INDEX

CHAPTER 1

INTRODUCTION

In the evolving landscape of information management, effective database systems play a pivotal role in facilitating seamless access to resources and enhancing organizational efficiency. Libraries, serving as repositories of knowledge, rely heavily on robust database management systems to organize, store, and retrieve vast amounts of bibliographic data. The integration of advanced technologies, such as C++ programming and SQL databases, offers a promising avenue for enhancing the capabilities of library database management systems.

This report explores the amalgamation of C++ and SQL to develop a sophisticated library database management system that caters to the diverse needs of users and administrators alike. The project endeavors to address the inherent challenges faced by conventional library systems, including data redundancy, limited accessibility, and cumbersome maintenance processes. Through a systematic analysis of system requirements and a meticulous examination of existing methodologies, this project aims to design and implement a solution that leverages the strengths of both C++ and SQL.
By harnessing the flexibility and performance of C++ programming, coupled with the robust data management capabilities of SQL databases, the envisioned system endeavors to streamline library operations, empower users with intuitive access to resources, and provide administrators with comprehensive tools for database administration.

Key components of this project include the development of user and administration login functionalities using C++, seamless integration of SQL queries within C++ code for data manipulation, and the implementation of a well-structured database schema tailored to the specific needs of library management. By adhering to best practices in software engineering and database design, this project aims to deliver a scalable, efficient, and user-friendly solution that addresses the evolving demands of modern library management.

Through this report, readers will gain insights into the intricacies of integrating C++ programming with SQL databases, the challenges encountered during the development process, and the potential implications of adopting such technologies in library settings. As we embark on this journey of exploration and innovation, we invite readers to delve deeper into the intricacies of our project, and join us in envisioning a future where technology empowers libraries to thrive in the digital age.

CHAPTER 2

SYSTEM ANALYSIS

In the initial phase of this project, a comprehensive system analysis was conducted to identify the requirements and challenges inherent in library database management. The primary goal was to understand the needs of both users and administrators, assess existing systems, and define clear objectives for the development of a robust solution.

Requirements Gathering:

Requirements gathering involved engaging with stakeholders, including librarians, administrators, and potential users, to elicit their expectations and preferences for the new system. Through interviews, surveys, and analysis of user feedback, key functional and non-functional requirements were identified. These included:

- **User Authentication**: The system must provide secure login mechanisms for both users and administrators to ensure data privacy and integrity.
- **Data Management**: Efficient handling of bibliographic data, including book titles, authors, editions, availability status, and user borrowing records.
- **Search and Retrieval**: Intuitive search functionality allowing users to locate resources based on various criteria such as title, author, genre, and ISBN.
- **User Interface**: A user-friendly interface that enhances accessibility and usability, catering to users of diverse technical backgrounds.
- **Administration Tools**: Tools for administrators to manage database operations, including data insertion, modification, deletion, and user management.
- **Performance and Scalability**: The system should be capable of handling large volumes of data efficiently and scale seamlessly as the library collection grows.
- **Security and Backup:** Implementation of robust security measures to prevent unauthorized access and ensure data integrity, with provisions for regular backups to prevent data loss.

Analysis of Existing Systems:

A critical analysis of existing library management systems was conducted to understand their strengths, weaknesses, and limitations. This involved studying off-the-shelf solutions, open-source platforms, and proprietary systems commonly used in library settings. Common issues observed in traditional systems included:

- **Limited Flexibility:** Many existing systems lacked flexibility in customization, making it challenging to adapt to the unique requirements of different libraries.
- **Complexity:** Some systems were overly complex, leading to usability issues and steep learning curves for users and administrators.
- **Performance Bottlenecks:** Inefficient data handling and retrieval mechanisms resulted in performance bottlenecks, especially in large library collections.

- **Lack of Integration:** Few systems offered seamless integration with external technologies or programming languages, limiting their extensibility and interoperability.

Use Case Scenarios:

Use case scenarios were developed to illustrate the various interactions between users, administrators, and the system itself. These scenarios helped to identify key functionalities and user workflows, guiding the design and development process. Use cases included:

- **User Login and Search:** A user logs into the system, searches for a book by title, and checks its availability status.
- **Book Checkout:** A user selects a book for checkout, and the system updates the database to reflect the borrowing transaction.
- **Administrator Data Management:** An administrator adds a new book to the database, modifies existing records, and generates reports on library usage statistics.

By conducting a thorough system analysis, we gained valuable insights into the requirements, challenges, and opportunities inherent in library database management. These insights informed the subsequent phases of system design, implementation, and testing, ensuring that the final solution addresses the needs of stakeholders effectively and efficiently.

CHAPTER 3

Integration of SQL and C++

The integration of SQL and C++ forms the cornerstone of our project, enabling seamless interaction between the application layer (C++) and the database layer (SQL). This section delineates the methodologies employed to harmonize these technologies and the rationale behind their selection.

1. Choice of SQL:

- Data Management Capabilities: SQL (Structured Query Language) was chosen for its robust data management capabilities, including data retrieval, manipulation, and storage.
- Standardization: SQL is an industry-standard language for interacting with relational databases, ensuring compatibility and interoperability across various database management systems.

