retrieve specific data
11. Create a web based GUI for our database using HTML, CSS, Javascript, Python and Flask.

12. Create Relational Algebra (RA) notation for all of our queries

Entities

Each of our entities has various fields/attributes with defined data types and constraints. We have outlined **6** entities as was the requirement of this project.
The tables below hold information about the entities in our Library Management DBMS:

Author:

Author entity has 2 fields : Author_Name, Author_ID.
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Author_Name: String data type with the constraint NOT NULL (which indicates this field cannot be empty)
- Author_ID: Integer data type (PRIMARY KEY)

Author_Name	Author_ID

Book_Fine:

Book fine entity has 6 fields : Fine_ID, Student_ID, Status, Amount, Reason, Fine_Date
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Fine_ID: Integer data type (PRIMARY KEY)
- Student_ID: Integer data type with the constraint NOT NULL
- Status: String data type with default value ‘unpaid’
- Amount: Floating point data type with the constraint NOT NULL.
- Reason: String data type with the constraint NOT NULL.
- Fine_Date: Date data type with the constraint NOT NULL

Fine_ID	Student_ID	Status	Amount	Reason	Fine_Date

Loan:

Loan entity has 4 fields : Loan_ID, Student_ID, ISBN, Loan_Date
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Loan_ID: Integer data type (PRIMARY KEY)
- Student_ID: Integer data type with the constraint NOT NULL
- ISBN: String data type with constraint NOT NULL

- Loan_Date: Date data type with the constraint NOT NULL.

Loan_ID	Student_ID	ISBN	Loan_Date

Book:

Book entity has 5 fields : ISBN, Book_Title, Author_ID, Publication_Year, Genre
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- ISBN: String data type (PRIMARY KEY)
- Book_Title: String data type with the constraint NOT NULL.
- Author_ID: Integer data type with the constraint NOT NULL.
- Publication_Year: Integer data type with the constraint NOT NULL.
- Genre: String data type with the constraint NOT NULL.

ISBN	Book_Title	Author_ID	Publication_Year	Genre

Student:

Student entity has 4 fields : Student_ID, Email Address, Student_Name, Phone_Number
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Student_ID: Integer data type (PRIMARY KEY)
- Email_Address: String data type with the constraint NOT NULL
- Student_Name: String data type with the constraint NOT NULL.
- Phone_Number: String data type with the constraint NOT NULL.

Student_ID	Email _Address	Student_Name	Phone_Number

Library Branch:

Library Branch entity has 6 fields : Branch_ID, Branch_Name, Contact_Number, City, Postal_Code, Street_Address
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Branch_ID: Integer data type (PRIMARY KEY)
- Branch_Name: String data type with the constraint NOT NULL
- Contact_Number: String data type with the constraint NOT NULL
- City: String data type with the constraint NOT NULL
- Postal_Code: String data type with the constraint NOT NULL

- Street_Address: String data type with the constraint NOT NULL

Branch_ID	Branch_Name	Contact_Number	City	Postal_Code	Street_Address

University Admin:

University admin entity has 5 fields : Admin_ID, Email_Address, Admin_Name, Phone_Number, Branch_ID

Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Admin_ID: Integer data type (PRIMARY KEY)
- Email_Address: String data type with the constraint NOT NULL
- Admin_Name: String data type with the constraint NOT NULL
- Phone_Number: String data type with the constraint NOT NULL
- Branch_ID: Integer data type

Admin_ID	Email_Address	Admin_Name	Phone_Number	Branch_ID

Relationships:

We have outlined **5** relationships between our entities as was the requirement of this project. Listed below are the relationships between our entities:

Book <> LibraryBranch
- Called 'can_contain'
- Each library can have many books, but each copy of a book can only belong to one librarybranch

Author <> Book
- Called 'published'
- An author can publish many books, but each book can only be published by one author

Student <> LibraryBranch
- Called 'part_of'
- A student can belong to many librarybranches, and each librarybranch can have many students

Branch <> Book
- Many-to-many relationship
- A branch can have many books, and a book can be in many branches

Loan <> Member
- Many-to-one relationship
- A book can have many loan records at different times, but a specific loan can only be applied to one book

Logical Database Design

Phase I

Assignment 1: Technical Report

In assignment 1 we created the first technical report of our project. It was mainly based on our initial planning of our project, how we wanted to implement our DBMS, planning on functions, entities, relationships, etc.

