TABLE OF CONTENTS

1.CORRELATED QUERIES

1.1 INTRODUCTION

Correlated subqueries, often referred to as correlated queries, are a type of subquery in SQL where the subquery references one or more columns from the outer query. In other words, the subquery is dependent on the outer query for its execution.The basic structure of a correlated subquery involves using a column from the outer query in the WHERE clause of the subquery. This creates a relationship between the inner and outer queries. The subquery is executed once for each row processed by the outer query, and the result is used in conjunction with the outer query to produce the final result.

Correlated subqueries are particularly useful when you need to perform comparisons or filtering based on data from related tables. However, they can be less efficient than non-correlated subqueries, and their performance may depend on the complexity of the subquery and the underlying database engine's optimization capabilities.

In correlated queries, the subquery's execution is dependent on the data retrieved by the outer query. The result of the outer query influences the execution of the subquery. Correlation can involve multiple columns. For example, a correlated subquery might compare two or more columns from the outer query with columns in the subquery. Correlated subqueries can use aggregate functions. For instance, you might use a correlated subquery to find records where a certain aggregate condition is met for related rows.

Due to the dependency on the outer query, correlated queries can have a higher performance impact compared to non-correlated queries. The database engine may need to execute subquery multiple times

1.2 IMPLEMENTATION AND RESULTS

```
mysql> SELECT fine_id,issue_id,member_id
    -> FROM Fine
    -> WHERE fine_amount>(SELECT avg(fine_amount) FROM Fine);
+---------+----------+-----------+
| fine_id | issue_id | member_id |
+---------+----------+-----------+
|     504 |     1004 | STU005    |
+---------+----------+-----------+
1 row in set (0.00 sec)
```

2. VIEWS

2.1 INTRODUCTION

A view in a relational database is a virtual table based on the result of a SELECT query. It doesn't store the data itself but provides a way to represent the data stored in one or more tables. A view is a saved SQL query that behaves like a table. It represents a set of rows and columns derived from one or more underlying tables.Views provide a layer of abstraction, allowing users to interact with a simplified representation of the data without needing to understand the underlying table structures or complex joins.Views can restrict access to specific columns, ensuring that users only see the data they are authorized to access.

Views can simplify complex queries by encapsulating joins, calculations, or filters into a single virtual table.Views are often used to present a subset of data or aggregated information to users or applications.They can be employed to join tables in a way that is more convenient for users or applications.Views can encapsulate business logic, providing a consistent and abstracted interface to applications.In some cases, data can be updated through a view if certain conditions are met. However, there are restrictions, and not all views are updatable.

Some views may be read-only, especially those involving multiple tables, certain SQL constructs, or aggregations.In some database systems, there are materialized views that store the result set, allowing for faster query performance at the cost of data freshness.Views can provide a layer of abstraction that makes applications less dependent on the underlying database structure, promoting portability.Views are a powerful feature in relational databases, offering flexibility, security, and a simplified way to interact with complex data structures. They play a crucial role in database design, query optimization, and data access control.

2.1 IMPLEMENTATION AND RESULTS

```
mysql> CREATE view LoginMembersOnAday AS
    -> SELECT COUNT(*) AS num_users
    -> FROM Login
    -> WHERE t_date = '2023-10-24';
Query OK, 0 rows affected (0.02 sec)

mysql> select * from LoginMembersOnAday;
+-----------+
| num_users |
+-----------+
|         4 |
+-----------+
1 row in set (0.02 sec)
```

```
mysql> CREATE view FINEPAIDMEMBERS AS
    -> SELECT Members.first_name, Members.last_name, Fine.fine_amount, Fine.fine_description
    -> FROM Fine
    -> INNER JOIN Members ON Fine.Member_id = Members.Member_id
    -> WHERE Fine.fine_paid = 0;
Query OK, 0 rows affected (0.07 sec)

mysql> select * from FINEPAIDMEMBERS;
+------------+-----------+-------------+-----------------+
| first_name | last_name | fine_amount | fine_description |
+------------+-----------+-------------+-----------------+
| ANAGHA     | NULL      |          40 | Book damage      |
| KIRTANA    | NULL      |          40 | Return delay     |
| Abhishek   | M B       |         800 | Book lost        |
+------------+-----------+-------------+-----------------+
3 rows in set (0.00 sec)
```

3. EXSISTS, NOT EXIST

3.1 INTRODUCTION

The **EXISTS** operator in SQL is a logical operator used to determine the existence of rows in the result set of a subquery. It returns a Boolean value, either TRUE or FALSE, indicating whether the subquery returns any rows.EXISTS is commonly used in the WHERE clause of a SQL query to filter results based on the presence or absence of rows in a correlated subquery.

The subquery used with EXISTS is often correlated, meaning it refers to columns from the outer query. This correlation allows the subquery to be evaluated for each row in the outer query.If the subquery returns at least one row, EXISTS evaluates to TRUE; otherwise, it evaluates to FALSE.EXISTS can be more efficient than other means of achieving the same result, especially when you only need to check for the existence of rows rather than retrieving specific column values.The subquery doesn't need to return unique rows; the operator checks only for the existence of any rows.The structure of the subquery within EXISTS can vary based on the specific condition you are checking.EXISTS is a valuable tool in SQL for writing queries that involve conditional filtering based on the existence of rows in related tables. It provides a concise and efficient way to express these conditions in SQL statements.

The **NOT EXISTS** operator in SQL is a logical operator used to negate the existence of rows in the result set of a subquery. It is the complement of the EXISTS operator, and it returns a Boolean value, either TRUE or FALSE, indicating whether the subquery returns no rows.Similar to EXISTS, NOT EXISTS is commonly used in the WHERE clause of a SQL query to filter results based on the absence of rows in a correlated subquery.The subquery used with NOT EXISTS is often correlated, meaning it refers to columns from the outer query. This

correlation allows the subquery to be evaluated for each row in the outer query.If the subquery returns no rows, NOT EXISTS evaluates to TRUE; if the subquery returns at least one row, it evaluates to FALSE.NOT EXISTS can be more efficient than other means of achieving the same result, especially when you only need to check for the absence of rows rather than retrieving specific column values.Similar to EXISTS, the subquery within NOT EXISTS doesn't need to return unique rows; the operator checks only for the absence of any rows.The structure of the subquery within NOT EXISTS can vary based on the specific condition you are checking.NOT EXISTS is a valuable tool in SQL for expressing conditions where you want to filter results based on the absence of rows in related tables. It provides a concise and efficient way to handle these scenarios in SQL statements.

3.2 IMPLEMENTATION AND RESULTS

```
mysql> select * from issue where exists (
    -> select 1 from Fine where issue.issue_id=fine.issue_id );
+----------+------------+------------+-------------+
| issue_id | issue_date | due_date   | return_date |
+----------+------------+------------+-------------+
|     1000 | 2023-10-01 | 2023-10-16 | 2023-10-20  |
|     1001 | 2023-10-02 | 2023-10-17 | 2023-10-21  |
|     1002 | 2023-10-02 | 2023-10-17 | 2023-10-22  |
|     1003 | 2023-10-03 | 2023-10-18 | 2023-10-22  |
|     1004 | 2023-10-03 | 2023-10-18 | 2023-10-21  |
+----------+------------+------------+-------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT  member_id,first_name as name
    -> FROM members
    -> WHERE EXISTS (SELECT * FROM borrows where borrows.member_id= members.member_id)
    -> AND
    -> EXISTS (SELECT * FROM fine where Fine.member_id= members.member_id);
+-----------+----------+
| member_id | name     |
+-----------+----------+
| STU001    | Abhishek |
| STU002    | Anagha   |
| STU003    | Sinchana |
| STU004    | Kirtana  |
| STU005    | Aniket   |
+-----------+----------+
5 rows in set (0.01 sec)
```

```
mysql> SELECT Member_id,first_name as name
    -> FROM Members
    -> WHERE NOT EXISTS(SELECT *FROM Borrows WHERE Members.Member_id = Borrows.Member_id);
+----------+---------+
| Member_id | name    |
+----------+---------+
| STU009    | Jagruth |
+----------+---------+
1 row in set (0.00 sec)
```

```
mysql> SELECT fine_description,SUM(fine_amount) AS total_amount
    -> FROM fine
    -> WHERE fine_date!='2023-10-17'
    -> GROUP BY fine_description
    -> HAVING SUM(fine_amount)>50
    -> ORDER BY total_amount DESC;
+------------------+--------------+
| fine_description | total_amount |
+------------------+--------------+
| Book lost        |          800 |
| Book damage      |           90 |
| Return delay     |           70 |
+------------------+--------------+
3 rows in set (0.02 sec)
```

```
mysql> SELECT Member_id,first_name,last_name
    -> FROM Members
    -> WHERE NOT EXISTS ((SELECT member_id FROM Access WHERE Members.member_id = Access.Member_id)
    -> EXCEPT (SELECT Member_id FROM Members WHERE first_name = "john"));
+----------+-----------+----------+
| Member_id | first_name | last_name |
+----------+-----------+----------+
| STU006    | John      | Samuel   |
| STU009    | Jagruth   | Kumar    |
+----------+-----------+----------+
2 rows in set (0.01 sec)
```

4. AGGREGATE FUNCTIONS

4.1 INTRODUCTION

Aggregate functions in SQL are essential tools for performing calculations on sets of data. These functions operate on a group of rows and return a single, summarized result. Here's a comprehensive overview of aggregate functions in SQL:

Common Aggregate Functions:

1.SUM: Calculates the total sum of a numeric column.

2.AVG (Average): Computes the average value of a numeric column.

3.COUNT: Counts the number of rows in a result set. COUNT(*) counts all rows, while COUNT(column) counts non-null values in a specific column.

4.MIN (Minimum): Finds the minimum value in a numeric or alphanumeric column.

5.MAX (Maximum): Finds the maximum value in a numeric or alphanumeric column.