2. Integration Strategies:

- Database Connectivity Libraries: To facilitate interaction between C++ and SQL, we utilized database connectivity libraries such as ODBC (Open Database Connectivity) or specific database APIs provided by vendors (e.g., MySQL Connector/C++ for MySQL databases).
- Object-Relational Mapping (ORM): Alternatively, ORM frameworks like Hibernate or Objectivity/DB can abstract database operations, enabling developers to work with C++ objects directly while transparently handling database interactions.

3. Implementation Approach:

- Connection Establishment: Our system initiates a connection to the SQL database upon application startup, utilizing connection strings or configuration parameters to specify database connection details.
- Query Execution: SQL queries are formulated within the C++ codebase using appropriate syntax and are subsequently executed through the established database connection.
- Error Handling: Robust error handling mechanisms are implemented to manage exceptions arising from SQL operations, ensuring graceful recovery and user feedback in case of failures.

4. Advantages of Integration:

- Performance Optimization: Leveraging SQL for database operations offloads computational overhead from the application layer, enhancing system performance and scalability.
- Data Integrity: By utilizing SQL transactions and constraints, our system ensures data integrity and consistency, mitigating risks associated with concurrent access and data manipulation.
- Modularity and Maintainability: Separating database logic from application logic promotes modularity and maintainability, facilitating easier debugging, testing, and future enhancements.

CHAPTER 4

SYSTEM DESIGN

The system design phase of the advanced library database management project encompasses the architectural blueprint, database schema design, and essential diagrams elucidating the system's functionality.

1. Architecture Overview:
The proposed library management system follows a client-server architecture to facilitate concurrent access and seamless interaction between multiple users. The architecture comprises the following components:

- Client Interface: Provides a user-friendly interface for patrons and librarians to interact with the system. Implemented using a graphical user interface (GUI) developed in C++.

- Application Logic: Contains the core functionalities of the system, including database interactions, business logic, and user authentication. Implemented in C++ to ensure platform independence and performance efficiency.

- Database Management System (DBMS): Serves as the backend storage for library data, including book records, patron information, and transaction logs. Utilizes a SQL-based DBMS (e.g., MySQL, SQLite) for efficient data management and retrieval.

- Network Communication: Facilitates communication between the client interface, application logic, and database backend. Utilizes TCP/IP sockets or HTTP protocols to ensure reliable data exchange over the network.

2. Database Schema Design:

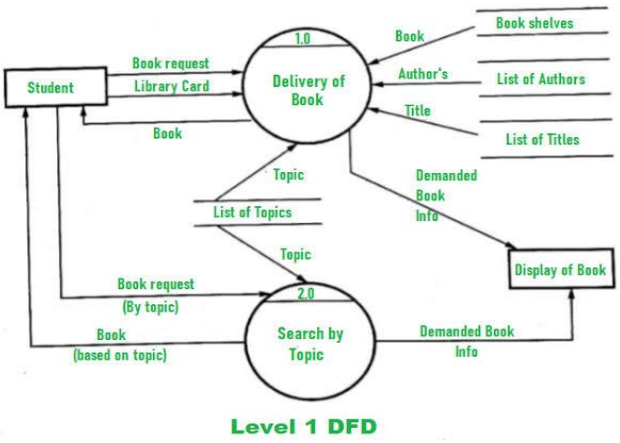
The database schema design forms the backbone of the system, defining the structure and relationships among various entities. The schema consists of the following tables:

- Books: Stores information about library books, including title, author, ISBN, genre, availability status, and location.

- Patrons: Contains details of library patrons, such as name, contact information, membership status, and borrowing history.

- Transactions: Records all transactions within the library, including book checkouts, returns, renewals, and reservations.

- Administrators: Stores credentials and access rights for system administrators, enabling secure user authentication and authorization.



Level 1 DFD

3. Use Case Diagrams:

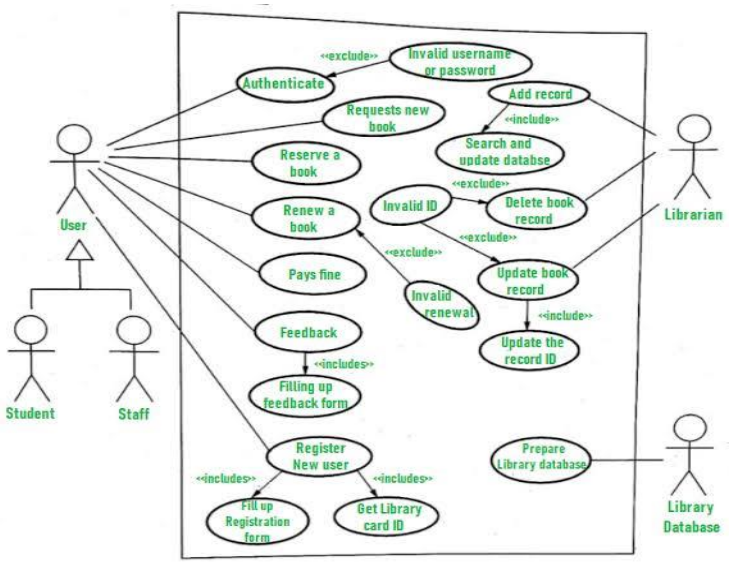
Use case diagrams illustrate the interaction between system actors (users) and the various functionalities offered by the system. Key use cases include:

- Search Books: Allows users to search for books based on title, author, genre, or ISBN.

