

Library Book Management System

1 Problem Domain Requirements

1.1 Business Rules:

- Students can borrow and return library books using the system.
- Administrators can add, delete, and modify library book information.
- Administrators can evaluate student borrowing records and blacklist students.
- Students can access library book information. They can search a specific book.
- Title refers to the basic information of a book, while copy represents a specific physical copy of the book. So one title can have 0 to many copies. Each copy can only have one title.
- Students can borrow many books at the same time.

1.2 Possible Nouns

book, student, administrator, title, copy, record, email

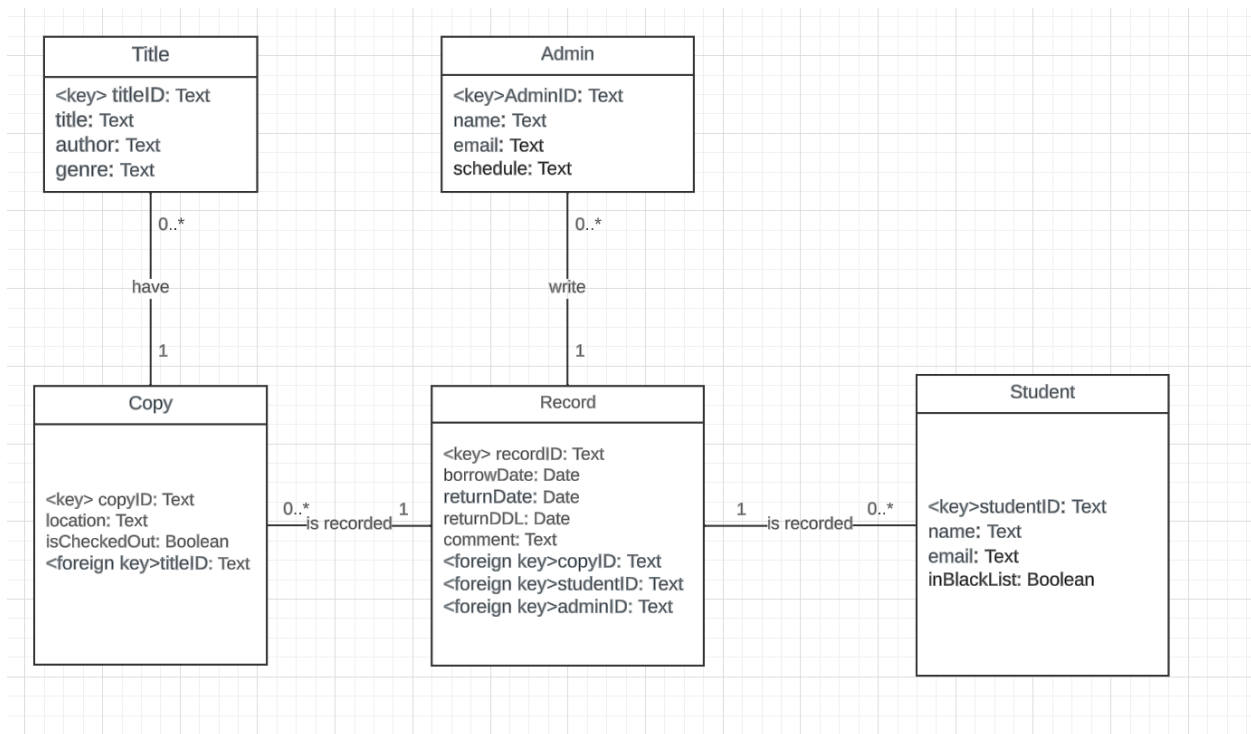
1.3 Possible Actions

- AddBook: To add a new book to the library's collection.
- DeleteBook: To remove a book from the library's collection.
- UpdateBook: To modify book information in the library.
- AddStudent: To register a new student in the system.
- BlacklistStudent: To restrict a student's borrowing privileges.
- UnblacklistStudent: To remove the restriction from a blacklisted student.
- BorrowBook: To check out a book from the library.
- ReturnBook: To return a borrowed book to the library.
- AddEvaluation: To provide feedback on a student's borrowing behavior.