However, throughout the following assignments we kept updating and modifying our functions, entities , adding new relationships, etc.

This part was the initial database design and planning phase. Below are the screenshots of the report:

BiblioTech

A Library Database System

CPS 510 Section - 07

Instructor: Soheila Bashardoust Tajali

TA: Sepideh Banihashemi

Assignment 1: Technical Report of DBMS

Project Members:

1. Alizeh Zafar (Student No: 501279589)
2. Do Young Lee (Student No: 500961549)
3. Nidhi Biswas (Student No: 501118460)

1

Table of Contents

2

Project Description

Introduction:
With library users increasing day by day, it's important that libraries have up to date and modern database management systems. For our CPS 510 project, we have decided to work on a library database system. This will drastically improve their data management, user experience, as well as running their day-to-day operations smoothly.

In order to implement this, we will need to collect data and find the requirements. Then design our database and implement it. After that, we will test it to ensure that it is working as it is supposed to. And then, finally we will work on the maintenance of our database to ensure that any problems that come up are tackled.

Goal:
The purpose of our project is efficient book management, and building an improvised user experience for patrons in using our LMS by allowing them to search books, borrow books, etc.

Expected Learning Outcome:
By working on this DBMS project, we will also learn more about:

1. Designing a relational database management system (RDBMS)
2. Defining data types, structures, and constraints of the data in our database
3. Database Management (inserting, modifying, deleting data)
4. Creating entities such as a book entity, a new member entity, an author entity etc.
5. Data Integrity by avoiding inconsistencies so that any incorrect or inconsistent data is not stored in our system
6. Data efficiency by making sure there is no redundant data other than absolutely necessary
7. Relationships between entities and the different forms of relationships such as one-to-one, one-to-many, many-to-many
8. Data manipulation by querying the database using SQL to retrieve specific data
9. Security checks if necessary
10. Giving different users different access based on their needs. For eg, users and administrators will interact with the system on different levels
11. Parameters to ensure the data entered is valid
12. Maintenance of the database

Key Functions

Our library database system will have the following key functions which also adds both the admin's (administrator) role and patron's (user) role in each functions :

1. **Member Management**
   Admin's role: Manage membership data, new member's registration, update/modify existing member information, and process member withdrawals
   Patron's role: view/edit/add/delete informations from their profile, cancel membership, view borrowed books history and fine
2. **Log-in/Sign-up:**
   Admin's role: admins will have a separate login page for only authorized library members to access the system's administrative features
   Patron's role : Patron's can sign-up for membership or log-in if they are existing member
3. **Book Management**
   Admin's role: Register new books, edit and delete the information of existing books in the database,etc.
   Patron's role: browse books based on the search filter function, view book's information, check availability, reserve books, borrow books, track borrowed books for deadlines, etc.
4. **Loan**
   Admin's role: Manage availability of books, keep a record of the time/date the book was borrowed, keep a track of due dates of the books to be returned, etc.
   Patron's role: View deadlines of book return, status of their loaned book
5. **Overdue/Fine**
   Admin's role: Keep track of not-returned books, notify patrons, keep track of fine payments
   Patron's role: Receive a fine status for overdue books when they log-in to the system
6. **Search Filter**
   Admin's role: N/A
   Patron's role: Filter the search results based on Book attributes, view availability of books

4

Entities

Each of our entities will have various fields/attributes with defined data types and constraints.
We have outlined **6** entities as was the requirement of this project.
The tables below hold information about the entities in our Library Management DBMS:

Author:
Author entity will have 3 fields : First_Name, Last_Name, Author_ID.
Below is given the data types and constraints of each of our field/Attribute:
Fields/Attributes:

- First_Name: String data type with the constraint NOT NULL (which indicates this field cannot be empty).
- Last_Name: String data type with the constraint NOT NULL.
- Author_ID: Integer data type with the constraint NOT NULL.

First_Name	Last_Name	Author_ID

Fine:
Fine entity will have 6 fields : Member_ID, Amount, Reason, Fine_Date, Date_of_Payment, Status.
Below is given the data types and constraints of each of our field/Attribute:
Fields/Attributes:

- Member_ID: Integer data type with the constraint NOT NULL.
- Amount: Floating point data type with the constraint NOT NULL.
- Reason: String data type with the constraint NOT NULL.
- Fine_Date: Date data type.
- Date_of_Payment: Date data type with the constraint NOT NULL.
- Status: String data type with the constraint NOT NULL.

