Database Testing

A Library Management System

Created – Md. Rashadul Islam

Date – July 20, 2023

Contents

1. Introduction……. 2
2. Objective……. 2
3. Tools……..... 3
4. ER diagram……... 3
5. Test Scope……... 5
6.Result Analysis ……..... 5
7.Testing Approach……..... 20
8.Conclusion …….…... 20

1. Introduction

Database testing is essential to ensure that a Library Management System (LMS) runs smoothly and reliably. It involves verifying the database's accuracy, integrity, and performance, which serves as the system's backbone.

STUDENTS, BOOK, USERS, BORROWING, TRANSACTIONS, and REPORTS are some of the most important tables in the Library Management System database. Each table contains data about students, books, users, borrowing transactions, and reports. The database was created to keep track of students, books, and their interactions with the system.

We verify that the system's database runs correctly, keeping accurate information and its optimum efficiency, through careful examination. The document includes an overview of the testing methodology, scenarios run, outcomes, and any issues encountered with their solutions.

A successful database testing process ensures that the Library Management System is well-prepared to provide an orderly and dependable library resource management system to the academic community.

2. Objectives

The testing effort's goals can be stated as follows:

1. **Verify Database Creation:** Check if the database "Library_Management_System" was properly established with the appropriate values.

2. **Validate Table Structure:** To ensure data integrity and consistency, validate table structures, including primary keys, foreign keys, and constraints.

3. **Ensure Accurate Data Insertion:** Validate that data insertion into database tables is accurate and follows the appropriate business standards.

4. **Validate Query Functionality:** Confirm the functionality of various database queries, such as "SELECT, INSERT, UPDATE, DELETE, and JOIN" procedures.

5. **Maintain Data Integrity:** To prevent data inconsistencies, verify the data validation rules, constraints, and triggers.

6. **Assess Database Performance:** To achieve optimal response times, evaluate the performance of database queries under various load scenarios.

7. **Test Backup and Recovery Procedures:** Ensure that database backup and recovery procedures are tested successfully, allowing data restoration as needed.

3. Tools

1. **MySQL workbench :** Database create, insert and query run
2. **MS word:** For documentation
3. **Snipping Tool:** For Screen shot

4. ER - Diagram

The ER diagram for database testing of the Library Management System represents the database schema, including tables, relationships, and key fields, ensuring data integrity and validating table structures.