- Borrow Book: Enables patrons to borrow books from the library, updating availability status and recording transaction details.

- Return Book: Allows patrons to return borrowed books, updating availability status and transaction logs accordingly.

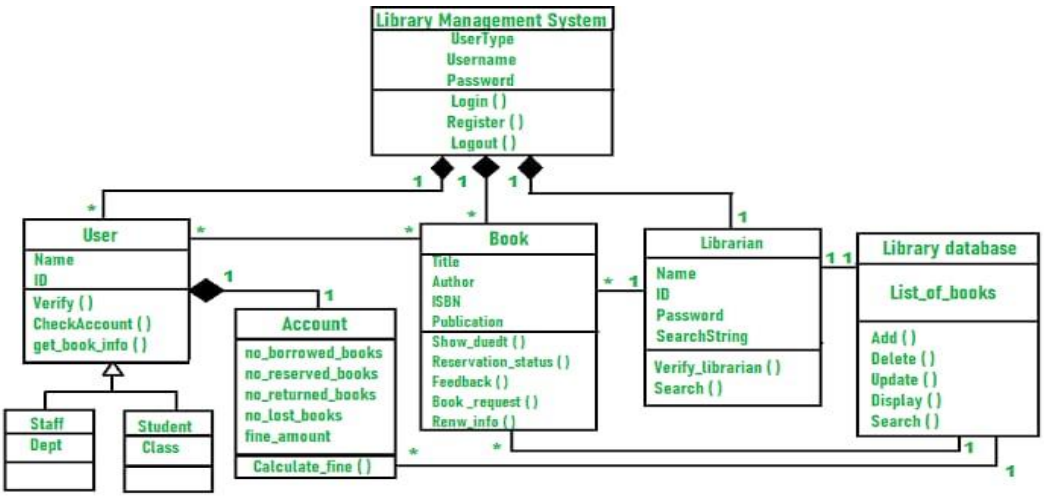
- Generate Reports: Enables administrators to generate various reports, such as overdue books, popular genres, and inventory status.



4. Class Diagrams:

Class diagrams depict the object-oriented design of the system, illustrating the classes, attributes, and methods involved in implementing system functionalities. Key classes include:

- Book: Represents a library book, encapsulating attributes such as title, author, genre, and availability status.
- Patron: Represents a library patron, encapsulating attributes such as name, contact information, and borrowing history.
- Transaction: Represents a library transaction, encapsulating details such as book ID, patron ID, transaction type, and timestamp.
- Administrator: Represents a system administrator, encapsulating credentials and access control methods.



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM

By adhering to this comprehensive system design, the advanced library database management system ensures scalability, maintainability, and extensibility while providing a robust platform for efficient library operations and user interactions.

