

Library Book Management System

1 Problem Requirements

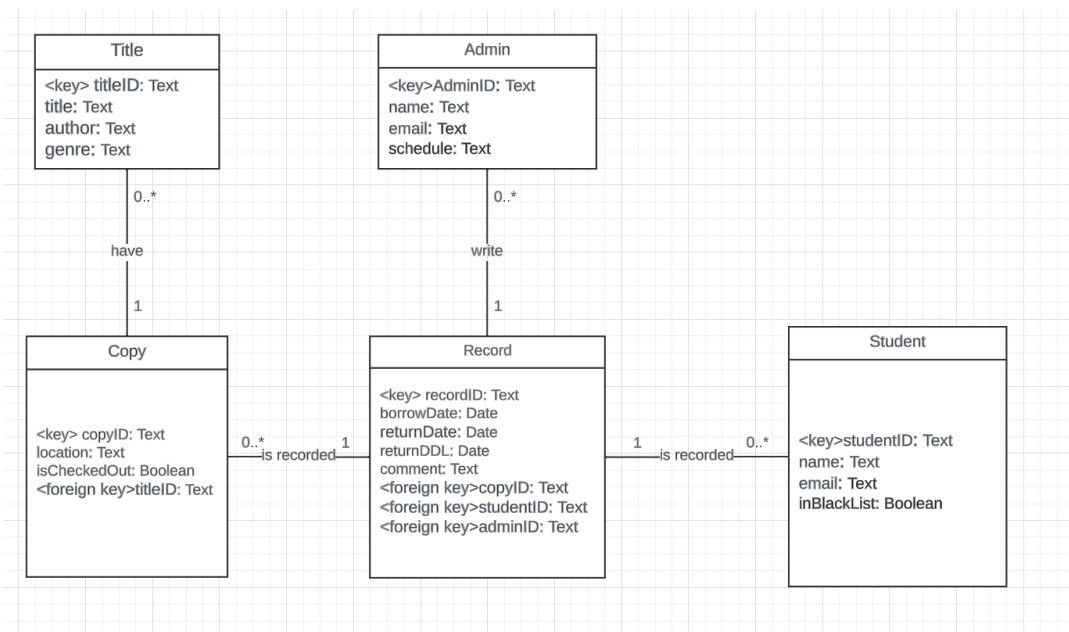
1.1 Business Rules:

- 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.

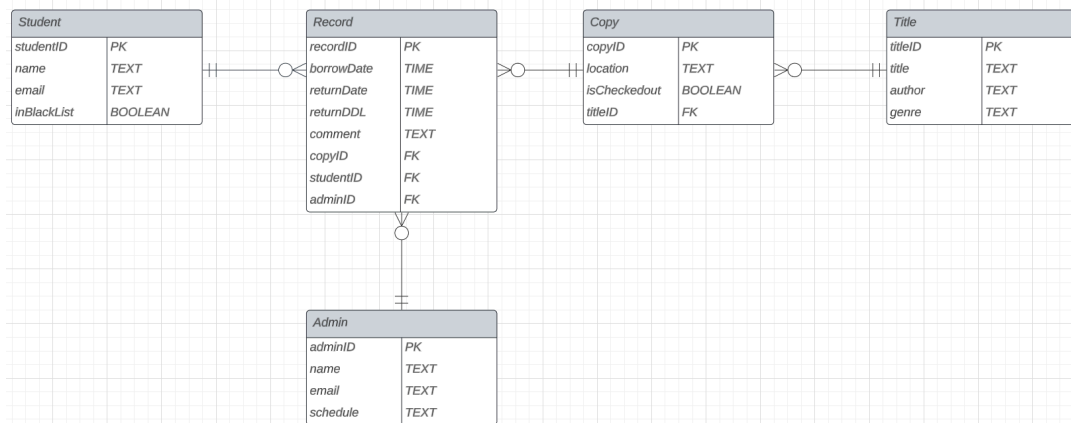
1.2 Conceptual model in UML

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



1.3 ERD



1.4 Functionalities be used with Redis

I plan to store each book's copies in Redis as this part can be queried frequently in real life. The logic is as follows:

If we query a book's copy, then we first look to Redis. If we find the result, just return to the front-end; Otherwise we read data from Sqlite and save the result to Redis, then return to the front-end.

