Course Project - Library Database Management System
Implemented from scratch in C++
Course Project CSL 3050

Database Systems

Chirag (B19CSE026)[1] , Bhawna Chopra(B19CSE104)[2],Darsh Patel(B19CSE115)[3]

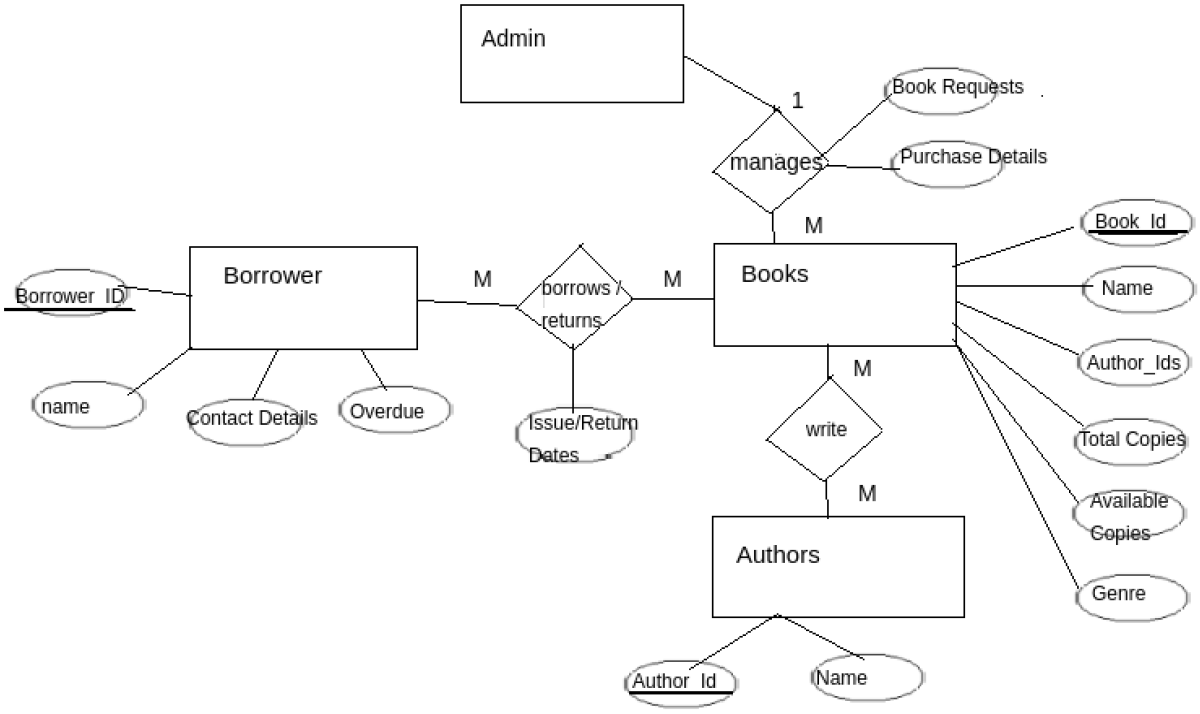
BTech Students, Dept of CSE, IIT Jodhpur

## I. Introduction

This document describes the library management system implemented from scratch in C++. This document covers the logical structure and implementation details of the database system in question. The logical structure is presented with the help of ER Model. To reduce redundancy among tables posed by the ER model, normalization of all tables is done till BCNF. The implementation of the model is done in C++.

## II. ER Model and diagram

In our library management system, as shown in the diagram we have the following entities and relation:



So according to the ER model presented above, we created the following Tables:

1. Book_Details for Books Entity
2. Author_Details for Author Entity
3. Borrower_Details for Borrower Entity
4. Book_Issue_Details for "Borrower borrows Books Relation"
5. Book_Purchase_Details for "Admin manages Books Relation"
6. Book_Requests for "Admin manages Books Relation"

## III. Tables

The tables in our library database management system are as follows, we have normalized every table to BCNF to avoid redundancy.

1. `Book_Details{book_id (A); book_name(B); author_id_1(C);author_id_2 (D);author_id_3 (E);genre (F);version (G);total_copies (H);available_copies (I);}`

This table contains details about books present in the library such as name of books, authors of book, genre, version, total number of copies in the library and available number of copies of each book (i.e number of books left out after renting to borrowers).

book_id is a unique number assigned to each book in the library.

author_ids are foreign keys referring to the author_id of the Author_Details table (shown next).

We have assumed that a book can have a maximum of 3 authors and at least. So if a book has only one author, then the other 2 columns will remain null.

The functional dependencies (FDs) present in the above table are:

A→ B,C,D,E,F,G,H,I

Candidate keys ={A }

Non Prime Attributes = {B,C,D,E,F,G,H,I}

Here there are no multivalued attributes, hence the table is in 1NF.