Member_ID	Amount	Reason	Fine_Date	Date_of_Payment	Status

5

Loan:

Loan entity will have 4 fields : Member_ID, Book_ID, Loan_Date, Due_Date
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Member_ID: Integer data type with the constraint NOT NULL.
- Book_ID: Integer data type with the constraint NOT NULL.
- Loan_Date: Date data type with the constraint NOT NULL.
- Due_Date: Date data type with the constraint NOT NULL.

Member_ID	Book_ID	Loan_Date	Due_Date

Book:

Book entity will have 5 fields : Title, Author, Publication_Year, ISBN, Genre
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Title: String data type with the constraint NOT NULL.
- Author: String data type with the constraint NOT NULL.
- Publication_Year: Integer data type with the constraint NOT NULL.
- ISBN: String data type with the constraint NOT NULL.
- Genre: String data type with the constraint NOT NULL.

Title	Author	Publication_Year	ISBN	Genre

Member:

Member entity will have 6 fields : Member_ID, First_Name, Last_Name, Home_Address, Phone_Number, Email Address
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Member_ID: Integer data type with the constraint NOT NULL.
- First_Name: String data type with the constraint NOT NULL.
- Last_Name: String data type with the constraint NOT NULL.
- Home_Address: String data type with the constraint NOT NULL.
- Phone_Number: String data type with the constraint NOT NULL.
- Email_Address: String data type with the constraint NOT NULL.

Member_ID	First_Name	Last_Name	Home_Address	Phone_Number	Email_Address

Library Branch:

Library Branch entity will have 4 fields : Branch_ID, Branch_Name, Telephone_Number, Address
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Branch_ID: Integer data type with the constraint NOT NULL.
- Branch_Name: String data type with the constraint NOT NULL.
- Telephone_Number: String data type with the constraint NOT NULL.
- Address: String data type with the constraint NOT NULL.

Branch_ID	Branch_Name	Telephone_Number	Address

Admin:

Admin entity will have 6 fields : Admin_ID, First_Name, Last_Name, Position, Email_Address, Phone_Number
Below is given the data types and constraints of each of our field/Attribute:

Fields/Attributes:

- Admin_ID: Integer data type with the constraint NOT NULL.
- First_Name: String data type with the constraint NOT NULL.
- Last_Name: String data type with the constraint NOT NULL.
- Position: String data type with the constraint NOT NULL.
- Email_Address: String data type with the constraint NOT NULL.
- Phone_Number: String data type with the constraint NOT NULL.

Admin_ID	First_Name	Last_Name	Position	Email_Address	Phone_Number

Relationships:

We have outlined **5** relationships between our entities as was the requirement of this project. Listed below are the relationships between our entities:

Author <> Book
- Many-to-one relationship
- Each author can write many books, but each book can have only one author

Loan <> Member
- Many-to-one relationship
- A member can have many loans, but each loan amount is associated with one member

Member <> Fine
- Many-to-one relationship
- A member can be fined many times, but a fine is unique to a member

Branch <> Book
- Many-to-many relationship
- A branch can have many books, and a book can be in many branches

Loan <> Book
- Many-to-one relationship
- A book can have many loan records at different times, but a specific loan can only be applied to one book