4.2 IMPLEMENTATION AND RESULTS

```
mysql> SELECT fine_description,SUM(fine_amount) AS total_amount
    -> FROM fine
    -> WHERE fine_date!='2023-10-17'
    -> GROUP BY fine_description
    -> HAVING SUM(fine_amount)>50
    -> ORDER BY total_amount DESC;
+------------------+--------------+
| fine_description | total_amount |
+------------------+--------------+
| Book lost        |          800 |
| Book damage      |           90 |
| Return delay     |           70 |
+------------------+--------------+
3 rows in set (0.02 sec)
```

```
mysql> SELECT eBook_title AS MAX_rating
    -> FROM Ebooks
    -> WHERE ratings_value = (SELECT MAX(ratings_value) FROM Ebooks);
+----------------------------------------+
| MAX_rating                             |
+----------------------------------------+
| FUNDAMENTALS OF DATABASE SYSTEMS       |
| MACHINE LEARNING                       |
| MECHATRONICS                           |
| DESIGNING DATA-INTENSIVE APPLICATIONS  |
| ELECTRONICS FUNDAMENTALS AND APPLICATIONS |
| PRACTICAL ELECTRONICS FOR INVENTORS    |
+----------------------------------------+
6 rows in set (0.03 sec)
```

```
mysql> SELECT Fine_id,issue_id,Member_id FROM fine
    -> where fine_amount<(
    -> select avg(fine_amount) from fine);
+---------+----------+-----------+
| Fine_id | issue_id | Member_id |
+---------+----------+-----------+
|     500 |     1000 | STU001    |
|     501 |     1001 | STU002    |
|     502 |     1002 | STU003    |
|     503 |     1003 | STU004    |
+---------+----------+-----------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT member_id,COUNT(*) AS no_of_books_taken
    -> FROM borrows
    -> GROUP BY member_id
    -> ORDER BY no_of_books_taken;
+-----------+-------------------+
| member_id | no_of_books_taken |
+-----------+-------------------+
| STU002    |                 1 |
| STU003    |                 1 |
| STU004    |                 1 |
| STU005    |                 1 |
+-----------+-------------------+
4 rows in set (0.02 sec)
```

5. TRIGGERS

5.1 INTRODUCTION

Triggers in SQL are special types of stored procedures that are automatically executed (or "triggered") in response to specific events occurring in the database. These events can include data modifications, such as INSERT, UPDATE, DELETE operations, and other database-related activities.Triggers are event-driven, meaning they are executed automatically when a predefined event occurs in the database. There are two main types of triggers: BEFORE triggers and AFTER triggers.

BEFORE Triggers: Executed before the event (e.g., before an INSERT or UPDATE operation).

AFTER Triggers: Executed after the event.

Triggers can be associated with various database events, such as INSERT, UPDATE, DELETE, or even database schema changes (e.g., CREATE, ALTER).

Triggers are commonly used to enforce business rules, validate data, maintain referential integrity, and automate tasks based on database events.Triggers are automatically invoked by the database engine when the associated event occurs, providing a seamless and automated way to respond to changes in the database.Triggers operate within the scope of the transaction that fires them, allowing them to access and modify data within the same transaction. This ensures data consistency and integrity.Triggers can include transaction control statements (e.g., ROLLBACK) to manage the outcome of the triggering operation.Some database systems support recursive triggers, where a trigger execution results in another trigger being fired.While triggers offer powerful capabilities, they should be used judiciously to avoid performance issues, especially in scenarios involving complex logic or frequent data modifications.Triggers in SQL provide a mechanism for automating actions in response to specified database events. Their use is valuable for maintaining data integrity, enforcing business rules, and streamlining database management tasks. However, proper design and consideration of potential performance impacts are essential when implementing triggers.

Triggers execute with the permissions of the user who performed the triggering action. Care must be taken to ensure that the executing user has the necessary permissions.

5.2 IMPLEMENTATION AND RESULTS

```
mysql> CREATE TRIGGER update_book_count
    ->  AFTER INSERT ON books
    ->  FOR EACH ROW
    ->  UPDATE Department
    ->  SET total_books=total_books+1
    ->  where department.subject_id=(
    ->  SELECT categorised FROM books where book_id=NEW.book_id);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO books (Book_id,Book_title,Book_author,Book_edition,categorised)
    -> VALUES
    -> (17,"DATABASE MANAGEMENT SYSTEMS","ELMASRI",6,"CS001");
Query OK, 1 row affected (0.01 sec)

mysql> select * from department;
+------------+-----------------------------------------------+---------------+-------+-------------+----------+
| subject_id | Sname                                         | Department_id | Dname | total_books | shelf_no |
+------------+-----------------------------------------------+---------------+-------+-------------+----------+
| CS001      | DATABASE MANAGEMENT SYSTEMS                   |             1 | CSE   |           2 |        1 |
| CS009      | MACHINE LEARNING                              |             1 | CSE   |           1 |        3 |
| CS112      | THEORY OF COMPUTATION                         |             1 | CSE   |           1 |        2 |
| CV111      | CIVIL ENGINEERING:CONVENTIONAL AND OBJECTIVE  |             4 | CIVIL |           1 |        8 |
| CV231      | ESTIMATING AND COSTING IN CIVIL ENGINEERING   |             4 | CIVIL |           1 |        9 |
| EC118      | ELECTRONICS FUNDAMENTALS AND APPLICATIONS     |             7 | ECE   |           1 |       14 |
| EC222      | PRACTICAL ELECTRONICS FOR INVENTORS           |             7 | ECE   |           1 |       15 |
| EE108      | ELECTRICAL ENGINEERING FUNDAMENTALS           |             3 | EEE   |           1 |        6 |
| EE112      | SIGNAL AND SYSTEMS                            |             3 | EEE   |           1 |        7 |
| IS109      | INFORMATION TECHNOLOGY                        |             5 | ISE   |           1 |       10 |
| IS222      | DESIGNING DATA-INTENSIVE APPLICATIONS         |             5 | ISE   |           1 |       11 |
| ME005      | BASIC MECHANICAL ENGINEERING                  |             2 | ME    |           1 |        4 |
| ME101      | MECHATRONICS                                  |             2 | ME    |           1 |        5 |
| PS004      | POLYMER SCIENCE AND TECHNOLOGY                |             6 | PSE   |           1 |       12 |
| PS013      | POLYMERIZATION                                |             6 | PSE   |           1 |       13 |
+------------+-----------------------------------------------+---------------+-------+-------------+----------+
15 rows in set (0.00 sec)
```

```
mysql>  DELIMITER //
mysql> CREATE TRIGGER UpdateFineAmountOnLateReturn
    -> AFTER UPDATE ON Issue
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE fine_amount INT;
    ->     SET fine_amount = DATEDIFF(NEW.return_date, NEW.due_date) * 10;
    ->  UPDATE Fine
    ->     SET fine_amount = fine_amount
    ->     WHERE issue_id = NEW.issue_id;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
mysql> UPDATE issue
    -> SET return_date="2023-10-25"
    -> WHERE issue_id=1003;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from fine;
+---------+-----------+----------+------------+-------------+------------------+-----------+
| Fine_id | Member_id | issue_id | fine_date  | fine_amount | fine_description | fine_paid |
+---------+-----------+----------+------------+-------------+------------------+-----------+
|     500 | STU001    |     1000 | 2023-10-17 |          90 | Return delay     |         1 |
|     501 | STU002    |     1001 | 2023-10-18 |          40 | Book damage      |         0 |
|     502 | STU003    |     1002 | 2023-10-18 |          50 | Book damage      |         1 |
|     503 | STU004    |     1003 | 2023-10-19 |          70 | Return delay     |         0 |
|     504 | STU005    |     1004 | 2023-10-19 |         800 | Book lost        |         0 |
+---------+-----------+----------+------------+-------------+------------------+-----------+
5 rows in set (0.00 sec)
```

6. STORED PROCEDURES

6.1 INTRODUCTION

Stored procedures in SQL are precompiled SQL statements that are stored in the database and can be executed by calling the procedure name.Stored procedures are precompiled SQL statements stored in the database. This precompilation can result in performance benefits as the SQL code is compiled only once.Stored procedures promote code reusability and modularity. Once created, they can be called from multiple parts of an application or even by different applications.Stored procedures can accept parameters, allowing for dynamic customization of their behavior based on the values passed during execution.Stored procedures can help enhance security by allowing controlled access to the database. Users can be granted permission to execute a stored procedure without direct access to underlying tables.Calling a stored procedure involves sending a single command to the database server, reducing the amount of data transferred over the network compared to executing multiple SQL statements individually.Stored procedures can be used to encapsulate multiple SQL statements within a single transaction, ensuring atomicity (all-or-nothing execution).The precompiled nature of stored procedures can lead to improved performance as the execution plan is cached, and subsequent calls can benefit from the cached plan.Business logic can be encapsulated in stored procedures, making it easier to maintain and update the logic centrally.Stored procedures provide a consistent way to execute complex database operations. This consistency can be crucial for maintaining data integrity and enforcing business rules.By centralizing logic in stored procedures, code duplication is minimized, making it easier to manage and update business rules.

Stored procedures can include error-handling mechanisms to gracefully handle unexpected situations and provide meaningful error messages.Changes to stored procedures can be version-controlled, enabling better management of database schemachanges over time.Stored procedures provide a layer of abstraction between the database and application code, allowing changes to the database without requiring modifications to application code.In summary, stored procedures in SQL offer a powerful mechanism for organizing and executing SQL code within a database. Their use can result in improved performance, enhanced security, and better maintainability of database-related operations.