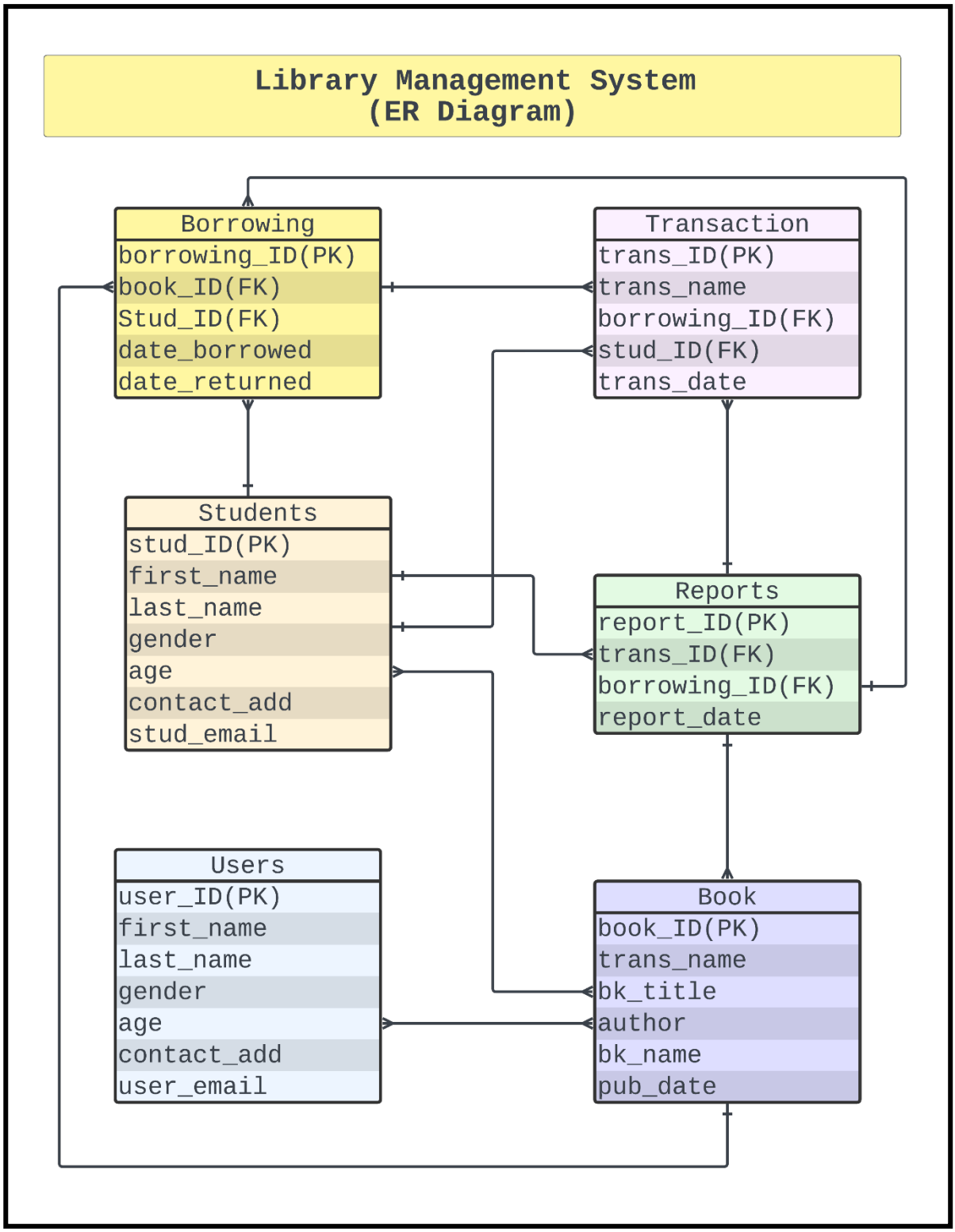

Fig – 1: ER – Diagram for Library Management System

5. Test Scope

The database testing covered the following areas:

1. **Database Creation:** Ensure the database is created successfully and with the expected settings.
2. **Table Structure:** Verify the correctness of table structures, including primary keys, foreign keys, and constraints.
3. **Data Insertion:** Ensure that data insertion into tables is accurate and meets the required business rules.
4. **Query Execution:** Validate the functionality of various database queries, including SELECT, INSERT, UPDATE, DELETE, and JOIN operations.
5. **Data Integrity:** Verify the data validation rules, constraints, and triggers to maintain data integrity.
6. **Performance:** Assess the performance of database queries under different load conditions.
7. **Backup and Recovery:** Test the database backup and recovery procedures to ensure data can be restored as needed.

6. Result Analysis

In this chapter, we will analyze the results obtained from the comprehensive database testing of the Library Management System to assess the system's performance, data integrity, and overall reliability.

1. Database Creation and using :

```sql
-- Database Creating
create database Library_Management_System;
-- Using the database
USE Library_Management_System;
```

Fig –2: Database Creation and Using

Status: Pass

2. Creating Tables:

```sql
-- Creating Tables
CREATE TABLE STUDENTS (
    stud_ID INT NOT NULL,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    gender CHAR(1),
    age INT,
    contact_add VARCHAR(255),
    stud_email VARCHAR(255),
    PRIMARY KEY (stud_ID)
);
```

Fig – 3: Student Table Create

```sql

CREATE TABLE BOOK (
    book_ID INT NOT NULL,
    bk_title VARCHAR(255),
    author VARCHAR(255),
    bk_num INT,
    pub_date DATE NOT NULL,
    PRIMARY KEY (book_ID)
);

```

Fig- 4: Book Table Create

```sql
CREATE TABLE USERS (
    user_ID INT NOT NULL,
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    gender CHAR(2),
    age INT,
    contact_add VARCHAR(255),
    user_email VARCHAR(255),
    PRIMARY KEY (user_ID)
);