No partial dependencies are present for the candidate key A, hence the table is in 2NF.

Transitive dependencies are absent among non prime attributes, hence the table is in 3NF.

The determinants in all FDs are superkeys, hence the table is in BCNF.

2. `Author_Details{author_id (A);author_name (B);};`

This table has details about authors. author_id is a unique numeric key assigned to each author.

The functional dependencies (FDs) present in the above table are:

A→ B

Candidate keys ={A }

Non Prime Attributes = { B}

Here there are no multivalued attributes, hence the table is in 1NF.

No partial dependencies are present for the candidate key A, hence the table is in 2NF.

Transitive dependencies are absent among non prime attributes, hence the table is in 3NF.

The determinants in all FDs are superkeys, hence the table is in BCNF.

3. `Book_Purchase_Details{book_id (A);purchase_date (B);copies (C); price(D);};`

This table has details about purchases made by a library. book_id is a foreign key referring to book_id of Book_details.

The functional dependencies (FDs) present in the above table are:

AB→ CD

Candidate keys ={AB}

Non Prime Attributes = { C,D }

Here there are no multivalued attributes, hence the table is in 1NF.

No partial dependencies are present for the candidate key AB, as both book_id and date together determine copies and price, hence the table is in 2NF.

Transitive dependencies are absent among non prime attributes, hence the table is in 3NF.

The determinants in all FDs are superkeys, hence the table is in BCNF.

4. `Borrower_Details{`
`borrower_id(A);name(B);dob(C);contact_num(D);email(E);due_amount(F);};`

This table has details about borrowers who borrow books from the library. borrower_id is a unique number assigned to each borrower of the library.

The functional dependencies (FDs) present in the above table are:

A→BCDEF

E→ABCDF

Candidate keys ={A,E}

Non Prime Attributes = {B,C,D,F}

Here there are no multivalued attributes, hence the table is in 1NF.

No partial dependencies are present for any candidate key A or E, hence the table is in 2NF.

Transitive dependencies are absent among non prime attributes, hence the table is in 3NF.

The determinants in all FDs are superkeys, hence the table is in BCNF.

5. Book_Issue_Details{borrower_id(A);book_id(B); issue_date(C); due_date(D);};

This table has details about the issued books. It contains information like which book is borrowed by whom ,when and when is the due date.

borrower_id is a foreign key referring to borrower_id of the Borrower_Details table.

book_id is a foreign key referring to book_id of the Book_Details table.

The functional dependencies (FDs) present in the above table are:

AB→CD

Candidate keys ={AB}

Non Prime Attributes = { C,D }

Here there are no multivalued attributes, hence the table is in 1NF.

No partial dependencies are present for any subset of candidate key A or B, hence the table is in 2NF.

Transitive dependencies are absent among non prime attributes, hence the table is in 3NF.

The determinants in all FDs are superkeys, hence the table is in BCNF.

6. Book_Requests {request_id(A),book_name(B);author_name(C);version(D);};

The functional dependencies (FDs) present in the above table are:

A→BCD

Candidate keys ={A }

Non Prime Attributes = {B, C,D }

Here there are no multivalued attributes, hence the table is in 1NF.

No partial dependencies are present for any subset of candidate key A , hence the table is in 2NF.

Transitive dependencies are absent among non prime attributes, hence the table is in 3NF.

The determinants in all FDs are superkeys, hence the table is in BCNF.

## IV. Integrity Constraints

Entity integrity constraint: Here for primary keys like book_id, author_id and borrower_id we are assigning not null unique numbers by using increment.

Referential integrity constraints:

There are following constraints:

| Primary Key | Foreign Key |
|---|---|
| Author_Details.author_id | Book_Details.author_id |
| Borrower_Details.borrower_id | Book_Issue_Details.borrower_id |
| Book_Details.book_id | Book_Purchase_Details.book_id |
| Book_Details.book_id | Book_Issue_Details.book_id |

## V. Triggers

The following triggers are implemented:

1. Tuple insertion/deletion of Book_issue_details leads to updation of available_copies of Book_Details.

   → If a book is issued then a tuple will be inserted in Book_issue_details and the available_copies will be reduced by 1. Here we are assuming that one borrower can take at most 1 copy of a particular book.

   → If a book is returned then a tuple will be deleted from Book_issue_details and the available_copies will be incremented by 1.

2. Tuple deletion of Book_issue_details leads to updation of due_amount of Borrower_Details.

   → Because we delete a tuple of Book_issue_details when the book is returned and if late, the dues can be updated in the Borrower_Details.

3. Tuple insertion/deletion of Book_Purchase_Details leads to updation of available_copies of Book_Details.

   → If a book is purchased, we will insert a tuple in Book_Purchase_Details and then the available_copies will be incremented.

   → If a book is returned to the seller due to some issue then a tuple will be deleted in Book_Purchase_Details and then the available_copies will be decremented.