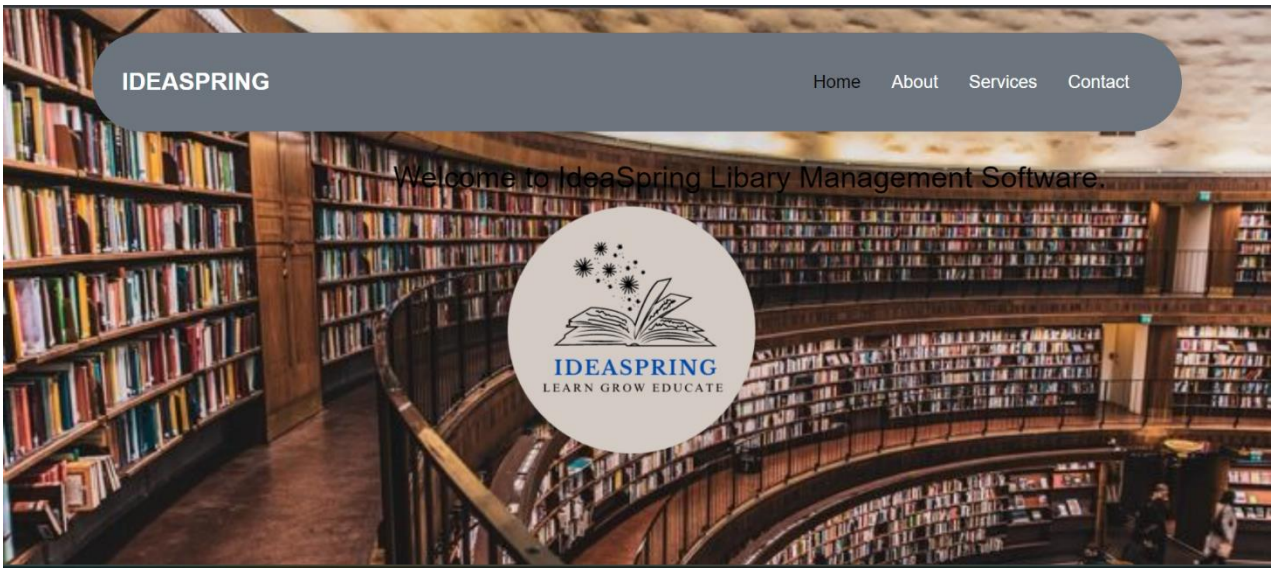
```
mysql> select * from members;
+-----------+------------+-----------+--------------------+------------+-----------------------------------------------------------------+--------------+----------+-
------------+
| Member_id | first_name | last_name | email              | phone_no   | address                                                         | next_renewal | login_id |
registeredby |
+-----------+------------+-----------+--------------------+------------+-----------------------------------------------------------------+--------------+----------+-
------------+
| STU001    | Aniket     | Shetty    | aniket@gmail.com   | 9019865623 | address                                                         | 2025-12-02   |        1 |
LIBSL013     |
| STU002    | ANAGHA     | NULL      | anagha@gmail.com   | 1234567890 | #16,4th cross,Siddhartha layout,Mysuru                          | 2025-01-21   |        2 |
LIDTE003     |
| STU003    | SINCHANA   | NULL      | sinchana@gmail.com | 2345678127 | #20,3rd cross,gangotri layout                                   | 2025-01-22   |        3 |
LIDTE003     |
| STU004    | KIRTANA    | NULL      | kirtana@gmail.com  | 8893452134 | #21,4th cross,rajkamal apartment,Rajajinagar,Bangalore          | 2025-01-22   |        4 |
LIDLI002     |
| STU005    | Abhishek   | M B       | aniket@gmail.com   | 9019865623 | mysore                                                          | 2025-12-02   |        5 |
LIBSL013     |
| STU006    | john       | samuel    | john@gmail.com     | 6478576879 | vijayanagar                                                     | 2025-10-23   |        6 |
LIDLI004     |
| STU007    | Jagruth    | Kumar     | jagruth@gmail.com  | 6478577779 | Kuvempunagar                                                    | 2025-11-03   |        7 |
LIDLI004     |
+-----------+------------+-----------+--------------------+------------+-----------------------------------------------------------------+--------------+----------+-
------------+
7 rows in set (0.00 sec)
```

```
mysql> use library_database;
Database changed
mysql> DELIMITER //
mysql> CREATE PROCEDURE AddMember(
    ->     IN p_Member_id VARCHAR(10),
    ->     IN p_first_name VARCHAR(20),
    ->     IN p_last_name VARCHAR(20),
    ->     IN p_email VARCHAR(20),
    ->     IN p_phone_no BIGINT,
    ->     IN p_address TEXT,
    ->     IN p_next_renewal DATE,
    ->     IN p_login_id INT,
    ->     IN p_registeredby VARCHAR(10)
    -> )
    -> BEGIN
    ->     INSERT INTO Members (Member_id, first_name, last_name, email, phone_no, address, next_renewal, login_id, registeredby)
    ->     VALUES (p_Member_id, p_first_name, p_last_name, p_email, p_phone_no, p_address, p_next_renewal, p_login_id, p_registeredby);
    -> END //
Query OK, 0 rows affected (0.04 sec)

mysql> DELIMITER ;
mysql> CALL AddMember ('STU007','Jagruth','Kumar','jagruth@gmail.com',6478577779,'Kuvempunagar','2025-11-03',7,'LIDLI004');
Query OK, 1 row affected (0.03 sec)
```

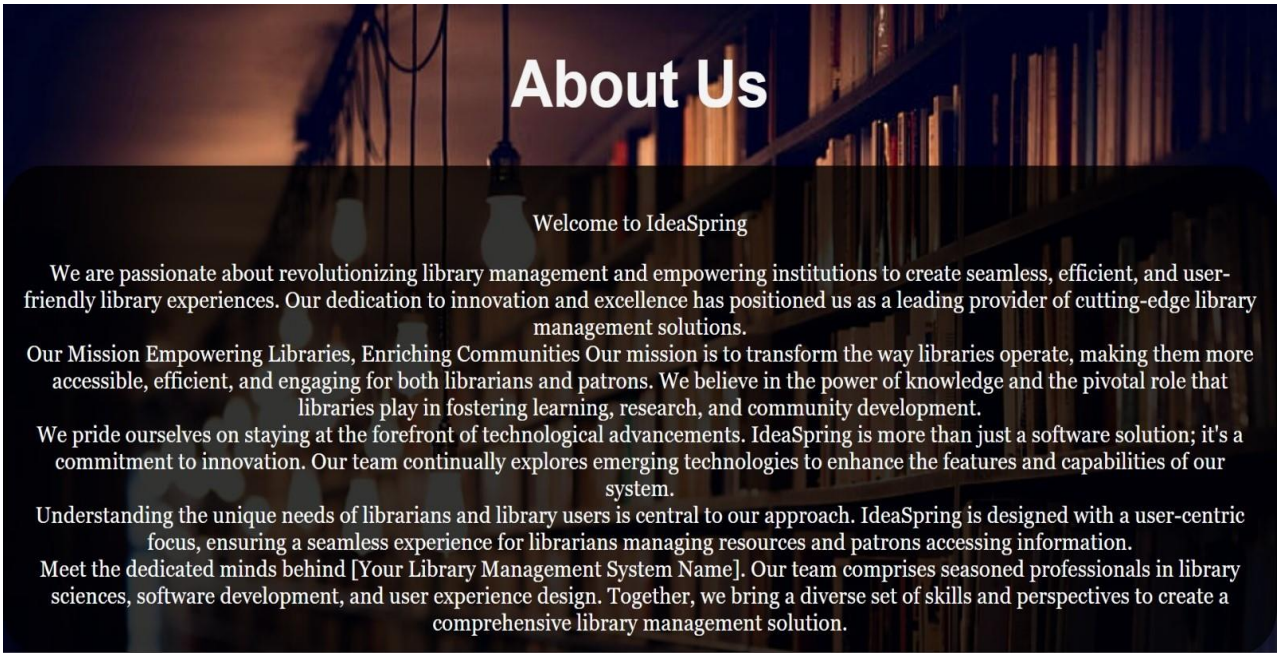
```
mysql> DELIMITER //
mysql> CREATE PROCEDURE CalculateTotalFineForAllMembers(OUT total_fine DECIMAL(10,2))
    -> BEGIN
    ->     SELECT SUM(fine_amount) INTO total_fine
    ->     FROM Fine
    ->     WHERE fine_paid = 0;
    -> END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL CalculateTotalFineForAllMembers(@total_fine);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT @total_fine AS TotalFineForAllMembers;
+-------------------------+
| TotalFineForAllMembers |
+-------------------------+
|                  910.00 |
+-------------------------+
1 row in set (0.00 sec)
```