CHAPTER 5

SYSTEM IMPLEMENTATION

CODE OVERVIEW

```
#include <bits/stdc++.h>
#include <iostream>
#include <mysql.h>
#include <mysqld_error.h>
#include <windows.h>
#include <sstream>
#include <cstdlib>

using namespace std;

const char* HOST = "localhost";
const char* USER = "root";
const char* PW = "Vibhor123#";
const char* DB = "Manage";

// Class definition for Student
class Student{
private:
 string Id;
 string Name;
 string No;
public:
Student() : Id(""), Name(""), No("") {}

// Setter functions for Student attributes
void setId(string id) {
  Id = id;
 }
 void setName(string name)
 {
         Name=name;
```

```
 }
 void setNo(string no)
 {
        No=no;
 }

 // Getter functions for Student attributes
 string getName()
 {
        return Name;
 }
 string getNo()
 {
        return No;
 }
 string getId() {
  return Id;
 }
};

// Class definition for Library
class Library{
private:
 string Name;
 int Quantity;
public:
Library() : Name(""), Quantity(0) {}

// Setter functions for Library attributes
void setName(string name) {
  Name = name;
}

void setQuantity(int quantity) {
  Quantity = quantity;
}

// Getter functions for Library attributes
int getQuantity() {
  return Quantity;
}
string getName() {
  return Name;
```

```
}
};

// Function for administration tasks
void admin(MYSQL* conn, Library l, Student s){
 bool closed = false;
 while(!closed){
 int choice;
cout << "1. Add Book." << endl;
 cout << "2. Add Student." << endl;
 cout << "0. Exit." << endl;
 cout << "Enter Choice: ";
 cin >> choice;

 if(choice==1){
  system("cls");
string name;
int quantity;

cout<<"Enter the name of the book: ";
cin>>name;
l.setName(name);

cout<<"Enter the Quantity of the book: ";
cin>>quantity;
l.setQuantity(quantity);

int Iq = l.getQuantity();
stringstream ss;
ss<<Iq;
string Sq = ss.str();

// Insert book into the database
string book = "INSERT INTO library (Name,Quantity) VALUES('"+l.getName()+"',
'"+Sq+"') ";
if(mysql_query(conn,book.c_str())){
 cout<<"Error: "<<mysql_error(conn)<<endl;
}
else{
 cout<<"Book is Inserted Successfully"<<endl;
}
 }
```

```
 else if(choice==2){
  system("cls");
string id;
string name;
string no;
 cout << "Enter the ID of Student: ";
 cin >> id;
 s.setId(id);
 cout<<"Enter the name of the Student";
 cin>>name;
 s.setName(name);
 cout<<"Enter the number of the Student: ";
 cin>>no;
 s.setNo(no);

 // Insert student into the database
  string st = "INSERT INTO student (Id,Student_Name,Student_Number) VALUES('"
+ s.getId() + "','"+s.getName()+"','"+s.getNo()+"')";
 if (mysql_query(conn, st.c_str())) {
 cout << "Error: " << mysql_error(conn) << endl;
 }
else {
  cout << "Student Inserted Successfully" << endl;
}
}
else if(choice ==0){
 closed = true;
 cout<<"System is closing"<<endl;
}
 }
Sleep(3000);
}

// Function to display available books
void display(MYSQL* conn){
 system("cls");
cout<<"Available Books"<<endl;
cout<<"***************"<<endl;

string disp= "SELECT * FROM library";
 if (mysql_query(conn, disp.c_str())) {
  cout << "Error: " << mysql_error(conn) << endl;
 }
```

```
 else{
  MYSQL_RES* res;
  res= mysql_store_result(conn);
  if(res){
   int num= mysql_num_fields(res);
  MYSQL_ROW row;
  while(row=mysql_fetch_row(res)){
   for(int i=0; i< num; i++){
    cout<<" "<<row[i];
   }
  cout<<endl;
  }
  mysql_free_result(res);
  }
 }
}

// Function to check if a book exists in the library
int book(MYSQL* conn, string Bname){
string exist = "SELECT Name, Quantity FROM library WHERE Name = '" + Bname
+ "'";
if (mysql_query(conn, exist.c_str())) {
 cout << "Error: " << mysql_error(conn) << endl;
 }
else{
MYSQL_RES* res;
res = mysql_store_result(conn);
if(res){
 int num = mysql_num_fields(res);
MYSQL_ROW row;
while(row=mysql_fetch_row(res)){
 for(int i=0; i< num; i++){
 if(Bname == row[i]){
 int quantity = atoi(row[i+1]);
 return quantity;
 }
 else{
  cout<<"Book Not Found."<<endl;
 }
 }
}
mysql_free_result(res);
}
```

```
}//else if exist
return 0;
Sleep(5000);
}

// Function for user actions
void user(MYSQL* conn, Library l, Student s){
 system("cls");
 display(conn);
string Sid;
cout<<"Enter Your ID: ";
cin>>Sid;

string com = "SELECT * FROM student WHERE Id = '"+Sid+"'";
 if (mysql_query(conn, com.c_str())) {
   cout << "Error: " << mysql_error(conn) << endl;
 }
 else{
MYSQL_RES* res;
res=mysql_store_result(conn);
if(res){
 int num= mysql_num_rows(res);
 if(num==1){
  cout<<"Student ID Found."<<endl;
 string Bname;
 cout<<"Enter Book Name: ";
 cin>>Bname;
 if(book(conn,Bname) > 0){
  int bookQuantity = book(conn,Bname)-1;
  stringstream ss;
  ss<<bookQuantity;
  string Sq = ss.str();

string upd ="UPDATE library SET Quantity ='"+Sq+"' WHERE Name = '"+Bname+"'
";
if(mysql_query(conn,upd.c_str())){
 cout<<"Error: "<<mysql_error(conn)<<endl;
}
else{
 cout<<"Book is available. Borrowing Book...."<<endl;
}
 }
```

```
 else{
  cout<<"Book is not Available."<<endl;
 }
 }
else if(num==0){
 cout<<"This Student is Not Registered."<<endl;
}
}
mysql_free_result(res);
 }
}

// Main function
int main() {
 Student s;
 Library l;

MYSQL* conn;
conn = mysql_init(NULL);

if(!mysql_real_connect(conn,HOST, USER, PW,DB,3306,NULL,0)){
 cout<<"Error: "<<mysql_error(conn)<<endl;
}
else{
 cout<<"Logged In!"<<endl;
}
Sleep(3000);
bool exit = false;
while(!exit){
 system("cls");
 int val;
cout << "Welcome To Library Management System" << endl;
cout << "************************************" << endl;
cout << "1. Administration." << endl;
cout << "2. User." << endl;
cout<< "0. Exit." << endl;
cout<<"Enter Choice: ";
cin>>val;

if(val==1){
 system("cls");
admin(conn,l,s);
}
```

```
else if(val==2){
user(conn,l,s);
Sleep(5000);
}
else if(val==0){
 exit= true;
 cout<<"Good Luck!"<<endl;
 Sleep(3000);
}

}
mysql_close(conn);
 return 0;
}
```

CODE FUNCTIONALITY

1. Header Files:
   - The code includes several header files such as `<iostream>`, `<mysql.h>`, and `<windows.h>`. These header files provide access to necessary libraries and functionalities for input-output operations, MySQL database connectivity, and Windows-specific functions like system commands.