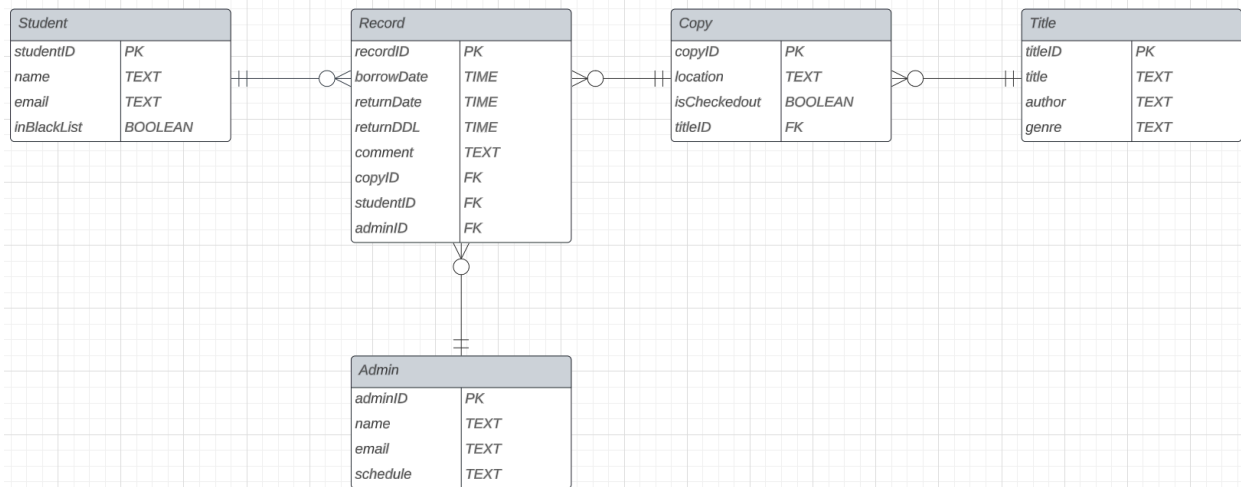
2 Conceptual model in UML

Explanation:

- Title refers to the basic information of a book, while Copy represents a specific physical copy of the book. The relationship between these two class is one-to-many.
- A student can borrow many copies. A copy can be borrowed by many students. The relationship between these two class is many-to-many



3 ERD



4 Define a relational schema

4.1 Schema

- Student(studentID, name, email, inBlackList)
- Record(recordID, borrowTime, returnTime, returnDDL, comment, *copyID*, *studentID*, *adminID*)
- Admin(adminID, name, email, schedule)
- Copy(copyID, location, isCheckedOut, *titleID*)
- Title(titleID, title, author, genre)

4.2 BCNF

A functional dependency exists between two sets of attributes in a relation when the value of one set of attributes uniquely determines the value of another set.

In table “Student”, we can use studentID uniquely determines other set’s value.

In table “Record”, we can use recordID uniquely determines other set’s value.

In table “Admin”, we can use adminID uniquely determines other set’s value.

In table “Copy”, we can use copyID uniquely determines other set’s value.

In table “Title”, we can use titleID uniquely determines other set’s value.

In conclusion, the schema is in at least BCNF.

5 Create Tables

5.1 SQL

```
CREATE TABLE IF NOT EXISTS Student (  
    studentID INTEGER PRIMARY KEY,  
    name TEXT,  
    email TEXT,  
    inBlackList BOOLEAN  
);
```

```
CREATE TABLE IF NOT EXISTS Record (  
    recordID INTEGER PRIMARY KEY,  
    borrowTime DATETIME,  
    returnTime DATETIME,  
    returnDDL DATETIME,  
    comment TEXT,  
    copyID INTEGER,  
    studentID INTEGER,  
    adminID INTEGER,  
    FOREIGN KEY (copyID) REFERENCES Copy(copyID),  
    FOREIGN KEY (studentID) REFERENCES Student(studentID),  
    FOREIGN KEY (adminID) REFERENCES Admin(adminID)  
);
```

```
CREATE TABLE IF NOT EXISTS Admin (  
    adminID INTEGER PRIMARY KEY,  
    name TEXT,  
    email TEXT,  
    schedule TEXT  
);
```

```
CREATE TABLE IF NOT EXISTS Copy (  
    copyID INTEGER PRIMARY KEY,  
    location TEXT,  
    isCheckedOut BOOLEAN,  
    titleID INTEGER,  
    FOREIGN KEY (titleID) REFERENCES Title(titleID)  
);
```

```
CREATE TABLE IF NOT EXISTS Title (  
    titleID INTEGER PRIMARY KEY,  
    title TEXT,  
    author TEXT,  
    genre TEXT  
);
```