7. FRONT END DESIGN

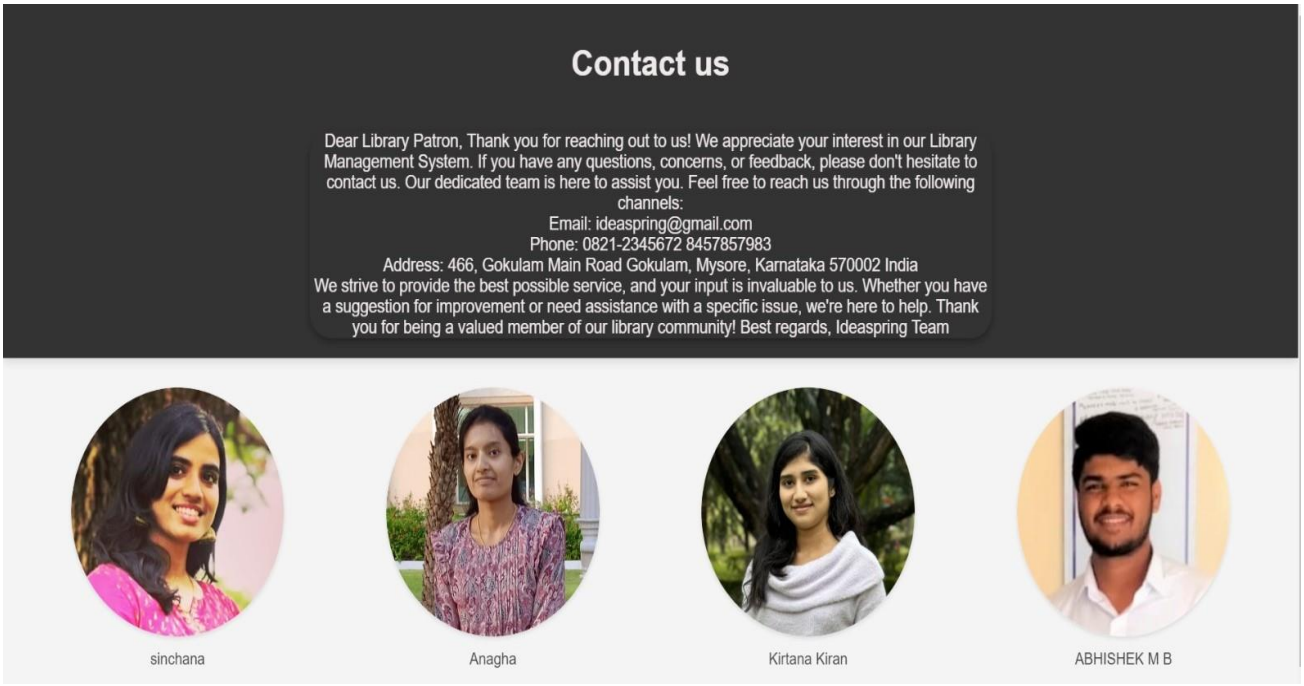
HOME PAGE



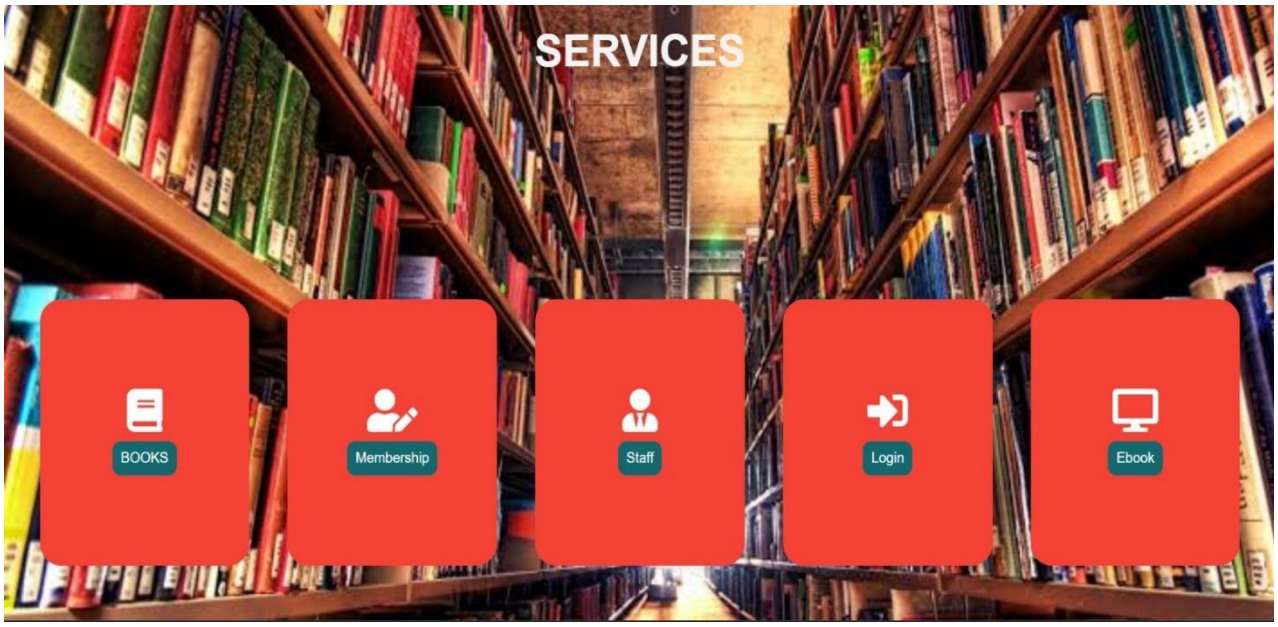
ABOUT US PAGE



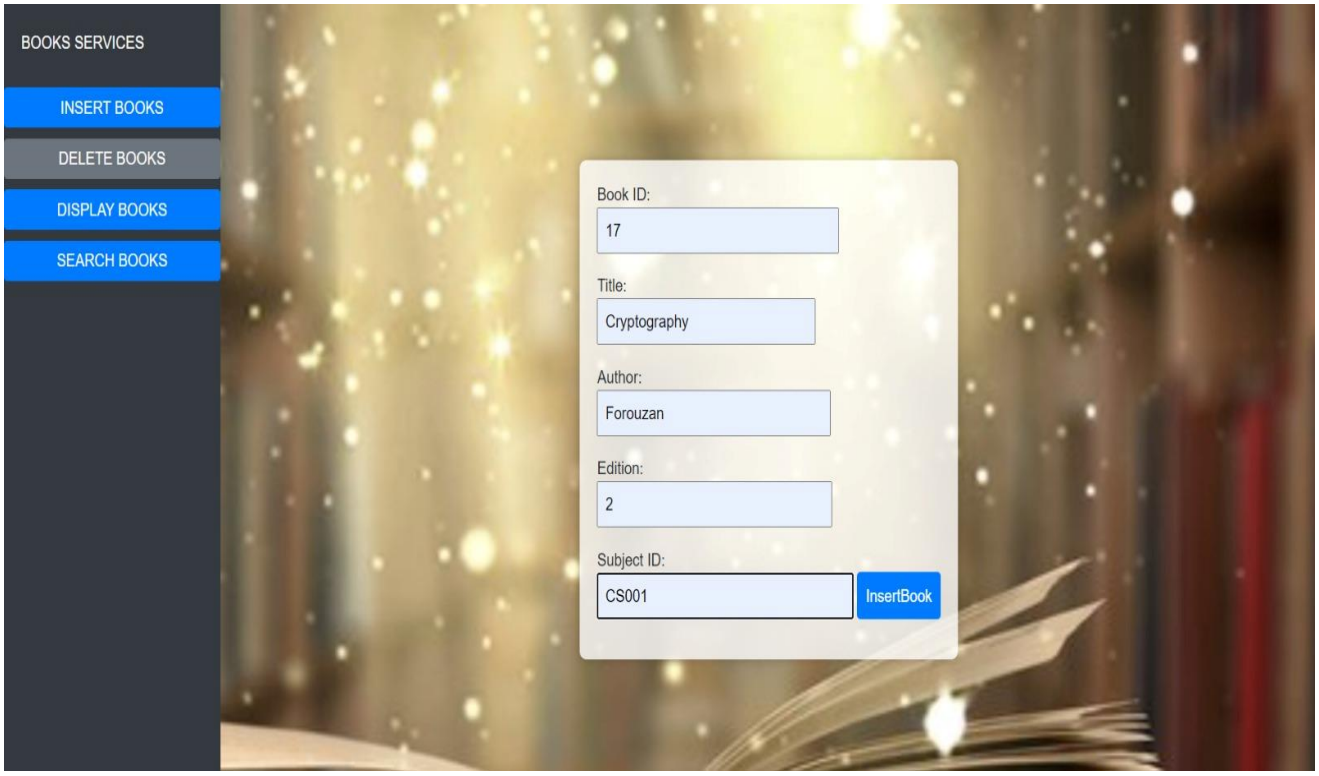
CONTACT US PAGE

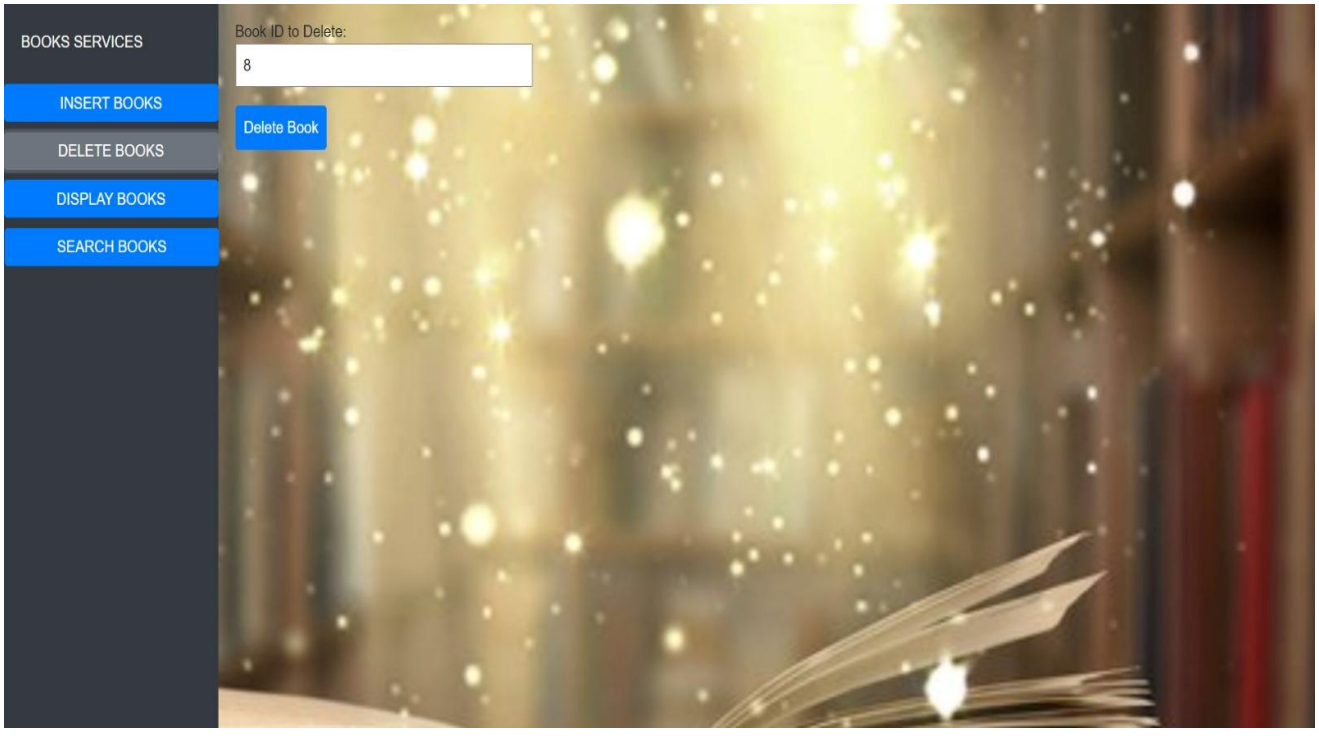


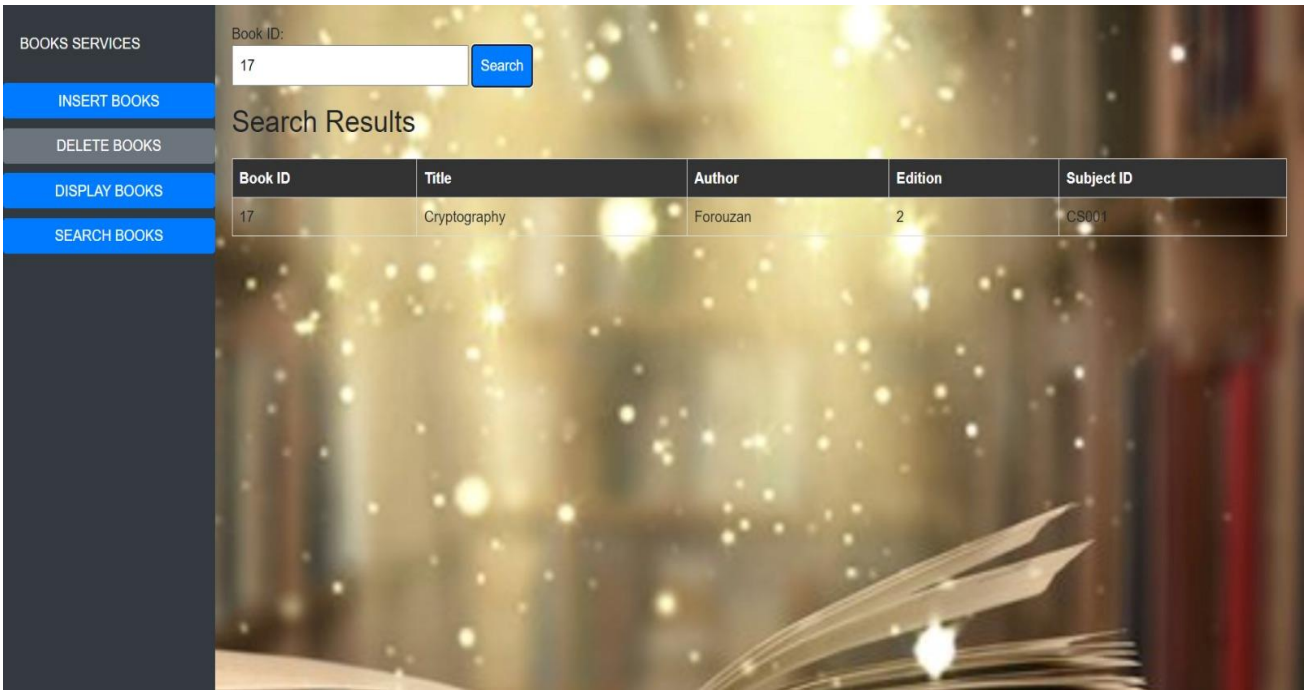