```

Fig – 5: Users Table Create

```sql
CREATE TABLE BORROWING (
    borrowing_ID INT NOT NULL AUTO_INCREMENT,
    book_ID INT,
    stud_ID INT,
    data_borrowed DATE NOT NULL,
    data_return DATE NOT NULL,
    PRIMARY KEY (borrowing_ID),
    FOREIGN KEY (book_ID)
        REFERENCES BOOK (book_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (stud_ID)
        REFERENCES STUDENTS (stud_ID)
        ON DELETE CASCADE
);
```

Fig – 6: Borrowing Table Create

```sql
CREATE TABLE TRANSACTIONS (
    trans_ID INT NOT NULL,
    trans_name VARCHAR(255),
    borrowing_ID INT,
    stud_ID INT,
    trans_date DATE NOT NULL,
    PRIMARY KEY (trans_ID),
    FOREIGN KEY (borrowing_ID)
        REFERENCES BORROWING (borrowing_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (stud_ID)
        REFERENCES STUDENTS (stud_ID)
        ON DELETE CASCADE
);
```

Fig – 7: Transactions Table Create

```sql
CREATE TABLE Reports (
    report_ID INT NOT NULL AUTO_INCREMENT,
    trans_ID INT,
    borrowing_ID INT,
    report_date DATE NOT NULL,
    PRIMARY KEY (report_ID),
    FOREIGN KEY (trans_ID)
        REFERENCES TRANSACTIONS (trans_ID)
        ON DELETE CASCADE,
    FOREIGN KEY (borrowing_ID)
        REFERENCES BORROWING (borrowing_ID)
        ON DELETE CASCADE
);
```

Fig – 8: Reports Table Create

2.1 Result:

Result Grid | Filter Rows:

Tables_in_library_management_system
book
borrowing
reports
students
transactions
users

Fig – 9: All Table Successfully Created

2.2 Status: PASS

3. Inserting Data into Created Tables:

3.1 Student Table:

```sql
-- INSERT statements for STUDENTS table
INSERT INTO STUDENTS (stud_ID, first_name, last_name, gender, age, contact_add, stud_email)
VALUES
    (1, 'John', 'Doe', 'M', 20, '123 Main St', 'john.doe@example.com'),
    (2, 'Jane', 'Smith', 'F', 22, '456 Elm St', 'jane.smith@example.com'),
    (3, 'Michael', 'Johnson', 'M', 19, '789 Oak Ave', 'michael.johnson@example.com'),
    (4, 'Emily', 'Williams', 'F', 21, '101 Pine Rd', 'emily.williams@example.com'),
    (5, 'David', 'Brown', 'M', 23, '222 Maple Blvd', 'david.brown@example.com'),
    (6, 'Sarah', 'Davis', 'F', 20, '333 Cedar Ln', 'sarah.davis@example.com'),
    (7, 'Christopher', 'Miller', 'M', 22, '444 Birch Rd', 'christopher.miller@example.com'),
    (8, 'Jessica', 'Wilson', 'F', 19, '555 Willow Dr', 'jessica.wilson@example.com'),
    (9, 'Daniel', 'Taylor', 'M', 21, '666 Oak St', 'daniel.taylor@example.com'),
    (10, 'Ashley', 'Anderson', 'F', 23, '777 Pine Ave', 'ashley.anderson@example.com'),
    (11, 'James', 'Thomas', 'M', 20, '888 Elm Rd', 'james.thomas@example.com'),
    (12, 'Amanda', 'Jackson', 'F', 22, '999 Cedar St', 'amanda.jackson@example.com');
```