2. Constants:
   - Constants like `HOST`, `USER`, `PW`, and `DB` store connection parameters for the MySQL database. These constants are used to establish a connection to the database throughout the code.

3. Class Definitions:
   - Two classes are defined: `Student` and `Library`. These classes encapsulate the properties and behaviors of student and book objects, respectively.
   - The `Student` class represents student objects with attributes like ID, name, and number. These attributes are accessed and modified using setter and getter functions.
   - Similarly, the `Library` class represents book objects with attributes like name and quantity. It also provides setter and getter functions for accessing and modifying these attributes.

4. Function Prototypes:

- Before the `main` function, prototypes or declarations for functions like `admin`, `display`, `book`, and `user` are provided. These prototypes inform the compiler about the functions' signatures before they are defined later in the code.

5. Admin Function:
- The `admin` function handles administrative tasks within the library management system. It serves as the interface for administrators to add new books or register new students.
- This function presents a menu with options to add a book, add a student, or exit the system. The user's choice determines the subsequent actions.
- When adding a book, the function prompts the administrator to enter the book's name and quantity. It then inserts this information into the database table named "library" using SQL queries.
- Similarly, when adding a student, the function prompts for the student's ID, name, and number. It inserts this information into the "student" table in the database.

6. Display Function:
- The `display` function retrieves and displays the list of available books from the database. It provides users with visibility into the books currently stocked in the library.
- This function executes a SQL query to select all records from the "library" table, retrieves the result set, and prints the details of each book to the console.

7. Book Function:
- The `book` function checks if a given book exists in the library by querying the database.
- It takes the name of the book as input and executes a SQL query to search for a matching record in the "library" table.
- If the book is found, the function returns the quantity of available copies. Otherwise, it returns 0, indicating that the book is not in the library's inventory.

8. User Function:
- The `user` function handles user actions such as borrowing books from the library.
- It prompts the user to enter their ID and checks if the ID exists in the "student" table of the database. If the user is registered, they can proceed to borrow a book.
- After entering the book's name, the function checks its availability using the `book` function. If the book is available, it decrements the quantity of the book in the database and notifies the user that the book has been borrowed successfully.

9. Main Function:
- The `main` function serves as the entry point of the program. It orchestrates the execution flow by presenting options to users, processing their choices, and performing corresponding actions.

- This function initializes the MySQL connection, displays a welcome message, and provides a menu for users to choose between administrative tasks, user actions, or exiting the system.
- Once the user chooses to exit, the `main` function closes the MySQL connection and terminates the program.

10. MySQL Connection:

(a).mysql_init:
- This function initializes the MySQL library environment and allocates necessary resources. It returns a pointer to a MySQL object, which is then used to establish a connection to the MySQL database server.

(b). mysql_real_connect:
- The `mysql_real_connect` function establishes a connection to the MySQL database server using the provided connection parameters.
- It takes parameters such as the MySQL connection object (`MYSQL* conn`), hostname (`HOST`), username (`USER`), password (`PW`), database name (`DB`), and port number (`3306` by default).
- If the connection is successful, it returns a non-null value, indicating a successful connection. Otherwise, it returns null, indicating connection failure.

(c). mysql_query:
- The `mysql_query` function is used to execute SQL queries on the connected MySQL database.
- It takes two parameters: the MySQL connection object (`conn`) and the SQL query string (`query`).
- This function sends the SQL query to the MySQL database server for execution. If the query is executed successfully, it returns 0. Otherwise, it returns a non-zero value indicating an error.

(d). mysql_store_result:
- The `mysql_store_result` function retrieves the result set (if any) from the most recently executed SQL query and stores it in a buffer allocated by the MySQL client library.
- It takes a single parameter, the MySQL connection object (`conn`), and returns a pointer to a MySQL result object (`MYSQL_RES*`).
- This function is typically used after executing SELECT queries to retrieve the rows returned by the query for further processing.

(e). mysql_num_rows:
- The `mysql_num_rows` function returns the number of rows in the result set obtained from a SELECT query.

- It takes a single parameter, a pointer to the MySQL result object (`MYSQL_RES*`), and returns an integer representing the number of rows in the result set.