SERVICES PAGE

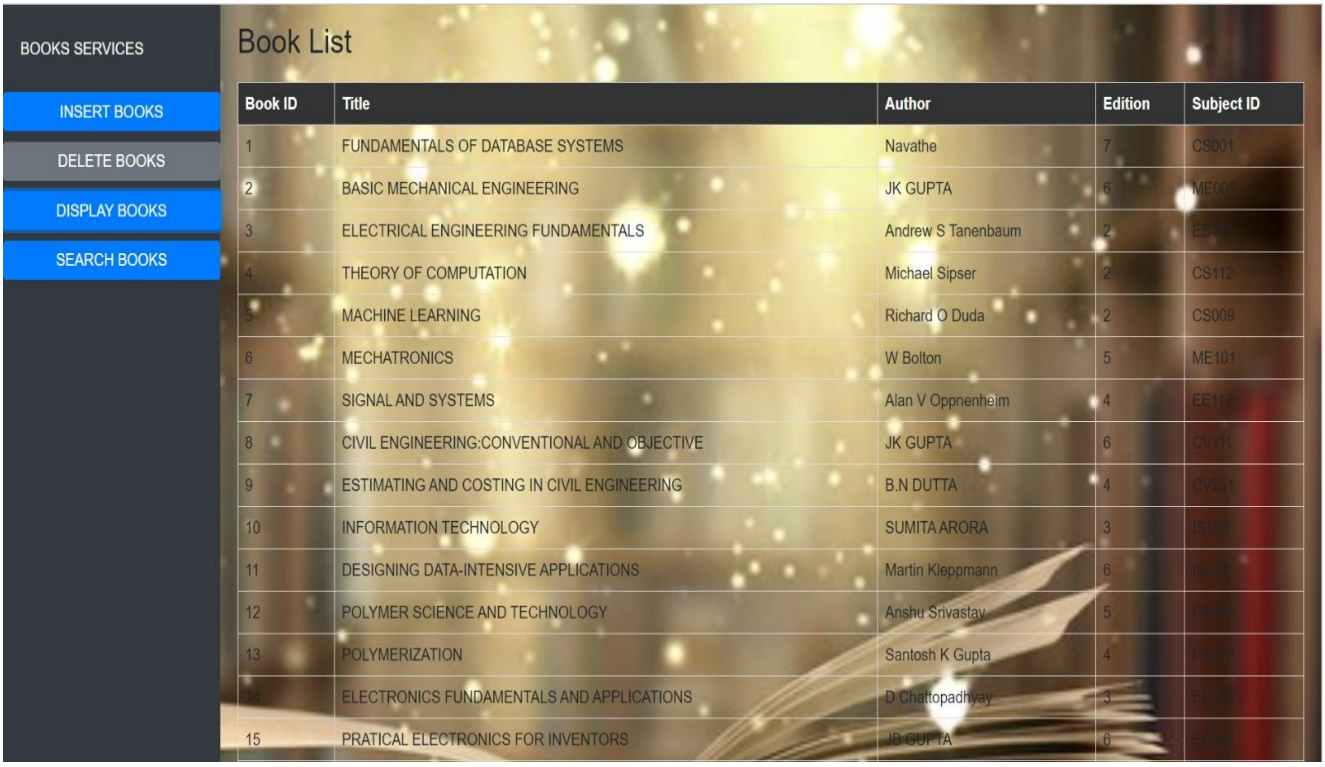


BOOK FORM

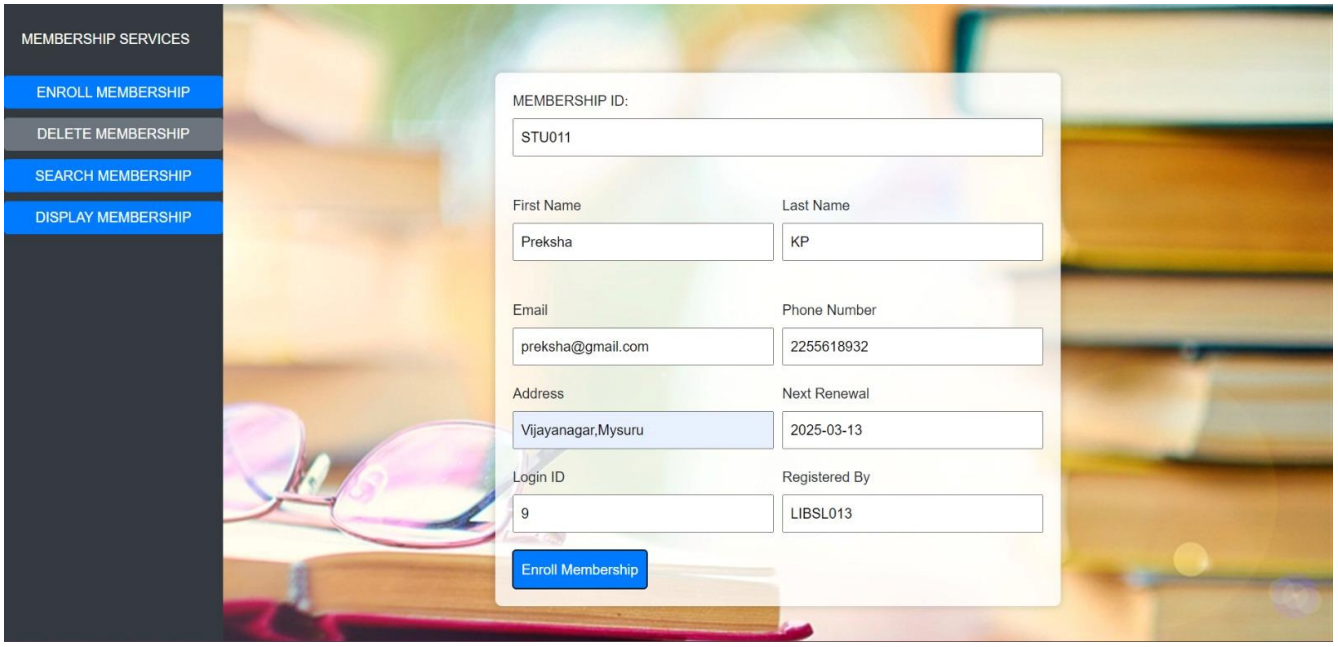

BOOKS SERVICES
INSERT BOOKS
DELETE BOOKS
DISPLAY BOOKS
SEARCH BOOKS
Book ID:
17
Title:
Cryptography
Author:
Forouzan
Edition:
2
Subject ID:
CS001
InsertBook