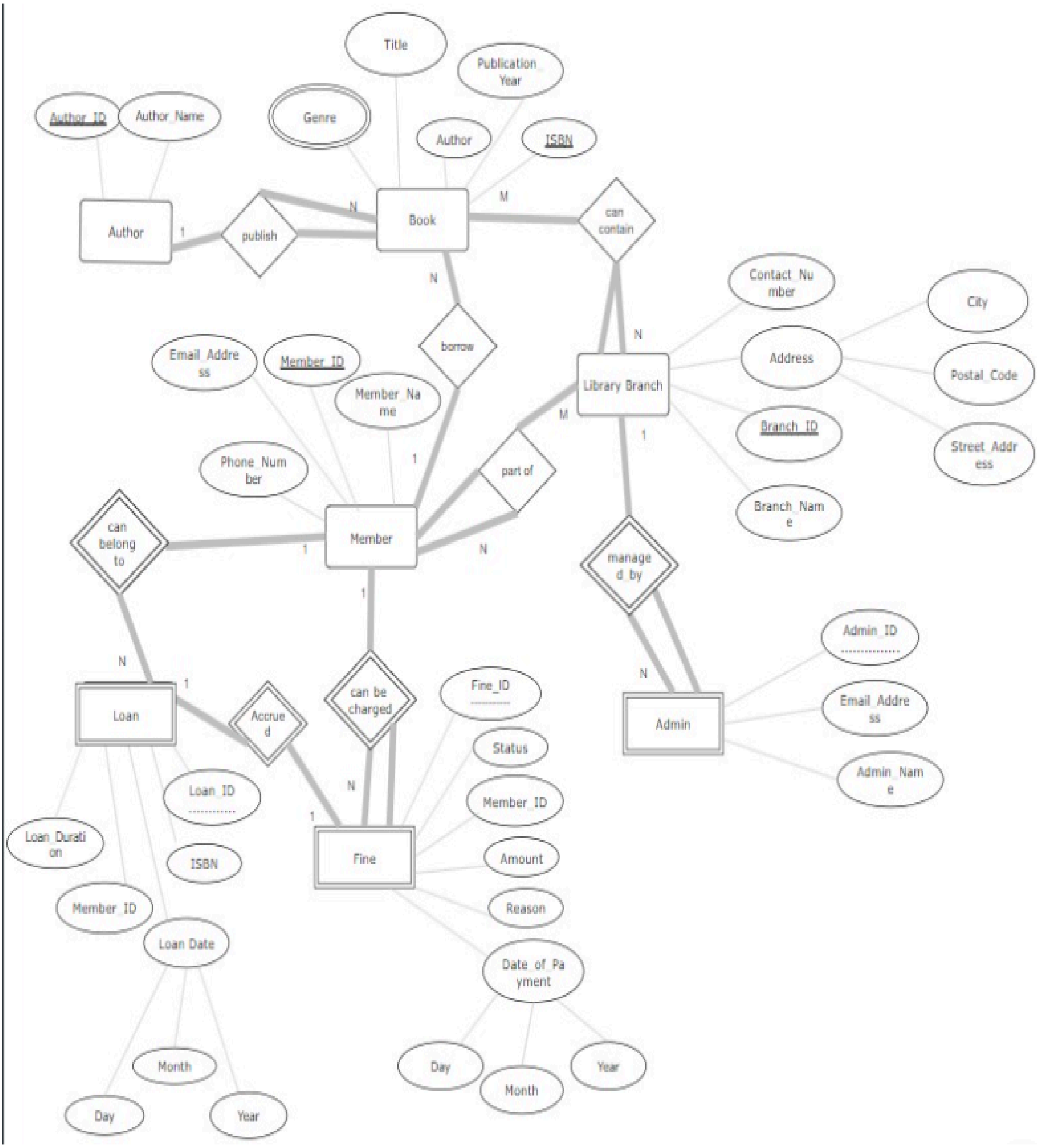
Sample SQL Queries

Here are some updates and SQL queries that can be applied to our database:

- Query: List the names of all current members of the library
- Query: List all the books by [*insert author ID here*]
- Query: List the most popular books in this library (books that have been the most borrowed by users)
- Update: Update a member's address or phone number
- Update: Insert new member data
- Update: the number of copies of a book

Assignment 2: ER Diagram

We created the ER diagram of our project. This was part of the database design phase. Below are the screenshots of our initial diagram:



Assignment 3: Queries

We created the tables in Oracle using our ER diagram. This was part of the database design phase. Below are the screenshots of our code in Oracle:

```sql
-- student entity table
CREATE TABLE student (
   student_id INTEGER PRIMARY KEY,
   email_address VARCHAR(100) NOT NULL,
   student_name VARCHAR(100) NOT NULL,
   phone_number VARCHAR(12) NOT NULL
) ;

-- librarybranch entity table
CREATE TABLE librarybranch (
   branch_id INTEGER PRIMARY KEY,
   branch_name VARCHAR(100) NOT NULL,
   contact_number VARCHAR(15) NOT NULL,
   city VARCHAR(100) NOT NULL,
   postal_code VARCHAR(20) NOT NULL,
   street_address VARCHAR(255) NOT NULL
);

-- university admin entity table
CREATE TABLE university_admin (
   admin_id INTEGER PRIMARY KEY,
   email_address VARCHAR(100) NOT NULL,
   admin_name VARCHAR(100) NOT NULL,
   phone_number VARCHAR(12) NOT NULL,
   branch_id INTEGER,
   FOREIGN KEY (branch_id) REFERENCES librarybranch(branch_id)
);

-- author entity table
CREATE TABLE author (
   author_id INTEGER PRIMARY KEY,
   author_name VARCHAR(100) NOT NULL

);

-- book entity table
CREATE TABLE book (
   ISBN VARCHAR(13) PRIMARY KEY,
   book_title VARCHAR(255) NOT NULL,
   author_id INTEGER NOT NULL,
```

```sql
    publication_year INT NOT NULL,
    genre VARCHAR(200)NOT NULL,
    FOREIGN KEY (author_id) REFERENCES author(author_id)
);

-- can_contain relation between book and Librarybranch entity
CREATE TABLE can_contain (
    ISBN VARCHAR(13),
    branch_id INTEGER,
    PRIMARY KEY (ISBN, branch_id),
    FOREIGN KEY (ISBN) REFERENCES book(ISBN),
    FOREIGN KEY (branch_id) REFERENCES librarybranch(branch_id)
);



-- published relation between author and book entity
CREATE TABLE published (
    author_ID INTEGER NOT NULL,
    ISBN VARCHAR(13)NOT NULL,
    PRIMARY KEY (author_ID, ISBN),
    FOREIGN KEY (author_ID) REFERENCES author(author_ID),
    FOREIGN KEY (ISBN) REFERENCES book(ISBN)
);

-- part_of relation between student and librarybranch entity
CREATE TABLE part_of (
    student_id INTEGER NOT NULL,
    branch_id INTEGER NOT NULL,
    PRIMARY KEY (student_id, branch_id),
    FOREIGN KEY (student_id) REFERENCES student(student_id),
    FOREIGN KEY (branch_id) REFERENCES librarybranch(branch_id)
);





-- book_fine entity table
CREATE TABLE book_fine (
    fine_id INTEGER PRIMARY KEY,
    student_id INTEGER NOT NULL,
    status VARCHAR(20) DEFAULT 'unpaid',
    amount DECIMAL(10, 2) NOT NULL,
    reason VARCHAR(255) NOT NULL,
```

```sql
    fine_date DATE NOT NULL,
    FOREIGN KEY (student_id) REFERENCES student(student_id)
);



-- librarybranch entity table
CREATE TABLE loan (
    loan_id INTEGER PRIMARY KEY,
    student_id INTEGER NOT NULL,
    ISBN VARCHAR(13) NOT NULL,
    loan_date DATE NOT NULL,
    FOREIGN KEY (student_id) REFERENCES student(student_id),
    FOREIGN KEY (ISBN) REFERENCES book(ISBN)
);







COMMIT;
```

Implementation Phase

Phase II

Assignment 4 Part 1: Simple Queries

For the first part of this assignment we designed some simple queries in Oracle after populating our database, at least one for each table.

Below are the screenshots of our code in Oracle:

```
SELECT student_name FROM student ORDER BY student_name; --List the names of all
current students of the library in alphabetical order

SELECT book_title FROM book WHERE author_id = 4325; --List all the books by [insert
author ID here]
SELECT book_title FROM book ORDER BY book_title; --listing all the books in
alphabetical order

UPDATE student SET email_address = 'ali567@gmail.com',  phone_number = 111222334
WHERE student_id = 5167; --Update a member??s address and phone number
SELECT email_address, phone_number FROM student WHERE student_id = 5167; --This is
to see if the information is updated

INSERT INTO student (student_id, email_address, student_name, phone_number) VALUES
(6754, 'jozy@gmail.com','Jozy Altidore', '444444437');
SELECT * FROM student WHERE student_id = 6754; --This is to see if the information
is updated

SELECT * FROM can_contain WHERE ISBN = '533443' AND branch_id = 1111; --This is
just to list the contents of the table

SELECT student_id, amount, reason FROM book_fine WHERE status = 'unpaid'; --List
student id, amount, reason who have not paid their fine yet

INSERT INTO author (author_id, author_name) VALUES (4332, 'J.K Rowling');
UPDATE author SET author_name = 'J.K. Rowling' WHERE author_id = 4332;
SELECT DISTINCT author_name FROM author;

UPDATE librarybranch SET branch_name = 'The Ryerson Library' WHERE branch_id =
3333;
```

```
SELECT loan_date FROM loan ORDER BY loan_date; --to view the upcoming loans in
order

SELECT * FROM part_of;

SELECT * FROM published;

UPDATE university_admin SET phone_number = '6663338898' WHERE admin_id = 1234;
SELECT * FROM university_admin WHERE admin_id = 1234;

SELECT DISTINCT email_address FROM student; --to get unique email addreses of
studens to prevent duplication

SELECT DISTINCT student_name FROM student;
```