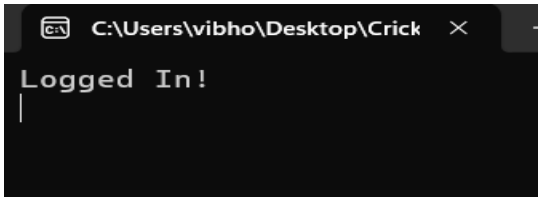
(f). mysql_fetch_row**:
- The `mysql_fetch_row` function retrieves the next row from the result set returned by a SELECT query and stores it in an array of strings.
- It takes a single parameter, a pointer to the MySQL result object (`MYSQL_RES*`), and returns an array of strings (`MYSQL_ROW`) representing the values in the current row.
- This function is typically used in a loop to iterate over all rows in the result set and process each row's data.

(g). mysql_free_result:
- The `mysql_free_result` function frees the memory allocated for a MySQL result object after it has been processed.
- It takes a single parameter, a pointer to the MySQL result object (`MYSQL_RES*`), and releases the memory associated with it.
- This function should be called after fetching and processing all rows from the result set to avoid memory leaks.

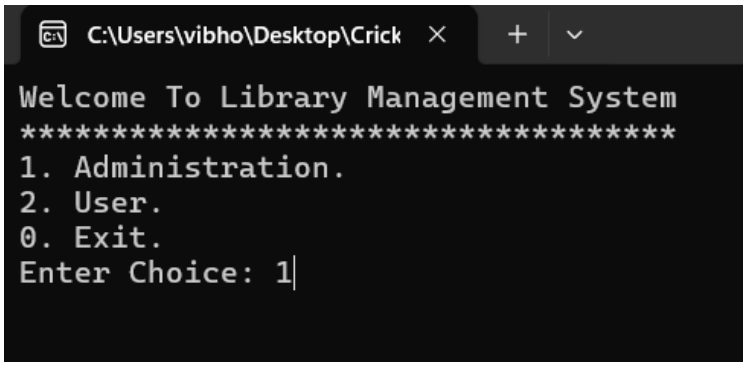
11. Error Handling:
- Error handling is implemented throughout the code using `mysql_error` to display descriptive error messages in case of database-related errors. This ensures that users are informed about any issues encountered during database operations.

OUTPUT DEMONSTRATION

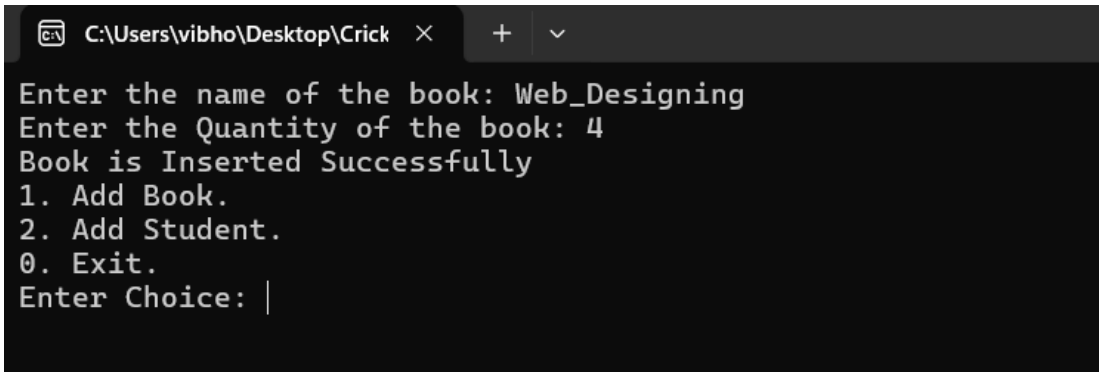
1) Login into the E-Library Management System:



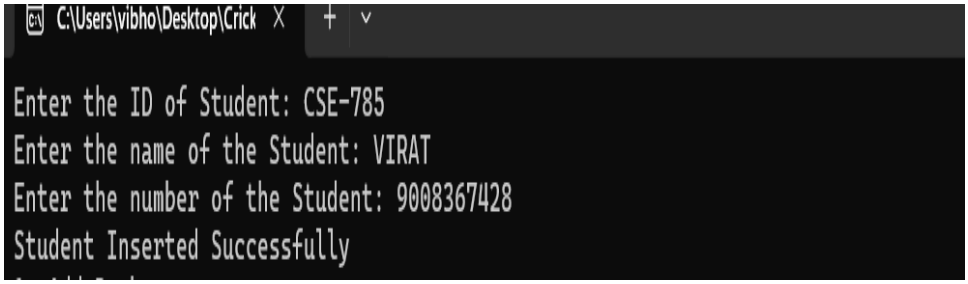
2) Choice of User and Administration Manipulate and Fetch the Database:



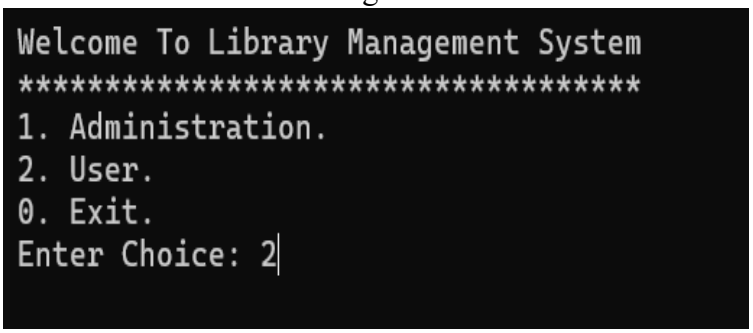
3) Administrator can Insert the Data related to Books and Student in SQL Database :