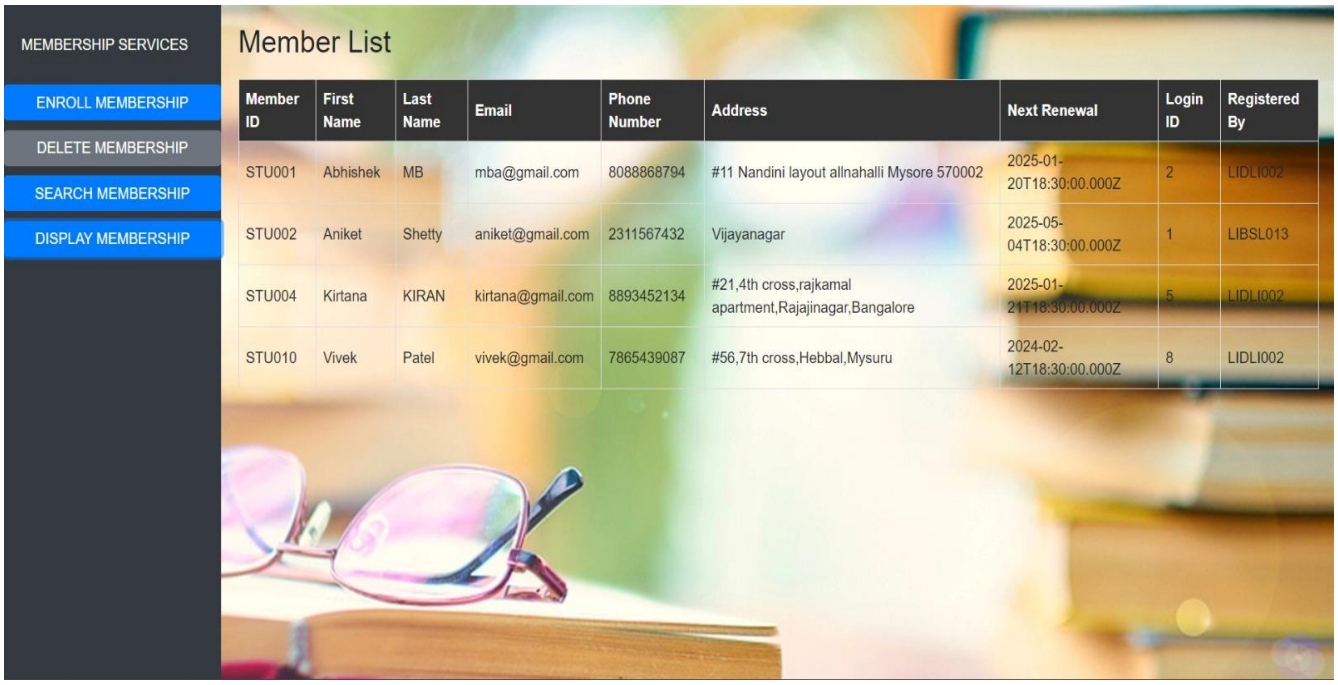

BOOKS SERVICES
INSERT BOOKS
DELETE BOOKS
DISPLAY BOOKS
SEARCH BOOKS
Book ID to Delete:
8
Delete Book

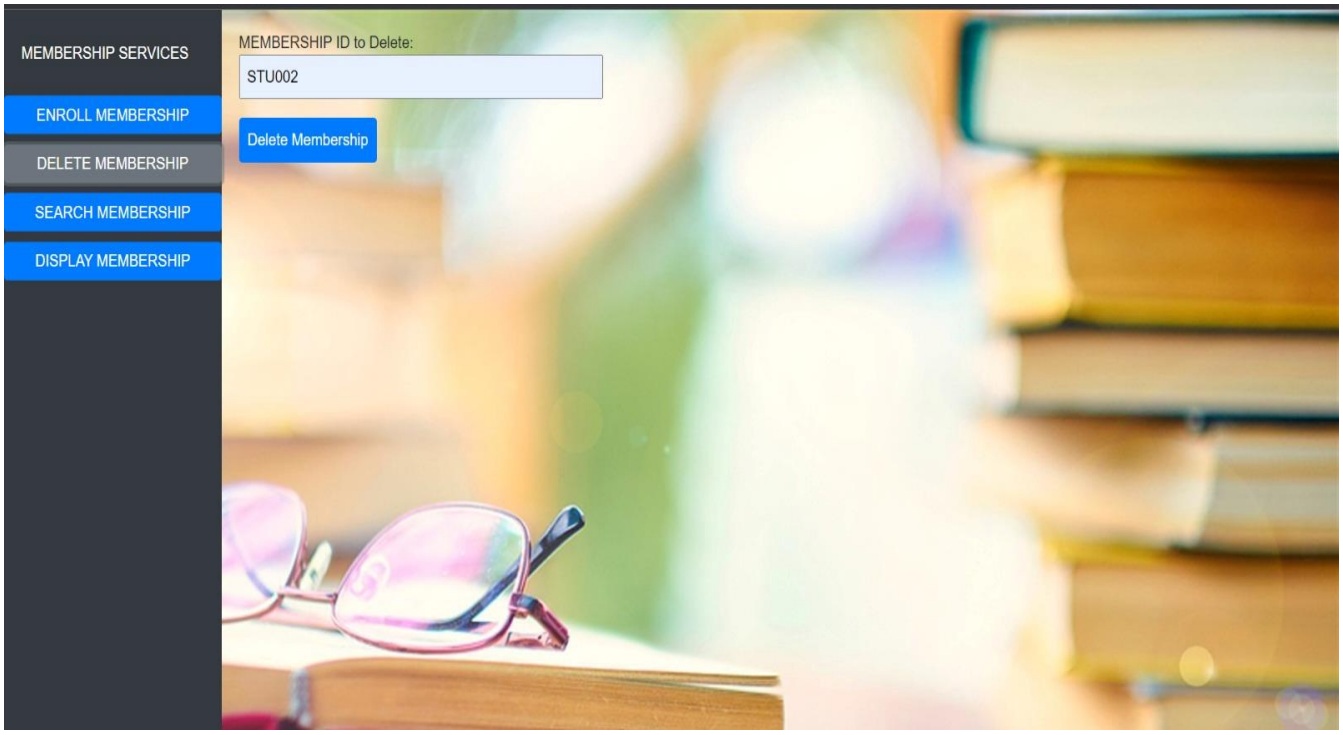

BOOKS SERVICES
INSERT BOOKS
DELETE BOOKS
DISPLAY BOOKS
SEARCH BOOKS
Book ID:
17
Search
Search Results
Book ID
Title
Author
Edition
Subject ID
17
Cryptography
Forouzan
2
CS001

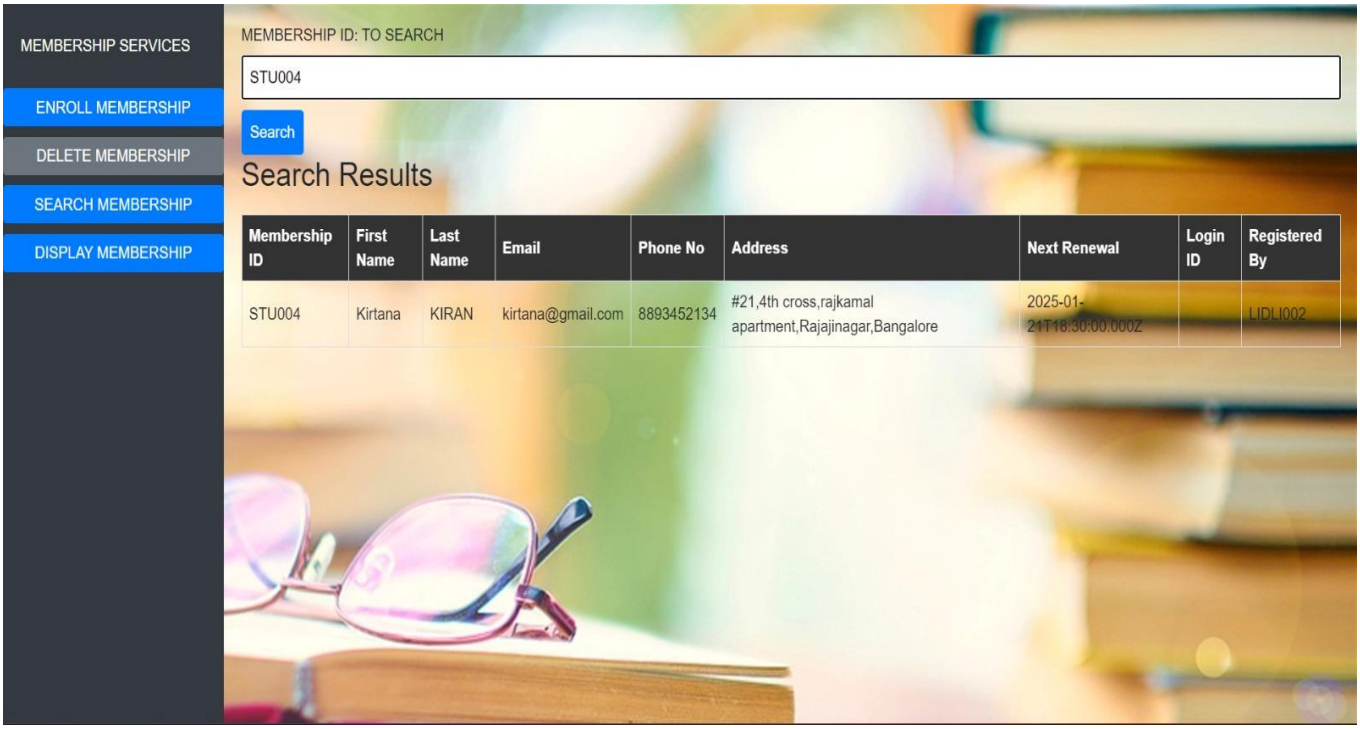

BOOKS SERVICES
INSERT BOOKS
DELETE BOOKS
DISPLAY BOOKS
SEARCH BOOKS
Book List
Book ID
Title
Author
Edition
Subject ID
1
FUNDAMENTALS OF DATABASE SYSTEMS
Navathe
7
CS001
2
BASIC MECHANICAL ENGINEERING
JK GUPTA
6
3
ELECTRICAL ENGINEERING FUNDAMENTALS
Andrew S Tanenbaum
2
4
THEORY OF COMPUTATION
Michael Sipser
2
CS112
5
MACHINE LEARNING
Richard O Duda
2
CS009
6
MECHATRONICS
W Bolton
5
ME101
7
SIGNAL AND SYSTEMS
Alan V Oppnenheim
4
8
CIVIL ENGINEERING:CONVENTIONAL AND OBJECTIVE
JK GUPTA
6
9
ESTIMATING AND COSTING IN CIVIL ENGINEERING
B.N DUTTA
4
10
INFORMATION TECHNOLOGY
SUMITA ARORA
3
11
DESIGNING DATA-INTENSIVE APPLICATIONS
Martin Kleppmann
6
12
POLYMER SCIENCE AND TECHNOLOGY
Anshu Srivastav
5
13
POLYMERIZATION
Santosh K Gupta
4
ELECTRONICS FUNDAMENTALS AND APPLICATIONS
D Chattopadhyay
3
15
PRATICAL ELECTRONICS FOR INVENTORS
JB GUPTA
6