Fig – 10: Insert Data into Students Table

3.1.2 Result:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

stud_ID	first_name	last_name	gender	age	contact_add	stud_email
1	John	Doe	M	20	123 Main St	john.doe@example.com
2	Jane	Smith	F	22	456 Elm St	jane.smith@example.com
3	Michael	Johnson	M	19	789 Oak Ave	michael.johnson@example.com
4	Emily	Williams	F	21	101 Pine Rd	emily.williams@example.com
5	David	Brown	M	23	222 Maple Blvd	david.brown@example.com
6	Sarah	Davis	F	20	333 Cedar Ln	sarah.davis@example.com
7	Christopher	Miller	M	22	444 Birch Rd	christopher.miller@example.com
8	Jessica	Wilson	F	19	555 Willow Dr	jessica.wilson@example.com
9	Daniel	Taylor	M	21	666 Oak St	daniel.taylor@example.com
10	Ashley	Anderson	F	23	777 Pine Ave	ashley.anderson@example.com
11	James	Thomas	M	20	888 Elm Rd	james.thomas@example.com
12	Amanda	Jackson	F	22	999 Cedar St	amanda.jackson@example.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig – 11: After insertion in Student Table

3.1.3 Status: PASS

3.2 Book Table:

```sql
-- INSERT statements for Book table
INSERT INTO BOOK (book_ID, bk_title, author, bk_num, pub_date)
VALUES
    (1, 'The Great Gatsby', 'F. Scott Fitzgerald', 12345, '2020-01-15'),
    (2, 'To Kill a Mockingbird', 'Harper Lee', 67890, '2019-08-30'),
    (3, '1984', 'George Orwell', 54321, '2018-03-22'),
    (4, 'Pride and Prejudice', 'Jane Austen', 98765, '2017-11-10'),
    (5, 'The Catcher in the Rye', 'J.D. Salinger', 24680, '2016-06-05'),
    (6, 'To Kill a Kingdom', 'Alexandra Christo', 13579, '2021-04-25'),
    (7, 'Harry Potter and the Sorcerer''s Stone', 'J.K. Rowling', 86420, '2005-12-01'),
    (8, 'The Hobbit', 'J.R.R. Tolkien', 97531, '2015-09-18'),
    (9, 'The Da Vinci Code', 'Dan Brown', 73529, '2003-07-30'),
    (10, 'The Hunger Games', 'Suzanne Collins', 62840, '2014-02-14'),
    (11, 'The Alchemist', 'Paulo Coelho', 28192, '1993-10-15'),
    (12, 'The Lord of the Rings: The Fellowship of the Ring', 'J.R.R. Tolkien', 38462, '2001-05-02');
```

Fig – 12: Insert Data into Book Table

3.2.1 Result:

Result Grid | Filter Rows: | Edit: | Export/Import:

book_ID	bk_title	author	bk_num	pub_date
1	The Great Gatsby	F. Scott Fitzgerald	12345	2020-01-15
2	To Kill a Mockingbird	Harper Lee	67890	2019-08-30
3	1984	George Orwell	54321	2018-03-22
4	Pride and Prejudice	Jane Austen	98765	2017-11-10
5	The Catcher in the Rye	J.D. Salinger	24680	2016-06-05
6	To Kill a Kingdom	Alexandra Christo	13579	2021-04-25
7	Harry Potter and the Sorcerer's Stone	J.K. Rowling	86420	2005-12-01
8	The Hobbit	J.R.R. Tolkien	97531	2015-09-18
9	The Da Vinci Code	Dan Brown	73529	2003-07-30
10	The Hunger Games	Suzanne Collins	62840	2014-02-14
11	The Alchemist	Paulo Coelho	28192	1993-10-15
12	The Lord of the Rings: The Fellowshi...	J.R.R. Tolkien	38462	2001-05-02
NULL	NULL	NULL	NULL	NULL