Assignment 4 Part 2: Advanced queries and VIEWS

We designed more join advanced queries and VIEWS in Oracle after populating our database. Below are the screenshots of our code in Oracle:

```
--COMPLEX QUERIES USING JOIN
--student name and id of students who have borrowed books from the library

SELECT s.student_name, s.student_id, l.loan_id, b.ISBN
FROM student s, loan l, book b
WHERE s.student_id = l.student_id
AND l.ISBN = b.ISBN;

--list the student id of students who's fines are unpaid and is below or equals to
5$
SELECT s.student_id, s.student_name
FROM student s
WHERE EXISTS (
SELECT f.student_id, f.amount
FROM book_fine f
WHERE f.student_id = s.student_id
AND status='unpaid' AND amount <= 5.00
);


--list all the students who don't have a loan due
```

```sql
SELECT s.student_id, s.student_name
FROM student s
WHERE NOT EXISTS (
SELECT l.student_id
FROM loan l
WHERE l.student_id = s.student_id
)
ORDER BY s.student_id ASC;
```

Assignment 5: Advanced Queries and Unix shell Implementation

We designed some more advanced queries using join, set operations, aggregate functions, grouping queries, etc, as well as and showed UNIX shell menu implementation. Below are the screenshots of our queries and our UNIX shell menu:

```sql
--NEW COMPLEX QUERIES

--lists all students who have loaned out a book and who have unpaid fines
SELECT DISTINCT s.student_name, s.student_id
FROM student s
WHERE EXISTS (
SELECT *
FROM loan l
WHERE l.student_id = s.student_id
) AND EXISTS(
SELECT *
FROM book_fine bf
WHERE bf.student_id = s.student_id
AND bf.status = 'unpaid'
);

INSERT INTO loan (loan_id, student_id, ISBN, loan_date) VALUES (6548, 5368,
'333213', '2023-07-20');


--this is dividing the loaned out books by genre
SELECT b.genre, COUNT(b.ISBN) AS loaned_books
```

```
FROM book b
WHERE EXISTS (
   SELECT *
   FROM loan l
   WHERE l.ISBN = b.ISBN
)
GROUP BY b.genre;

INSERT INTO book_fine (fine_id, student_id, status, amount, reason, fine_date)
VALUES (1433, 5156, 'unpaid', 2.00, 'book lost', '2023-07-20');

--listing all students who have unpaid fines and summing up the total amount they
have remaining
SELECT
   s.student_id,
   s.student_name,
   SUM(bf.amount) AS total_fines
FROM student s, book_fine bf
WHERE s.student_id = bf.student_id AND bf.status = 'unpaid'

GROUP BY s.student_id, s.student_name
HAVING SUM(bf.amount) > 0;

--summing the total amount of all the unpaid fines
SELECT
SUM(amount) AS total_unpaid_fines
FROM book_fine
WHERE status = 'unpaid';


--lisint all the students who have paid fines or who have loaned out a book
SELECT DISTINCT s.student_name
FROM student s
WHERE EXISTS (
   SELECT *
   FROM book_fine bf
   WHERE s.student_id = bf.student_id
   AND bf.status = 'paid'
)
UNION
SELECT DISTINCT s.student_name
FROM student s
WHERE EXISTS (
   SELECT *
   FROM loan l
```

```
   WHERE s.student_id = l.student_id
);

--listing all the books that have not been loaned out yet
SELECT *
FROM book
MINUS
SELECT b.*
FROM book b, loan l
WHERE b.ISBN = l.ISBN;

COMMIT;
```

Application Development with shell scripts : we designed menus to perform the functions of our application by executing related Oracle SQL commands. Our menu had 5 functions : Drop Tables, Create Tables, Populate Tables, Query Tables, View Tables.

```
=================================================================
| BiblioTech Database Management System|
| Main Menu - Select Desired Operation(s):|
| <CTRL-Z Anytime to Enter Interactive CMD Prompt>|
-----------------------------------------------------------------
  M) View Manual

 1) Drop Tables
 2) Create Tables
 3) Populate Tables
 4) Query Tables
 5) View Tables

 X) Force/Stop/Kill Oracle DB

 E) End/Exit
Choose:
```

Assignment 6: Functional dependencies

We showed all the functional dependencies of our database. Below are the screenshots of our results:

Functional Dependencies Table

Table	Determinant (primary key)	Dependents
student	student_id	email_address, student_name, phone_number
librarybranch	branch_id	branch_name, contact_number, city, postal_code, street_address
university_admin	admin_id	email_address, admin_name, phone_number, branch_id
author	author_id	author_name
book	ISBN	book_title, author_id, publication_year, genre
can_contain	ISBN, branch_id	(for can_contain)
published	author_id, ISBN	(for published)
part_of	student_id, branch_id	(for part_of)
book_fine	fine_id	student_id, status, amount, reason, fine_date
loan	loan_id	student_id, ISBN, loan_date

Assignment 7 and 8: Normalization of our database

We normalized our database. Below are the screenshots of our results:

Verification for Tables Being in 3NF

Table	**Attributes**	**Determinant (primary key)**	**Description**	**3NF**
student	student_id, email_address, student_name, phone_number	student_id	-All non-key attributes are fully functionally dependent on determinant(student_id) -No transitive dependencies	Yes
librarybranch	branch_id, branch_name, contact_number, city, postal_code, street_address	branch_id	-All non-key attributes are fully functionally dependent on determinant(brnach_id) -Transitive dependency occurred between postal code and city(A specific postal code can determine specific city) -Therefore we removed the transitive dependency by removing the postal code	Yes
university_admin	admin_id, email_address, admin_name, phone_number, branch_id	admin_id	-All non-key attributes are fully functionally dependent on determinant(admin_id) -No transitive dependencies	Yes
author	author_id, author_name	author_id	-All non-key attributes are fully functionally dependent on determinant(author_id) -No transitive dependencies	Yes
book	ISBN, book_title, author_id, publication_year, genre	ISBN	-All non-key attributes are fully functionally dependent on determinant(ISBN) -No transitive dependencies	Yes
can_contain	ISBN, branch_id	ISBN, branch_id	-Only has combined primary key(branch_id, ISBN)	Yes
published	author_id, ISBN	author_id, ISBN	-Only has combined primary key(author id, ISBN)	Yes

part_of	student_id, branch_id	student_id, branch_id	-Only has combined primary key(student_id, branch_id)	Yes
book_fine	fine_id, student_id, status, amount, reason, fine_date	fine_id	-All non-key attributes are fully functionally dependent on determinant(fine_id) -No transitive dependencies	Yes
loan	loan_id, student_id, ISBN, loan_date	loan_id	-All non-key attributes are fully functionally dependent on determinant(loan_id) -No transitive dependencies	Yes

The functional dependencies(→)

student
student_id → email_address, student_name, phone_number

librarybranch
branch_id → branch_name, contact_number, city, street_address

university_admin
admin_id → email_address, admin_name, phone_number, branch_id

author
author_id → author_name

book
ISBN → book_title, author_id, publication_year, genre

can_contain
{ISBN, branch_id} → (for can_contain)

published
{author_id, ISBN} → (for published)

part_of
{student_id, branch_id} → (for part_of)

book_fine
fine_id → student_id, status, amount, reason, fine_date

loan
loan_id → student_id, ISBN, loan_date

Assignment 8

BCNF Verification

The functional dependencies(→)

student
student_id → email_address, student_name, phone_number

Since **student_id** is the primary key and thus a candidate key, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key student_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on student_id (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

librarybranch
branch_id → branch_name, contact_number, city, street_address

Since **branch_id** is the primary key and thus a candidate key, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key student_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on branch_id (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we only had one (city and postal code). Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies (removing the postal code didn't change anything in our database)

university_admin
admin_id → email_address, admin_name, phone_number, branch_id

Since **admin_id** is the primary key and thus a candidate key, this table satisfies BCNF

Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

can_contain
{ISBN, branch_id} → (for can_contain)

Since **ISBN, branch_id** are combined primary key and does not have any additional attribute, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key(combined primary key) ISBN, branch_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on ISBN, branch_id (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

published
{author_id, ISBN} → (for published)

Since **ISBN, author_id** are combined primary key and does not have any additional attribute, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key(combined primary key) author_id, ISBN.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on author_id, ISBN (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key student_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on admin_id (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

author
author_id → author_name

Since **author_id** is the primary key and thus a candidate key, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key student_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on author_id (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

book
ISBN → book_title, author_id, publication_year, genre

Since **ISBN** is the primary key and does not have partial dependencies, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key student_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on ISBN (the candidate key and superkey).