MEMBERSHIP FORM





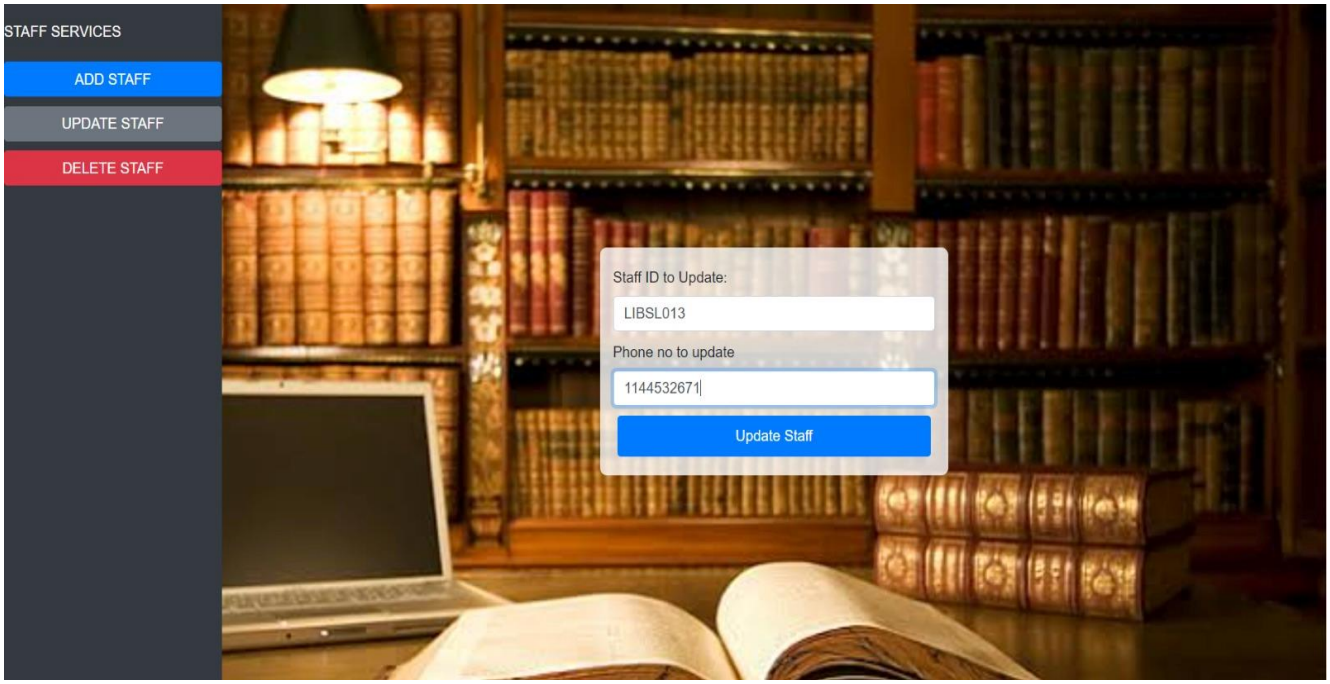

MEMBERSHIP SERVICES
ENROLL MEMBERSHIP
DELETE MEMBERSHIP
SEARCH MEMBERSHIP
DISPLAY MEMBERSHIP
MEMBERSHIP ID to Delete:
STU002
Delete Membership


MEMBERSHIP SERVICES
ENROLL MEMBERSHIP
DELETE MEMBERSHIP
SEARCH MEMBERSHIP
DISPLAY MEMBERSHIP
MEMBERSHIP ID: TO SEARCH
STU004
Search
Search Results
Membership ID
First Name
Last Name
Email
Phone No
Address
Next Renewal
Login ID
Registered By
STU004
Kirtana
KIRAN
kirtana@gmail.com
8893452134
#21,4th cross,rajkamal apartment,Rajajinagar,Bangalore
2025-01-21T18:30:00.000Z
LIDLI002

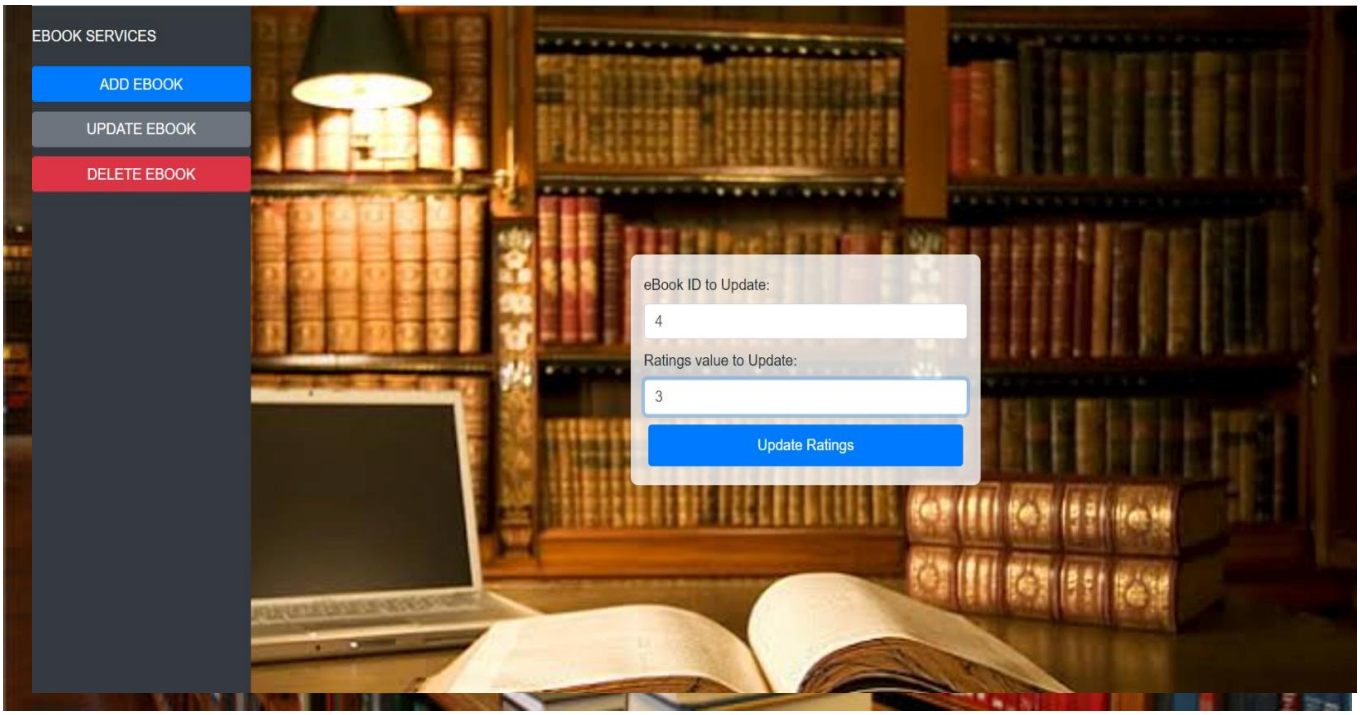
STAFF FORM


STAFF SERVICES
ADD STAFF
UPDATE STAFF
DELETE STAFF
Staff ID:
LIDTE004
First Name:
Priya
Last Name:
SM
Gender:
Female
Designation:
Support_Staff
Phone Number:
2245167548
Address:
Vijayanagar
Joining Year:
1998
Add Staff


STAFF SERVICES
ADD STAFF
UPDATE STAFF
DELETE STAFF
Staff ID to Delete:
LIDTE003
Delete Staff


STAFF SERVICES
ADD STAFF
UPDATE STAFF
DELETE STAFF
Staff ID to Update:
LIBSL013
Phone no to update
1144532671
Update Staff