Fig – 13: After Insertion in Book Table

3.2.2 Status: PASS

3.3 Users Table:

```sql
-- INSERT statements for USERS table
INSERT INTO USERS (user_ID, first_name, last_name, gender, age, contact_add, user_email)
VALUES
    (1, 'Robert', 'Johnson', 'M', 28, '123 Oak St', 'robert.johnson@example.com'),
    (2, 'Emily', 'Smith', 'F', 25, '456 Maple Ave', 'emily.smith@example.com'),
    (3, 'Michael', 'Brown', 'M', 30, '789 Elm Rd', 'michael.brown@example.com'),
    (4, 'Sophia', 'Davis', 'F', 22, '101 Pine St', 'sophia.davis@example.com'),
    (5, 'William', 'Anderson', 'M', 26, '222 Birch Dr', 'william.anderson@example.com'),
    (6, 'Olivia', 'Miller', 'F', 24, '333 Cedar Ln', 'olivia.miller@example.com'),
    (7, 'James', 'Wilson', 'M', 29, '444 Willow Rd', 'james.wilson@example.com'),
    (8, 'Emma', 'Taylor', 'F', 27, '555 Oak Ave', 'emma.taylor@example.com'),
    (9, 'Alexander', 'Martinez', 'M', 23, '666 Maple Blvd', 'alexander.martinez@example.com'),
    (10, 'Ava', 'Lee', 'F', 21, '777 Elm St', 'ava.lee@example.com'),
    (11, 'Daniel', 'Garcia', 'M', 31, '888 Pine Rd', 'daniel.garcia@example.com'),
    (12, 'Isabella', 'Lopez', 'F', 33, '999 Cedar Ave', 'isabella.lopez@example.com');
```

Fig – 14: Insert Data into Users Table

3.3.1 Result:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell

user_ID	first_name	last_name	gender	age	contact_add	user_email
1	Robert	Johnson	M	28	123 Oak St	robert.johnson@example.com
2	Emily	Smith	F	25	456 Maple Ave	emily.smith@example.com
3	Michael	Brown	M	30	789 Elm Rd	michael.brown@example.com
4	Sophia	Davis	F	22	101 Pine St	sophia.davis@example.com
5	William	Anderson	M	26	222 Birch Dr	william.anderson@example.com
6	Olivia	Miller	F	24	333 Cedar Ln	olivia.miller@example.com
7	James	Wilson	M	29	444 Willow Rd	james.wilson@example.com
8	Emma	Taylor	F	27	555 Oak Ave	emma.taylor@example.com
9	Alexander	Martinez	M	23	666 Maple Blvd	alexander.martinez@example.com
10	Ava	Lee	F	21	777 Elm St	ava.lee@example.com
11	Daniel	Garcia	M	31	888 Pine Rd	daniel.garcia@example.com
12	Isabella	Lopez	F	33	999 Cedar Ave	isabella.lopez@example.com
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig – 15: After Insertion in Users Table

3.3.2 Status: PASS

3.4 Borrowing Table:

```sql
-- INSERT statements for BORROWING table
INSERT INTO BORROWING (book_ID, stud_ID, data_borrowed, data_return)
VALUES
    (1, 3, '2023-07-01', '2023-07-15'),
    (2, 5, '2023-07-02', '2023-07-16'),
    (3, 8, '2023-07-03', '2023-07-17'),
    (4, 2, '2023-07-04', '2023-07-18'),
    (5, 10, '2023-07-05', '2023-07-19'),
    (6, 1, '2023-07-06', '2023-07-20'),
    (7, 6, '2023-07-07', '2023-07-21'),
    (8, 4, '2023-07-08', '2023-07-22'),
    (9, 9, '2023-07-09', '2023-07-23'),
    (10, 7, '2023-07-10', '2023-07-24'),
    (11, 12, '2023-07-11', '2023-07-25'),
    (12, 11, '2023-07-12', '2023-07-26');
```

Fig – 16: Insert Data into Borrowing Table

3.4.1 Result:

Result Grid | Filter Rows: | Edit:

borrowing_ID	book_ID	stud_ID	data_borrowed	data_return
1	1	3	2023-07-01	2023-07-15
2	2	5	2023-07-02	2023-07-16
3	3	8	2023-07-03	2023-07-17
4	4	2	2023-07-04	2023-07-18
5	5	10	2023-07-05	2023-07-19
6	6	1	2023-07-06	2023-07-20
7	7	6	2023-07-07	2023-07-21
8	8	4	2023-07-08	2023-07-22
9	9	9	2023-07-09	2023-07-23
10	10	7	2023-07-10	2023-07-24
11	11	12	2023-07-11	2023-07-25
12	12	11	2023-07-12	2023-07-26
NULL	NULL	NULL	NULL	NULL