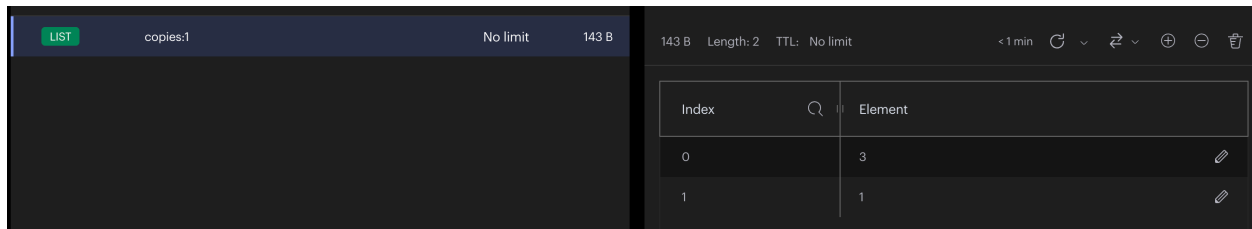
If we add a book's copy, we add it both in Redis and Sqlite.

If we modify a book's copy, we modify it both in Redis and Sqlite.

If we delete a book's copy, we delete it both in Redis and Sqlite.

2 Describe the Redis data structures

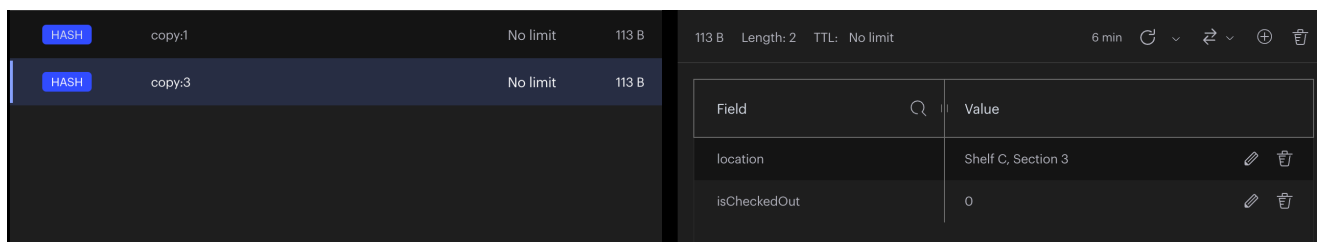
In Redis, because a book can have many copies, we decide to use list to store this information, where the key is book's id and value is a list of copy id. Just like this:



The screenshot shows the Redis UI for a list structure. The left sidebar has a green 'LIST' button. The main area shows the key 'copies:1' with a value of 'No limit' and a size of '143 B'. The right sidebar shows '143 B Length: 2 TTL: No limit' and a table with 2 elements.

Index	Element
0	3
1	1

Then for each copy id ,we use it as key to find a hashTable. In this hashTable, the key is the field in copy(location, isCheckedOut...) Just like this:



The screenshot shows the Redis UI for a hash table structure. The left sidebar has a blue 'HASH' button. The main area shows the key 'copy:3' with a value of 'No limit' and a size of '113 B'. The right sidebar shows '113 B Length: 2 TTL: No limit' and a table with 2 fields.

Field	Value
location	Shelf C, Section 3
isCheckedOut	0

3 Create a basic Node + Express application

This is the log in page

Library System

Name

Password

Log in

In this page, we can find all books and their copies.
The operations are “add new book”, “delete a book” or “edit a book”

Library System

Find books and their copies

Add a Book

Book ID: 1

Title: The Great Gatsby
Author: F. Scott Fitzgerald
Genre: Fiction

[View copies](#) [Edit](#) [Delete](#)

Book ID: 2

Title: To Kill a Mockingbird
Author: Harper Lee
Genre: Fiction

[View copies](#) [Edit](#) [Delete](#)

Book ID: 3

Title: 1984
Author: George Orwell
Genre: Science Fiction

[View copies](#) [Edit](#) [Delete](#)

Book ID: 4

Title: te
Author: te
Genre: te

[View copies](#) [Edit](#) [Delete](#)

In this page, we can see all the copies of a specific book. And we can also add, edit or delete a copy

Library System

Find all copies of this book

Add a copy

Copy ID: 1

Location: Shelf A, Section 1

Edit

Delete

Copy ID: 3

Location: Shelf C, Section 3

Edit

Delete

There are codes about Redis when querying the copies:

```
// check whether the Redis has the information, if not, update Redis
const test = await redisClient.get("copy"+titleID);
if (test === null) {
  console.log(copies)
  const copyIds = [];
  for (const copy of copies) {
    copyIds.push(copy.copyID);
  }
  console.log(copyIds)
  // create list of copyId in redis
  for (const copyId of copyIds) {
    await redisClient.LPUSH(`copies:${titleID}`, copyId.toString());
  }
  // create copyID-->hashtable(location,isCheckedOut) in redis
  copies.forEach((copy) => {
    const hashKey = `copy:${copy.copyID}`;
    const hashData = {
      location: copy.location,
      isCheckedOut: copy.isCheckedOut,
    };
    redisClient.HSET(hashKey, hashData, (hashError) => {
      if (hashError) {
        return callback(hashError);
      }
      console.log(`Hash created for copy ID ${copy.id} successfully.`);
    });
  });
};
```