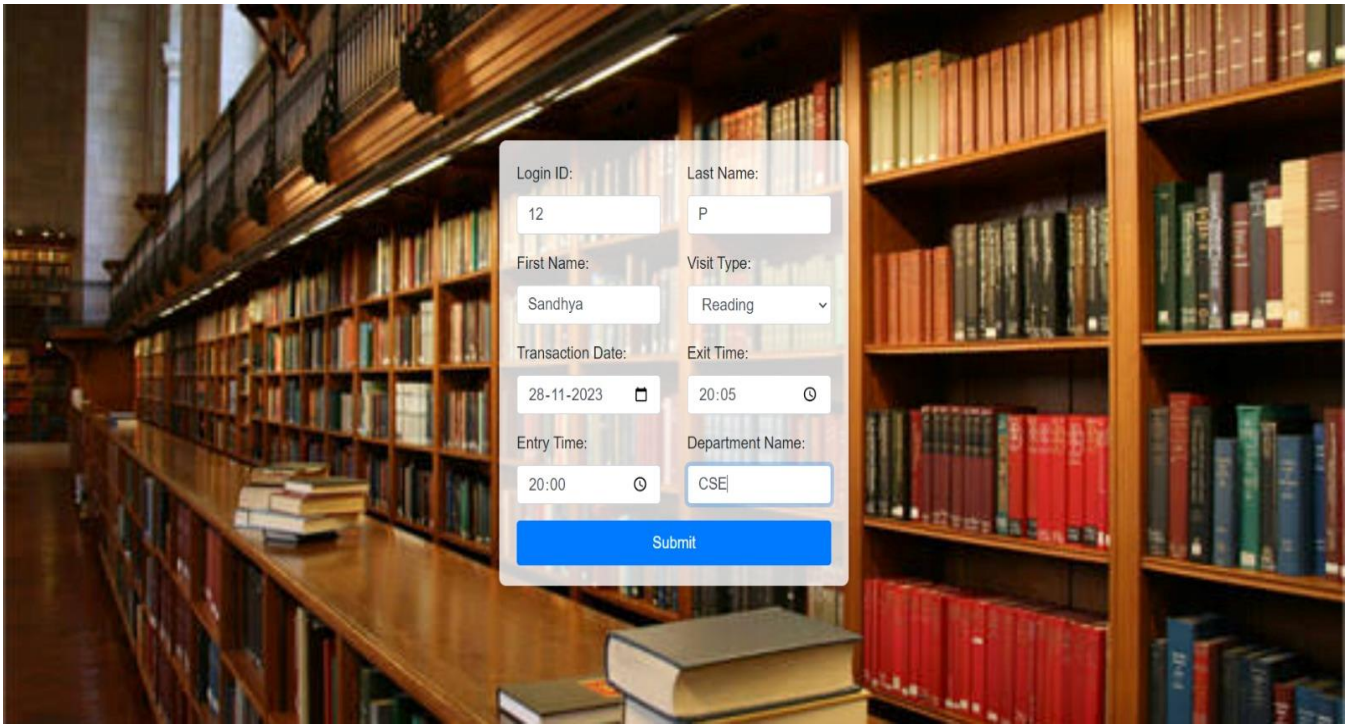
EBOOK FORM


EBOOK SERVICES
ADD EBOOK
UPDATE EBOOK
DELETE EBOOK
eBook ID to Update:
4
Ratings value to Update:
3
Update Ratings


EBOOK SERVICES
ADD EBOOK
UPDATE EBOOK
DELETE EBOOK
EBook ID:
1
Title:
Database Manageme
Author
Navathe
Category
CS001
URL
https://www.academi
Rating
5
Format
PDF
Add EBook


EBOOK SERVICES
ADD EBOOK
UPDATE EBOOK
DELETE EBOOK
eBook ID to Delete:
14
Delete eBook

LOGIN FORM

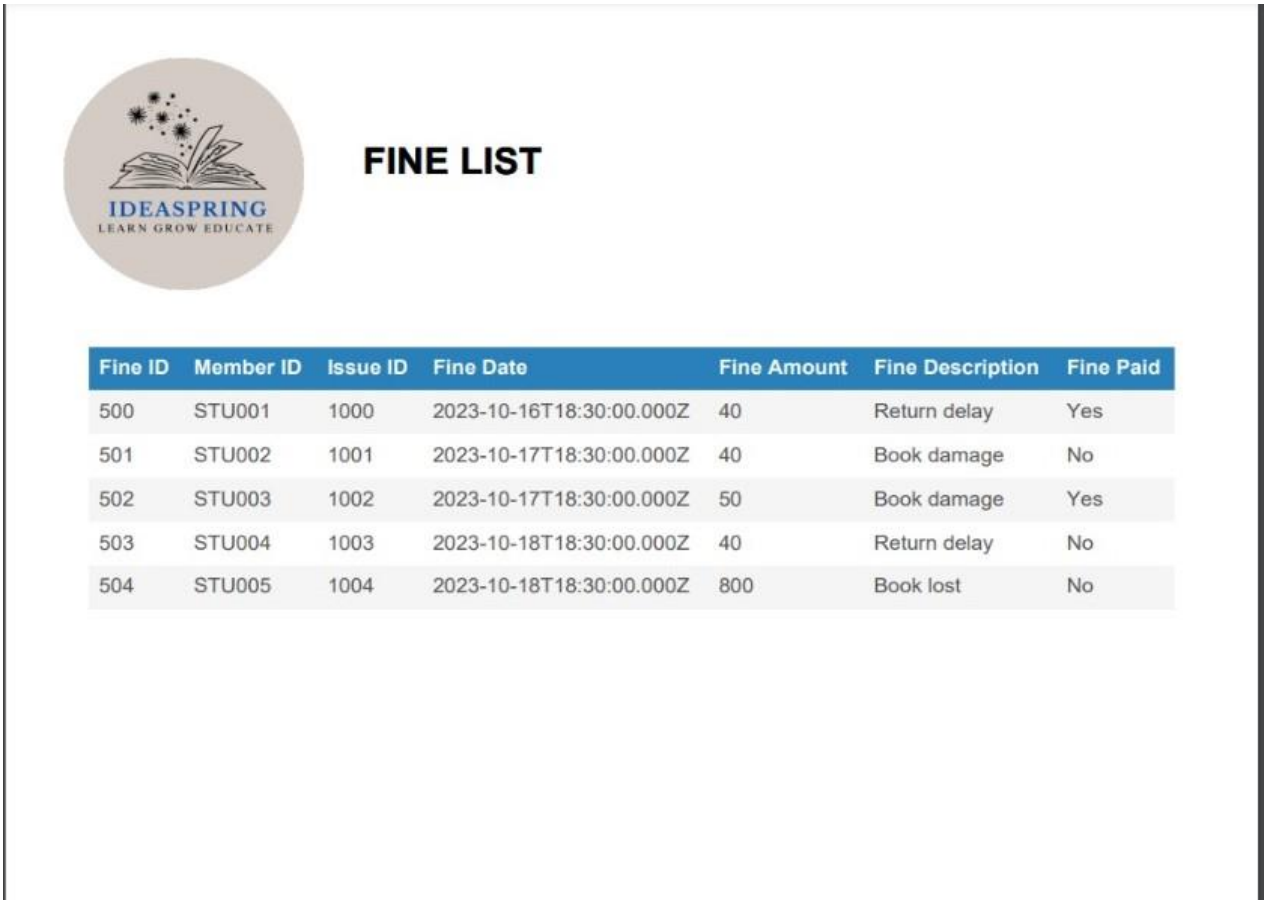


8. REPORT GENERATION

IDEASPRING
LEARN GROW EDUCATE

BOOKS

Book ID	Title	Author	Edition	Category
1	FUNDAMENTALS OF DATABASE SYSTEMS	Navathe	7	CS001
2	BASIC MECHANICAL ENGINEERING	JK GUPTA	6	ME005
3	ELECTRICAL ENGINEERING FUNDAMENTALS	Andrew S Tanenbaum	2	EE108
4	THEORY OF COMPUTATION	Michael Sipser	2	CS112
5	MACHINE LEARNING	Richard O Duda	2	CS009
6	MECHATRONICS	W Bolton	5	ME101
7	SIGNAL AND SYSTEMS	Alan V Oppnenheim	4	EE112
8	CIVIL ENGINEERING:CONVENTIONAL AND OBJECTIVE	JK GUPTA	6	CV111
9	ESTIMATING AND COSTING IN CIVIL ENGINEERING	B.N DUTTA	4	CV231
10	INFORMATION TECHNOLOGY	SUMITA ARORA	3	IS109
11	DESIGNING DATA-INTENSIVE APPLICATIONS	Martin Kleppmann	6	IS222
12	POLYMER SCIENCE AND TECHNOLOGY	Anshu Srivastav	5	PS004
13	POLYMERIZATION	Santosh K Gupta	4	PS013
14	ELECTRONICS FUNDAMENTALS AND APPLICATIONS	D Chattopadhyay	3	EC118
15	PRATICAL ELECTRONICS FOR INVENTORS	JB GUPTA	6	EC222



Fine ID	Member ID	Issue ID	Fine Date	Fine Amount	Fine Description	Fine Paid
500	STU001	1000	2023-10-16T18:30:00.000Z	40	Return delay	Yes
501	STU002	1001	2023-10-17T18:30:00.000Z	40	Book damage	No
502	STU003	1002	2023-10-17T18:30:00.000Z	50	Book damage	Yes
503	STU004	1003	2023-10-18T18:30:00.000Z	40	Return delay	No
504	STU005	1004	2023-10-18T18:30:00.000Z	800	Book lost	No

9. CONCLUSION

The Library Management System is essential for colleges, schools and many more places these days. A lot of manual work can be reduced with this Library Management System. And also, a lot of glitches like wrong borrow date and miscalculation of fine amount can be avoided. This computer-managed system is efficient and cost-effective. The Library Management System stores details of books, e-book, staff, members, and fine details as well. So overall we have learnt: how to build a database to store related information, how to build tables separately to store data, implementations of MySQL and how the software allows storing all the details related to the library. This system makes entire process online where student can search books, staff can generate reports and maintain book transaction. The database schema is well-designed to organize information about departments, books, eBooks, staff, members, login activities, issued books, fines, and more. This ensures efficient data retrieval and management. The project demonstrates the use of relationships between tables, enforcing referential integrity through foreign key constraints. This ensures that data remains consistent and accurate across related tables. The implementation of cascading referential actions for certain foreign

key relationships ensures data consistency. For example, when a department is deleted, related books or eBooks are also removed, preventing orphaned records. In conclusion, the Library Management System DBMS project demonstrates effective database design principles, providing a robust solution for managing library operations. The project's attention to data integrity, security, and user-friendliness makes it a valuable tool for libraries aiming to streamline their processes.

10.REFERENCES

- https://www.w3schools.com/MySQL/default.asp
- https://www.oracle.com/mysql/what-is-mysql/
- https://www.daitm.org.in/wp-content/uploads/2019/04/Gr.-06library-project-report.pdf
- https://www.researchgate.net/publication/347245735_Library_Management_System
- https://www.academia.edu/37726542/Library_Management_System_Mini_Project_R

WEBSITE LINK (IDEASPRING)

- https://webdbms.vercel.app