Fig – 17: After Insertion in Borrowing Table

3.4.2 Status: PASS

3.5 Transaction Table:

```sql
-- INSERT statements for TRANSACTIONS table
INSERT INTO TRANSACTIONS (trans_ID, trans_name, borrowing_ID, stud_ID, trans_date)
VALUES
    (1, 'Fine Payment', 3, 8, '2023-07-15'),
    (2, 'Book Return', 1, 3, '2023-07-15'),
    (3, 'Fine Payment', 9, 7, '2023-07-22'),
    (4, 'Book Return', 2, 5, '2023-07-16'),
    (5, 'Book Return', 4, 2, '2023-07-18'),
    (6, 'Fine Payment', 11, 12, '2023-07-25'),
    (7, 'Fine Payment', 5, 10, '2023-07-19'),
    (8, 'Book Return', 6, 1, '2023-07-20'),
    (9, 'Book Return', 8, 4, '2023-07-22'),
    (10, 'Fine Payment', 10, 7, '2023-07-24'),
    (11, 'Fine Payment', 7, 6, '2023-07-21'),
    (12, 'Book Return', 3, 8, '2023-07-17');
```

Fig – 18: Insert Data into Transaction Table

3.5.1 Result:

Result Grid | Filter Rows: | Edit:

trans_ID	trans_name	borrowing_ID	stud_ID	trans_date
1	Fine Payment	3	8	2023-07-15
2	Book Return	1	3	2023-07-15
3	Fine Payment	9	7	2023-07-22
4	Book Return	2	5	2023-07-16
5	Book Return	4	2	2023-07-18
6	Fine Payment	11	12	2023-07-25
7	Fine Payment	5	10	2023-07-19
8	Book Return	6	1	2023-07-20
9	Book Return	8	4	2023-07-22
10	Fine Payment	10	7	2023-07-24
11	Fine Payment	7	6	2023-07-21
12	Book Return	3	8	2023-07-17
NULL	NULL	NULL	NULL	NULL

Fig – 19: After Insertion in Transaction Table

3.5.2 Status: PASS

3.6 Reports Table:

```sql
-- INSERT statements for Reports table
INSERT INTO Reports (trans_ID, borrowing_ID, report_date)
VALUES
    (1, 3, '2023-07-15'),
    (2, 1, '2023-07-15'),
    (3, 9, '2023-07-22'),
    (4, 2, '2023-07-16'),
    (5, 4, '2023-07-18'),
    (6, 11, '2023-07-25'),
    (7, 5, '2023-07-19'),
    (8, 6, '2023-07-20'),
    (9, 8, '2023-07-22'),
    (10, 10, '2023-07-24'),
    (11, 7, '2023-07-21'),
    (12, 3, '2023-07-17');
```

Fig – 20: Insert into Reports Table

3.6.1 Result:

Result Grid | Filter Rows: | Edit:

report_ID	trans_ID	borrowing_ID	report_date
1	1	3	2023-07-15
2	2	1	2023-07-15
3	3	9	2023-07-22
4	4	2	2023-07-16
5	5	4	2023-07-18
6	6	11	2023-07-25
7	7	5	2023-07-19
8	8	6	2023-07-20
9	9	8	2023-07-22
10	10	10	2023-07-24
11	11	7	2023-07-21
12	12	3	2023-07-17
NULL	NULL	NULL	NULL

Fig – 19: After Insertion in Transaction Table

3.5.2 Status: PASS

4. Running some Query over the Database:

1. Query:

```
-- Query using AS (alias) and ORDER BY
SELECT stud_ID AS student_id, first_name AS name, age AS student_age
FROM STUDENTS
ORDER BY student_age DESC;
```

1.1 **Result:**

Result Grid | Filter Rows:

student_id	name	student_age
5	David	23
10	Ashley	23
2	Jane	22
7	Christopher	22
12	Amanda	22
4	Emily	21
9	Daniel	21
1	John	20
6	Sarah	20
11	James	20
3	Michael	19
8	Jessica	19

1.2 **Status:** PASS

2. Query:

```
-- Query using GROUP BY and aggregate function (COUNT)
SELECT gender, COUNT(*) AS count_of_students
FROM STUDENTS
GROUP BY gender;
```