## VI. Queries And Optimization:

We implemented queries which are mostly used in a library scenario like:

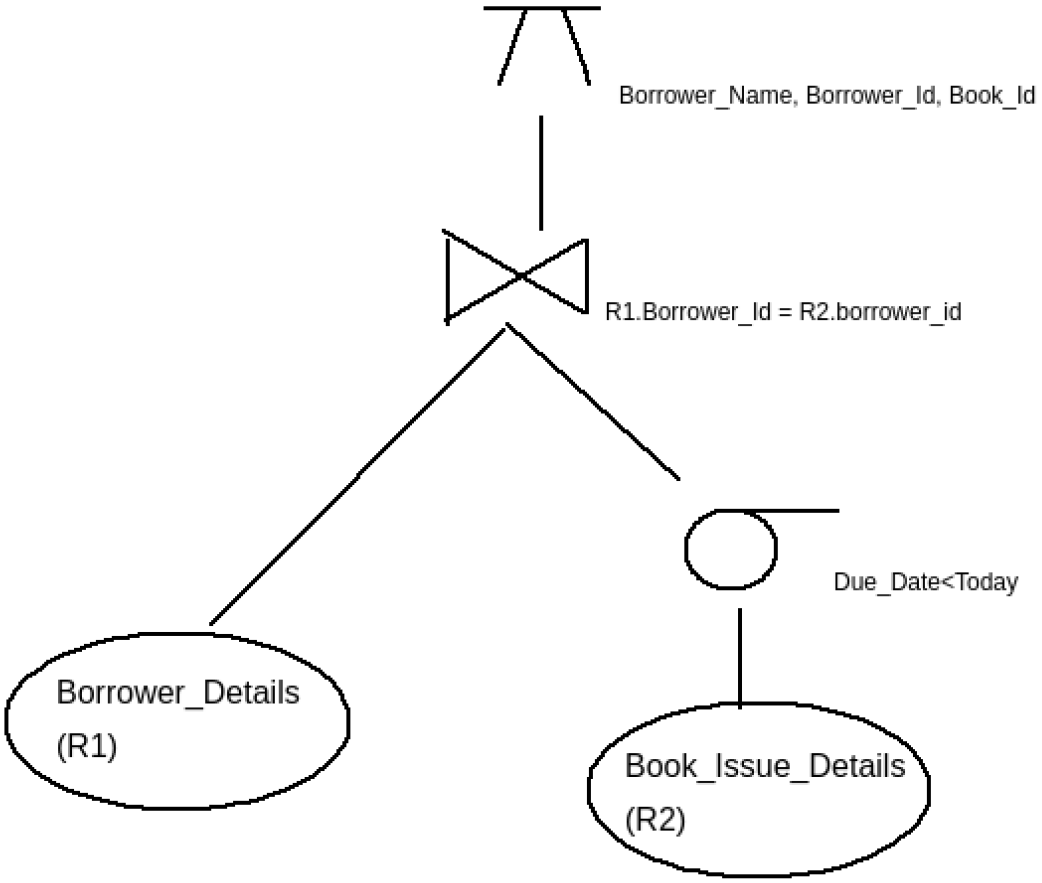
1. Listing all Books ( type: normal SELECT query)
2. Listing all Borrower ( type: normal SELECT query)
3. Listing all Borrowers who have not returned books after the due date. ( type: SELECT query with JOIN )

etc.

For complex queries ( like the 3rd one with JOIN ), we have tried to implement less expensive plans.

For E.g for the 3rd query:

As shown in the query tree below, we are pushing **σ** (relational algebra select) as low as possible so that the number of tuples required in the join above are reduced as joins tend to be expensive.




## VII. Hashing:

In order to expedite common queries like search by book_name, we created a hash table using Linear Hashing which uses book_name as index.

We used the sum of ascii values of characters in book_name for indexing and using modulo as the hash function.

## VIII. Results and Conclusion:

The demo of the running code can be found at the following link:
https://drive.google.com/file/d/1aq7jdteiAbzjGL9Aj3IDPnBvHMQrjKjt/view?usp=sharing

## IX. Contributions

The end-to-end implementation of the system was thoroughly discussed among the team members. All minute details were debated over and decided. Every decision was made unanimously.

The work for writing the code was divided as below.

*Chirag (B19CSE043)* - Wrote the code for database creation from text files, queries, integrity constraints and triggers.

*Darsh Patel (B19CSE115)* - Wrote the code for creating classes, hashing and calculating number of days between dates in string format. Co-authored this report.

*Bhawna Chopra(B19CSE104)* - Wrote all the text files for the database in required format. Co-authored this report.

Except for code writing, all the other activities were done together. Documentation, verification, and finalization were also done together.

## ACKNOWLEDGMENT

.

Dr. Romi Banerjee and all TAs of the course, CSL 3050.