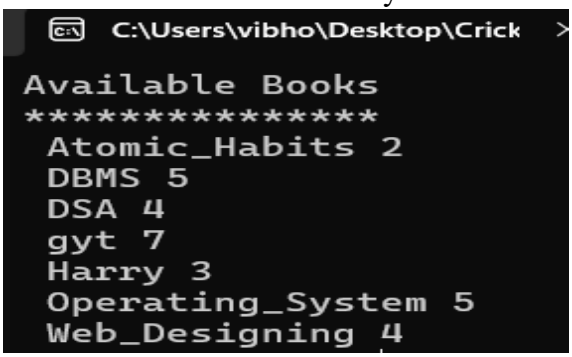
4) This Shows the Adding of Student data in SQL Database:



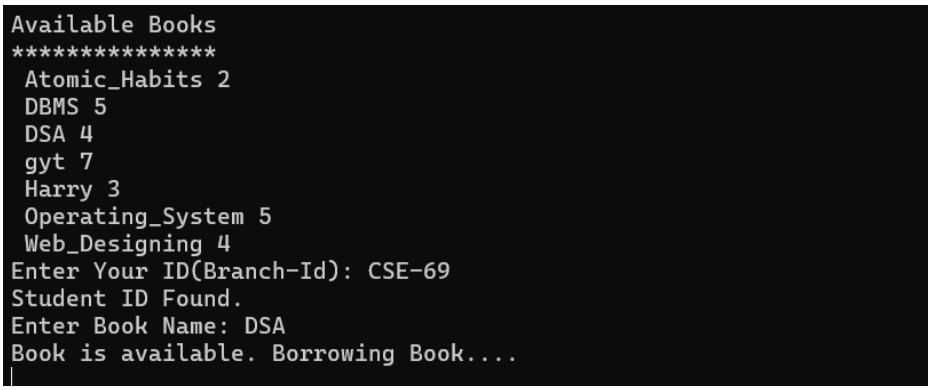
5) This Shows the Choice being 2 i.e. User End:



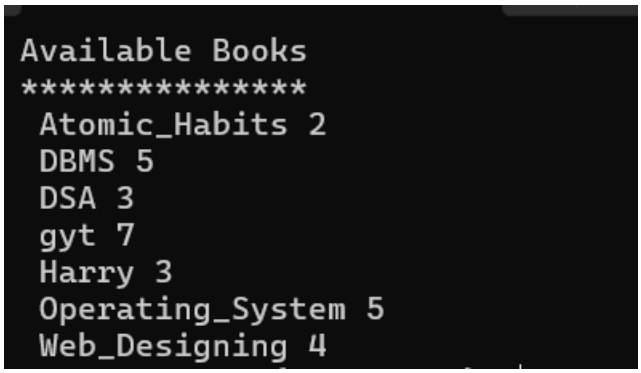
6) Available Books in the Library Database will be Shown to User:



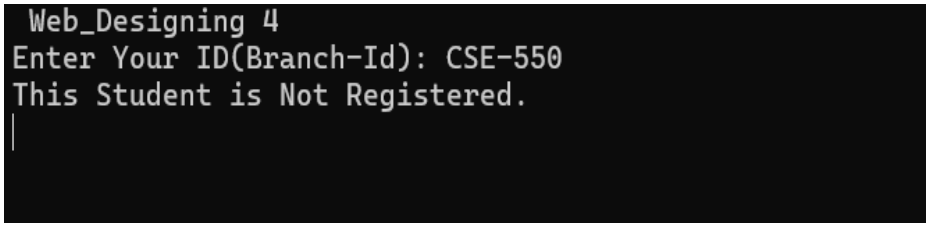
7) If Book is Available then User can Borrow it by Entering book name and Student ID and When book is borrowed, the quantity of book will be reduced by one in the Library Database:



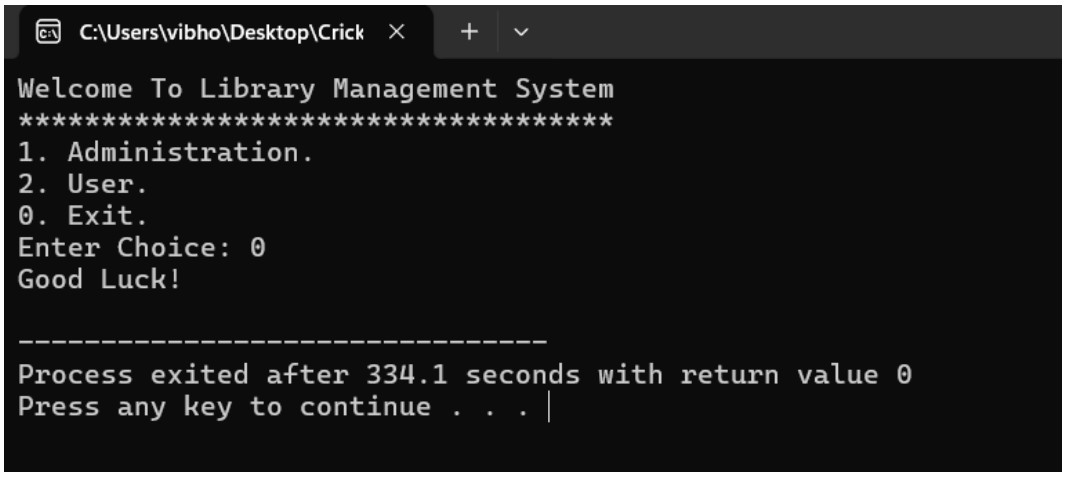
8) User Borrowed DSA book hence it got Reduced from 4 to 3 in the Database:



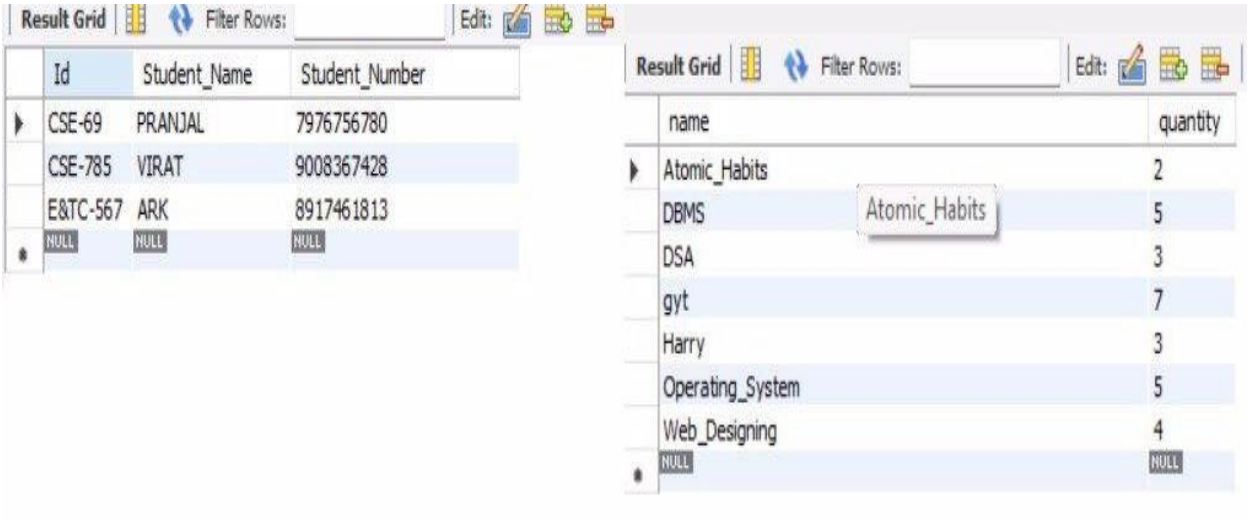
9) Student of ID CSE-550 is not present in Student Database as they did’nt registered therefore they can’t borrow the book from Library:



10) Entering 0 will Terminate the C++ code and Exit the Library Management System:



11) Entering the Data on C++ code will get Reflected in the SQL Database and Values get Stored in it. As we can see the two Tables of Student and Books are Generated as we Give Input in Admistration Section and if User Borrow the Book then Quantity will also get Affected in the SQL Database Due to Integration of C++ and SQL:



CHAPTER 6

CONCLUSION

In conclusion, the integration of C++ programming with SQL databases has yielded a comprehensive and efficient library database management system that addresses the diverse needs of users and administrators. Through systematic analysis, meticulous design, and rigorous implementation, our project has culminated in a solution that not only streamlines library operations but also lays the groundwork for future innovations in information management systems.

One of the key achievements of our project is the seamless integration of C++ and SQL, enabling intuitive user interactions and efficient database operations. By leveraging the strengths of C++ for application logic and SQL for data management, we have created a system that exhibits robustness, scalability, and maintainability.
Moreover, the design of our system, encompassing database schema design, system architecture, and security considerations, reflects a commitment to best practices and standards in software engineering. Through the adoption of modular design principles and adherence to established methodologies, we have ensured the reliability and extensibility of our system.

Looking ahead, there are several avenues for future enhancement and exploration. Continuous refinement of the user interface, integration with external systems, and adoption of emerging technologies such as NoSQL databases and machine learning algorithms present exciting opportunities for further innovation and optimization.

In essence, our project represents a significant step forward in the realm of library database management, showcasing the transformative potential of integrating C++ and SQL technologies. As libraries continue to evolve in response to technological advancements and changing user needs, our system stands ready to adapt and thrive in the digital age, empowering libraries to remain vibrant hubs of knowledge dissemination and exploration.