5.2 Screenshots

▼	Tables (5)	
▼	Admin	CREATE TABLE Admin (adminID INTEGER PRIMARY KEY, name TEXT, email TEXT, schedule TEXT)
	admin...	"adminID" INTEGER
	name	"name" TEXT
	email	"email" TEXT
	sched...	"schedule" TEXT
▼	Copy	CREATE TABLE Copy (copyID INTEGER PRIMARY KEY, location TEXT, isCheckedOut BOOLEAN, titleID INTEGER,
	copyID	"copyID" INTEGER
	location	"location" TEXT
	isChe...	"isCheckedOut" BOOLEAN
	titleID	"titleID" INTEGER
▼	Record	CREATE TABLE Record (recordID INTEGER PRIMARY KEY, borrowTime DATETIME, returnTime DATETIME, return
	recor...	"recordID" INTEGER
	borro...	"borrowTime" DATETIME
	return...	"returnTime" DATETIME
	return...	"returnDDL" DATETIME
	comm...	"comment" TEXT
	copyID	"copyID" INTEGER
	stude...	"studentID" INTEGER
	admin...	"adminID" INTEGER
▼	Student	CREATE TABLE Student (studentID INTEGER PRIMARY KEY, name TEXT, email TEXT, inBlackList BOOLEAN)
	stude...	"studentID" INTEGER
	name	"name" TEXT
	email	"email" TEXT
	inBlac...	"inBlackList" BOOLEAN
▼	Title	CREATE TABLE Title (titleID INTEGER PRIMARY KEY, title TEXT, author TEXT, genre TEXT)
	titleID	"titleID" INTEGER
	title	"title" TEXT
	author	"author" TEXT
	genre	"genre" TEXT

6 Populate the tables with test data

I use chatGPT to help me generate the data and then check the data manually to make sure they satisfy the foreign key constrain. The result is as follows:

adminID	name	email	schedule
Filter	Filter	Filter	Filter
1	John Smith	john.smith@example.com	Monday to Friday, 9 AM - 5 PM
2	Alice Johnson	alice.johnson@example.com	Tuesday and Thursday, 10 AM - 6 PM
3	Michael Brown	michael.brown@example.com	Monday, Wednesday, and Friday, 8 AM - 4 PM
4	Emily Davis	emily.davis@example.com	Wednesday and Saturday, 11 AM - 7 PM
5	David Wilson	david.wilson@example.com	Thursday, 1 PM - 9 PM

studentID	name	email	inBlackList
Filter	Filter	Filter	Filter
1	Alice Johnson	alice.johnson@example.com	0
2	John Smith	john.smith@example.com	1
3	Michael Brown	michael.brown@example.com	0
4	Emily Davis	emily.davis@example.com	0
5	David Wilson	david.wilson@example.com	1

titleID	title	author	genre
Filter	Filter	Filter	Filter
1	The Great Gatsby	F. Scott Fitzgerald	Fiction
2	To Kill a Mockingbird	Harper Lee	Fiction
3	1984	George Orwell	Science Fiction

copyID	location	isCheckedOut	titleID
Filter	Filter	Filter	Filter
1	Shelf A, Section 1	0	1
2	Shelf B, Section 2	0	2
3	Shelf C, Section 3	0	1
4	Shelf D, Section 4	1	3
5	Shelf E, Section 1	0	2