2.1 **Result:**

Result Grid | Filter Rows:

gender	count_of_students
M	6
F	6

2.2 **Status:** PASS

3. Query:

```
-- Query using JOIN (INNER JOIN)
SELECT s.first_name, b.bk_title
FROM BORROWING bor
INNER JOIN STUDENTS s ON bor.stud_ID = s.stud_ID
INNER JOIN BOOK b ON bor.book_ID = b.book_ID;
```

3.1 **Result:**

Result Grid | Filter Rows: | Expo

first_name	bk_title
Michael	The Great Gatsby
David	To Kill a Mockingbird
Jessica	1984
Jane	Pride and Prejudice
Ashley	The Catcher in the Rye
John	To Kill a Kingdom
Sarah	Harry Potter and the Sorcerer's Stone
Emily	The Hobbit
Daniel	The Da Vinci Code
Christopher	The Hunger Games
Amanda	The Alchemist
James	The Lord of the Rings: The Fellowshi...

3.2 **Status:** PASS

4. Query:

```
-- Query using WHERE and ORDER BY
SELECT bk_title, author, pub_date
FROM BOOK
WHERE pub_date >= '2022-01-01'
ORDER BY pub_date DESC;
```

4.1 **Result:**

Result Grid | Filter Rows:

bk_title	author	pub_date

4.2 **Status:** PASS

5. Query:

```sql
-- Query using GROUP BY, aggregate function (SUM), and HAVING
SELECT stud_ID, COUNT(*) AS num_of_borrowings, SUM(1) AS total_books_borrowed
FROM BORROWING
GROUP BY stud_ID
HAVING total_books_borrowed > 10;
```

5.2 Result:

Result Grid | Filter Rows: | Export:

	stud_ID	num_of_borrowings	total_books_borrowed

5.3 Status: PASS

6. Query:

```sql
-- Query using LEFT JOIN and IS NULL to find students
-- who have not borrowed any books
SELECT s.stud_ID, s.first_name, s.last_name
FROM STUDENTS s
LEFT JOIN BORROWING b ON s.stud_ID = b.stud_ID
WHERE b.borrowing_ID IS NULL;
```

6.2 Result:

Result Grid | Filter Rows:

	stud_ID	first_name	last_name

6.3 Status: PASS

7. Query:

```sql
-- Query using functions (UPPER) and ORDER BY
SELECT UPPER(first_name) AS capitalized_name
FROM STUDENTS
ORDER BY first_name;
```

7.2 **Result:**

Result Grid | Filter

capitalized_name
AMANDA
ASHLEY
CHRISTOPHER
DANIEL
DAVID
EMILY
JAMES
JANE
JESSICA
JOHN
MICHAEL
SARAH

7.3 **Status:** PASS

8. Query:

```
-- Query using JOIN (LEFT JOIN), functions (CONCAT), and WHERE
SELECT CONCAT(u.first_name, ' ', u.last_name) AS user_fullname, t.trans_name
FROM TRANSACTIONS t
LEFT JOIN USERS u ON t.trans_ID = u.user_ID
WHERE u.age < 30;
```

8.2 **Result:**

Result Grid | Filter Rows:

user_fullname	trans_name
Robert Johnson	Fine Payment
Emily Smith	Book Return
Sophia Davis	Book Return
William Anderson	Book Return
Olivia Miller	Fine Payment
James Wilson	Fine Payment
Emma Taylor	Book Return
Alexander Martinez	Book Return
Ava Lee	Fine Payment

8.3 **Status:** PASS

9. Query:

```sql
-- Query to get the list of books currently borrowed along with their borrowers' information
SELECT b.bk_title AS book_title, b.author, s.first_name AS borrower_firstname, s.last_name AS borrower_lastname
FROM BOOK b
JOIN BORROWING bor ON b.book_ID = bor.book_ID
JOIN STUDENTS s ON bor.stud_ID = s.stud_ID;
```