Step1: we listed all the attributes and FD's

part_of
{student_id, branch_id} → (for part_of)

Since **student_id, branch_id** are combined primary key and does not have any additional attribute, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key(combined primary key) student_id, branch_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on student_id, branch_id (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

book_fine
fine_id → student_id, status, amount, reason, fine_date

Since **fine_id** is the primary key and thus a candidate key, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key fine_id.
- This table satisfies 3NF and BCNF because all non-key attributes are fully dependent on fine_id (the candidate key and superkey).

Step1: we listed all the attributes and FD's
Step2: reduced the list of FD's, but in this case we didn't have many. Then we got a list of minimum FD's
Step3: identified the key's (left hand side)
Step4: we derived the final schema which was lossless and preserved all the other dependencies

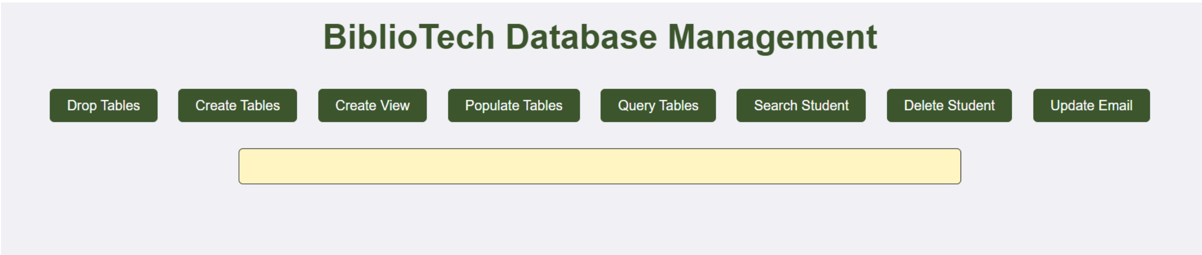
loan
loan_id → student_id, ISBN, loan_date

Since **loan_id** is the primary key, this table satisfies BCNF

- There are no partial or transitive dependencies, as each non-key attribute fully depends on the candidate key loan_id.

Assignment 9: Python based UI of our application

We made a GUI of our project. We used HTML, CSS, and Javascript as well as Python and Flask to create a web application of our project: This was part of the database implementation phase. Below are the screenshots of our GUI:



Documentation phase

Phase III

Assignment 10 and Relational Algebra (RA) Notation:

This is the final technical report of our project as part of the last assignment. We have also added the Relational Algebra (RA) notation. This is part of the documentation phase. Below are the RA notations of our SQL queries:

1. SORTstudent_name(πstudent_name(student))
2. πbook_title(σauthor_id=4325(book))
3. SORTbook_title(πbook_title(book))
4. πemail_address,phone_number(σstudent_id=5167(student))
5. student←student∪{(6759,’judyy@gmail.com’,’Joudy Altidore’,444474437,…)}
6. σstudent_id=6754(student)
7. σISBN=′533443′∧branch_id=1111(can_contain)
8. SORTpublication_year DESC(πbook_title,publication_year(σpublication_year>2004(book)))
9. πauthor_name(author)
10. πbook_title,genre(σgenre=′sci−fi′∨genre=′romance′(book))
11. πstudent_name,student_id,loan_id,ISBN((student⋈student_id=student_idloan)⋈ISBN=ISBNbook)
12. πstudent_id,student_name(student⋈student_id=student_idσstatus=’unpaid’∧amount≤5.00(book_fine))
13. SORTstudent_id ASC(πstudent_id,student_name(student−πstudent_id(σstudent_id=s.student_id(loan))))
14. πstudent_name,student_id(student⋈student_id=student_id(σstudent_id=s.student_id(loan))⋈student_id=student_id(σstudent_id=s.student_id∧status=′unpaid′(book_fine)))
15. γgenre;COUNT(ISBN)(πgenre,ISBN(book⋈ISBN=ISBNσISBN=b.ISBN(loan)))
16. γstudent_id,student_name;SUM(amount)(πstudent_id,student_name,amount(student⋈student_id=student_idσstatus=′unpaid′(book_fine)))
17. πstudent_name(σstatus=′paid′(book_fine)⋈student_id=student_idstudent)∪πstudent_name(σstudent_id=s.student_id(loan)⋈student_id=student_idstudent)
18. book−πISBN,title,…(book⋈ISBN=ISBNloan)