recordID	borrowTime	returnTime	returnDDL	comment	copyID	studentID	adminID
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	2023-10-01 08:00:00	2023-10-03 16:00:00	2023-10-10 16:00:00	Good condition	1	1	1
2	2023-09-25 10:30:00	2023-09-28 14:15:00	2023-10-05 14:15:00	No issues	2	2	2
3	2023-10-05 15:45:00	2023-10-08 10:30:00	2023-10-15 10:30:00	Returned on time	3	3	3
4	2023-10-12 11:15:00	2023-10-14 17:30:00	2023-10-21 17:30:00	Minor damage	4	4	4
5	2023-09-28 13:00:00	2023-09-30 12:30:00	2023-10-07 12:30:00	Late return	5	5	5
6	2023-10-02 10:45:00	2023-10-04 14:00:00	2023-10-11 14:00:00	Excellent condition	1	2	1
7	2023-10-11 09:30:00	2023-10-14 15:45:00	2023-10-21 15:45:00	No issues	2	3	2
8	2023-10-03 08:30:00	2023-10-06 16:15:00	2023-10-13 16:15:00	Minor scratches	3	4	3
9	2023-09-26 14:45:00	2023-09-29 11:00:00	2023-10-06 11:00:00	Late return	4	5	4
10	2023-10-09 10:00:00	2023-10-12 12:30:00	2023-10-19 12:30:00	Good condition	5	1	5

7 Define and execute at least five queries

7.1 Query contains a join of at least three tables

```
SELECT
  R.borrowTime AS BorrowTime,
  R.returnTime AS ReturnTime,
  R.returnDDL AS ReturnDeadline,
  S.name AS StudentName,
  C.location AS CopyLocation,
  T.title AS TitleName
FROM Record AS R
JOIN Student AS S ON R.studentID = S.studentID
JOIN Copy AS C ON R.copyID = C.copyID
JOIN Title AS T ON C.titleID = T.titleID;
```

From this query, we can know the relationship between studentName and the TitleName clearly.

BorrowTime	ReturnTime	ReturnDeadline	StudentName	CopyLocation	TitleName
2023-10-01 08:00:00	2023-10-03 16:00:00	2023-10-10 16:00:00	Alice Johnson	Shelf A, Section 1	The Great Gatsby
2023-09-25 10:30:00	2023-09-28 14:15:00	2023-10-05 14:15:00	John Smith	Shelf B, Section 2	To Kill a Mockingbird
2023-10-05 15:45:00	2023-10-08 10:30:00	2023-10-15 10:30:00	Michael Brown	Shelf C, Section 3	The Great Gatsby
2023-10-12 11:15:00	2023-10-14 17:30:00	2023-10-21 17:30:00	Emily Davis	Shelf D, Section 4	1984

7.2 Query contains a subquery

```
SELECT S.name AS StudentName
FROM Student AS S
WHERE (
  SELECT COUNT(*)
  FROM Record AS R
  WHERE R.studentID = S.studentID
) > (
  SELECT AVG(BorrowCount)
  FROM (
    SELECT studentID, COUNT(*) AS BorrowCount
    FROM Record
    GROUP BY studentID
  ) AS StudentBorrowCount
);
```

From this query, we can calculate the names of students who have borrowed books more frequently than the average

	StudentName
1	Alice Johnson

7.3 Group by with a having clause

```
SELECT studentID, COUNT(*) AS BorrowCount
FROM Record
GROUP BY studentID
HAVING COUNT(*) > 3;
```

From this query, we can know the id of students who have borrowed books three times more

studentID	BorrowCount
1	4

7.4 Complex search criterion

```
SELECT *
FROM Record
WHERE (returnTime < returnDDL) AND (borrowTime BETWEEN '2023-10-10 00:00:00' AND '2023-10-15 23:59:59')
```

From this query, we can know the information of record that returned on time and borrowed between 2023-10-10 and 2023-10-15

recordID	borrowTime	returnTime	returnDDL	comment	copyID	studentID	adminID
4	2023-10-12 11:15:00	2023-10-14 17:30:00	2023-10-21 17:30:00	Minor damage	4	4	4
7	2023-10-11 09:30:00	2023-10-14 15:45:00	2023-10-21 15:45:00	No issues	2	3	2

```
7.5 Experiment with advanced query mechanisms: SELECT
CASE/WHEN
SELECT recordID ,
case
    when returnTime<returnDDL then "on time"
    when returnTime>returnDDL then "late"
    end as status
FROM Record;
```

From this query, we can see clearly which record is late or not

recordID	status
1	on time
2	on time
3	on time
4	on time
5	late
6	on time
7	on time
8	on time

8 Create a basic Node + Express application