9.2 Result:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

book_title	author	borrower_firstname	borrower_lastname
The Great Gatsby	F. Scott Fitzgerald	Michael	Johnson
To Kill a Mockingbird	Harper Lee	David	Brown
1984	George Orwell	Jessica	Wilson
Pride and Prejudice	Jane Austen	Jane	Smith
The Catcher in the Rye	J.D. Salinger	Ashley	Anderson
To Kill a Kingdom	Alexandra Christo	John	Doe
Harry Potter and the Sorcerer's Stone	J.K. Rowling	Sarah	Davis
The Hobbit	J.R.R. Tolkien	Emily	Williams
The Da Vinci Code	Dan Brown	Daniel	Taylor
The Hunger Games	Suzanne Collins	Christopher	Miller
The Alchemist	Paulo Coelho	Amanda	Jackson
The Lord of the Rings: The Fellowshi...	J.R.R. Tolkien	James	Thomas

9.3 Status: PASS

10. Query:

```sql
-- Query to calculate the total number of books borrowed by each student and display the results in descending order
SELECT s.stud_ID, s.first_name, s.last_name, COUNT(bor.book_ID) AS total_books_borrowed
FROM STUDENTS s
LEFT JOIN BORROWING bor ON s.stud_ID = bor.stud_ID
GROUP BY s.stud_ID, s.first_name, s.last_name
ORDER BY total_books_borrowed DESC;
```

10.2 Result:

Result Grid | Filter Rows: | Export:

stud_ID	first_name	last_name	total_books_borrowed
1	John	Doe	1
2	Jane	Smith	1
3	Michael	Johnson	1
4	Emily	Williams	1
5	David	Brown	1
6	Sarah	Davis	1
7	Christopher	Miller	1
8	Jessica	Wilson	1
9	Daniel	Taylor	1
10	Ashley	Anderson	1
11	James	Thomas	1
12	Amanda	Jackson	1

10.3 **Status:** PASS

11. Query:

```
-- Query to find out the top 5 most popular books (most borrowed) along with the number of times each book has been borrowed
SELECT b.bk_title AS book_title, b.author, COUNT(bor.book_ID) AS borrow_count
FROM BOOK b
LEFT JOIN BORROWING bor ON b.book_ID = bor.book_ID
GROUP BY b.bk_title, b.author
ORDER BY borrow_count DESC
LIMIT 5;
```

11.2 **Result:**

Result Grid | Filter Rows: | Export:

book_title	author	borrow_count
The Great Gatsby	F. Scott Fitzgerald	1
To Kill a Mockingbird	Harper Lee	1
1984	George Orwell	1
Pride and Prejudice	Jane Austen	1
The Catcher in the Rye	J.D. Salinger	1

11.3 **Status:** PASS

12. Query:

```
-- Query to get the list of students who have overdue books (books not returned on time):
SELECT s.stud_ID, s.first_name, s.last_name
FROM STUDENTS s
JOIN BORROWING bor ON s.stud_ID = bor.stud_ID
WHERE bor.data_return < CURDATE();
```

12.2 **Result:**

Result Grid | Filter Rows:

stud_ID	first_name	last_name
3	Michael	Johnson
5	David	Brown
8	Jessica	Wilson
2	Jane	Smith
10	Ashley	Anderson
1	John	Doe
6	Sarah	Davis
4	Emily	Williams
9	Daniel	Taylor
7	Christopher	Miller
12	Amanda	Jackson
11	James	Thomas

12.3 **Status:** PASS

13. Query:

```
-- Query to find out the users who haven't borrowed any books
SELECT u.user_ID, u.first_name, u.last_name
FROM USERS u
LEFT JOIN TRANSACTIONS t ON u.user_ID = t.trans_ID
WHERE t.borrowing_ID IS NULL;
```

13.2 **Result:**

Result Grid | Filter Rows:

	user_ID	first_name	last_name

13.3 **Status:** PASS

7. Testing Approach

1. **Unit Testing:** Individual components of the database, such as tables, columns, and constraints, were tested in isolation to ensure their correctness.

2. **Integration Testing:** Data flow and integrity between various database elements were tested, including relationships between tables.

3. **Functional Testing:** The database was tested against functional requirements to ensure it supports the Library Management System's features.

4. **Performance Testing:** Queries were executed under different load conditions to measure database performance.

8. Conclusion

The database testing for the Library Management System has been successfully completed. All essential aspects, including database creation, table structure, data insertion, query execution, data integrity, performance, and backup/recovery, have been verified. The system's database is stable and ready for production use.